Université Paris-Sud

Apprentissage artificiel pour l'ordonnancement des tâches dans les grilles de calcul

Thèse de doctorat

en vue de l'obtention du grade de
docteur de l'université Paris XI
Spécialité : Informatique

par
Julien Perez

sous la direction de
Cécile Germain Renaud
Balázs Kégl

Table des matières

Table des figures

Notations et symboles

$\mathbb{R}$	L'ensemble des réels
$\mathbb{N}$	L'ensemble des entiers naturels
$[m]$	Notation compacte de $\{1, \ldots, m\}$
θ	Ensemble des paramètres d'un modèle de régression linéaire généralisée
ϕ	Fonction de base d'un modèle de régression linéaire généralisée
$\mathbf{w}$	Poids d'un modèle de régression linéaire généralisée
Σ	Matrice de covariance d'une fonction de base gausienne
μ	Moyenne d'une fonction de base gaussienne
S	Espace des états d'un Processus de Décision de Markov
s	Etat d'un Processus de Décision de Markov
A	Espace des actions d'un Processus de Décision de Markov
a	Action d'un Processus de Décision de Markov
R	Fonction de gain d'un Processus de Décision de Markov
p	Fonction de probabilité de transition d'un Processus de Décision de Markov
V	Fonction d'espérance de gain associé à chaque état s d'un MDP
Q	Fonction d'espérance de gain associé à chaque couple état,action d'un MDP
π	Politique de contrôle
$x \in [a, b]$	Intervalle $a \leq x \leq b$
M^T	Matrice ou vecteur transposé
X	Ensemble de vecteurs d'entrée $X = \{x_1, \ldots, x_m\}$
x_i	Vecteur d'entrée
y_i	Valeur cible $y_i \in \mathbb{R}$, ou (en reconnaissance de forme) classe $y_i \in \pm 1$
W^{in}	Matrice des poids d'entrée d'un Echo State Network
W^{res}	Matrice des poids du réservoir d'un Echo State Network
W^{out}	Matrice des poids de sortie d'un Echo State Network
W_i^{in}	I-ème ligne de la matrice des poids d'entrée d'un Echo State Network

Chapitre 1

Introduction

Cette thèse étudie l'application du paradigme de l'apprentissage par renforcement au problème de l'allocation de ressources dans les systèmes de mutualisation des ressources de calcul à grande échelle. Plus précisément, ce travail porte sur la réalisation d'une qualité de service différenciée du point de vue du délai de réponse de l'infrastructure, en particulier pour la prise en charge des jobs à courte durée d'exécution, appelé couramment ordonnancement réactif.

Le partage organisé des ressources est une nécessité fondamentale pour les collaborations scientifiques à grande échelle caractéristiques de l'e-science, qui mettent en place d'immenses ensembles de simulation, correspondant à des durées de calcul de plusieurs dizaines d'années machine, telle que dans la communauté scientifique de la physique des hautes énergies. La réactivité des infrastructures, c'est à dire l'aptitude à répondre efficacement à des requêtes non planifiées, constitue un impératif fondamental pour l'intégration des plateformes de calcul à grande échelle dans l'environnement de travail des utilisateurs. Le besoin de réactivité intervient dans de nombreuses situations concrètes, telle que la demande de calcul urgent, où le délai d'allocation de ressources demandées peut être très restreint, mais le volume de calcul important. Les applications interactives, mettant en jeux une interaction homme-machine où les demandes de représentent des temps d'allocation extrêmement courts en sont un autre exemple significatif [GTO05a, CCD06, CWG09].

Deux approches sont actuellement en compétition pour fournir des infrastructures de calcul partagées à grande échelle, les grilles et les *clouds*. Très sommairement, la première répond au besoin de partage organisé, et la seconde à celui de réactivité.

Dans le modèle grille, les institutions acquièrent des ressources et les mettent à disposition des utilisateurs. Le partage est alors une notion clef [FKT01] :

> "Resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs."

Les grilles de production suivent un paradigme de mutualisation : l'administration de ces plateformes de calcul est non seulement un processus de décision distribué mais aussi largement indépendant. La qualité de service différenciée, quant à elle, a émergé comme une demande majeure dans les usages contemporains des grilles, portée notamment par l'utilisation grandissante d'applications interactives. La grille EGEE [ea05] (Enabling Grid for E-sciencE), cadre applicatif des travaux présentés ici, est une plateforme de calcul mondialement distribuée parmi les plus importantes au monde. Des applications provenant de diverses communautés scientifiques demandent quotidiennement des temps de réponse quasi-interactifs. Une question difficile, et qui n'a pour l'instant pas reçu de solution complètement satisfaisante, est la réalisation opérationnelle des objectifs de qualité de service différenciée dans des systèmes mutualisés.

Dans le modèle récent des *clouds*, des ressources sont louées directement aux utilisateurs. Le point clef est ici la capacité à fournir dynamiquement, à la demande, des ensembles de ressources comme proposé dans les infrastructure *Amazon EC2* [1], *IBM smart businesses* [2] ou *Microsoft Azure* [3]. Les mécanismes sophistiqués nécessaires au partage et au contrôle du partage, en particulier de grandes masses de données, rendent improbables une utilisation directe des clouds par les participants aux grandes collaborations scientifiques. Ces mécanismes constituent la réalisation majeure des intergiciels gouvernant les grilles de production. En revanche, l'exploration de l'utilisation des clouds comme infrastructure des grilles (Infrastructure as a Service, IaaS) est poursuivie activement [Llo09]. Dans cette configuration, la question d'une utilisation optimale de la dynamicité des ressources est posée.

Les travaux présentés dans cette thèse proposent une approche de l'allocation de ressources qui vise à concilier réactivité et partage organisé. L'allocation de ressources doit optimiser la réalisation d'objectifs partiellement conflictuels : la qualité de service différenciée évoquée ci-dessus, l'équité pondérée et l'utilisation. L'équité pondérée et l'utilisation modélisent deux aspects de la volonté de retour sur investissement des institutions participant à la création d'infrastructures de calcul partagées. Motivé par ce but général, dont la portée dépasse l'infrastructure de calcul mentionnée ci-dessus, ce travail se concentre sur le problème spécifique de la gestion de flux de demandes combinant des demandes d'allocation quasi-immédiate de ressources de calcul pour une période de temps limitée et des demandes de ressource de calcul classiques.

C'est dans ce contexte que l'autonomic computing entre en jeu par (1) la formalisation des objectifs de haut niveau, tels que la réactivité, l'équité ou l'utilisation qui doivent être facilement paramétrables par les administrateurs système et (2) le mécanisme d'ordonnancement sous-jacent devant être en mesure de s'adapter seul aux incertitudes de l'environnement et aux tendances de l'usage de l'infrastructure administrée.

Cette thèse propose une approche utilisant l'apprentissage par renforcement (*reinforcement learning, RL*) comme mécanisme unifié d'allocation de ressources, incluant l'ordonnancement et la prise en charge de la dynamique des ressources. La flexibilité d'un système basé sur ce paradigme nous autorise à modéliser l'état des ressources, les jobs à ordonnancer ainsi que les objectifs de haut niveau d'acteurs variés de la grille comme un processus de décision de Markov. Une politique d'ordonnancement basée sur l'apprentissage par renforcement peut adapter ses décisions au changement de distributions des délais d'inter-arrivées, de demandes de qualité de service, ainsi que de ressources disponibles. De plus, cette approche exige une connaissance à priori très minimale concernant l'environne-

[1] http ://aws.amazon.com/ec2/
[2] http ://www.ibm.com/businesscenter/smb/us/en/smartbusiness
[3] http ://www.microsoft.com/windowsazure/

ment cible incluant les requêtes des utilisateurs et les caractéristiques de l'infrastructure supervisée.

Nous avons développé un modèle général d'apprentissage par renforcement incorporant des modèles pour les classes de travaux (actuellement deux, *réactif* et *best effort*), ainsi que pour les fonctions d'objectifs. La première étape a concerné l'ordonnancement uniquement [PGRKL08b, PGRKL08a], sous des hypothèses assez fortes de connaissance complète des caractéristiques du flux des travail. Dans la suite [PGRKL09], nous avons étendu le champ des objectifs de l'apprentissage par renforcement à la fourniture de ressources. Nous démontrons que l'introduction d'un service d'élasticité de l'infrastructure de calcul, c'est à dire l'aptitude à moduler la quantité de ressources mises à disposition pour répondre aux demandes des utilisateurs, est utile pour optimiser une mesure d'utilisation de l'infrastructure administrée. Ces derniers travaux se basent sur des hypothèses réalistes de connaissances incomplètes des caractéristiques du système de calcul supervisé. L'infrastructure de grille EGEE fournira, dans ce travail, à la fois le modèle de grille et les données expérimentales utilisées au cours des expérimentations présentées.

Les questions auxquelles nous tenterons donc de répondre dans ce manuscrit seront les suivantes :

- Dans quelles mesures un processus de décision de Markov est capable de modéliser un problème d'ordonnancement de tâche et de dimensionnement de ressources.
- Quels modèles d'apprentissage, déterministe ou à mémoire, offrent des performances suffisantes, au regard des objectifs de haut niveau que sont la réactivité, l'équité et l'utilisation, dans le cadre des problèmes de décision traités.
- Quelle robustesse attendre des algorithmes d'apprentissage par renforcement face à des flux de demandes chargés ou des spécifications d'objectifs irréalisables.
- Quels apports quantitatifs et qualitatifs attendre de l'apprentissage par renforcement en comparaison aux algorithmes d'ordonnancement classiques.

Les contributions principales de ce travail sont les suivantes :

- Nous proposons une formalisation du problème de supervision d'un site de grille comme deux problèmes d'apprentissage par renforcement multiobjectif, définis dans un espace continu d'état et d'action, sous des hypothèses réalistes.
- Nous explorons plusieurs implémentations du paradigme de l'apprentissage par renforcement intégrant les objectifs de haut niveau énoncés via des modèles de régression statistique déterministes et à mémoire.
- Nous présentons une plateforme de simulation d'ordonnancement de site de grille, développée au cours de cette thèse, capable d'utiliser des charges synthétiques générées suivant des distributions statistiques données de délais d'inter-arrivée et de durées

de service, ainsi que des charges extraites des traces d'activités des sites de la grille EGEE. Cette plateforme permet l'analyse et la comparaison d'algorithmes d'ordonnancement et de dimensionnement de site de grille.
– Nous montrons expérimentalement que notre superviseur, basé sur l'apprentissage par renforcement, atteint un niveau satisfaisant de réactivité, sans dégrader les performances d'équité et d'utilisation, estimées grâce à différentes mesures liées à la satisfaction des utilisateurs finaux et des administrateurs de l'infrastructure prise en charge.

La suite de ce document s'organise comme suit,

Le chapitre 2 présente les bases de l'autonomic computing, le cadre conceptuel du travail entrepris au cours de cette thèse. Les fondamentaux définissant cette approche de la conception de système seront résumés et une brève analyse de plusieurs travaux ayant déjà prouvé la pertinence de cette méthode récente sera proposée. Les questions nouvelles quant à la conception de tels systèmes seront abordés.

Le chapitre 3 revient sur le formalisme de l'apprentissage par renforcement et la recherche de politique de contrôle. Nous focaliserons notre présentation sur le cas de l'apprentissage en domaine continu et sur deux modèles statistiques, déterministe ou à mémoire, utilisables dans ce contexte.

Le chapitre 4 rappellera le fonctionnement général de la grille de calcul EGEE. Nous nous attarderons sur le fonctionnement des ordonnanceurs locaux disponibles pour les sites de calcul la constituant avant de discuter du besoin d'interactivité sur ce type de plateforme. Nous détaillerons ensuite les différents mécanismes de spécification des objectifs d'ordonnancement et les confronterons à la méthodologie que nous proposons. Enfin, nous effectuerons un parallèle entre le besoin d'élasticité dans ce type d'infrastructure et le principe du *cloud computing*.

Le chapitre 5 formalise le problème de l'ordonnancement des tâches dans les sites de grille, ainsi que le problème du dimensionnement dynamique des ressources de calcul disponibles, comme deux problèmes d'apprentissage par renforcement en domaine continu. Nous détaillerons et tenterons de justifier le choix des fonctions d'objectifs utilisées suivant les services que l'infrastructure d'ordonnancement se doit d'assurer.

Le chapitre 6 présente la plateforme de simulation d'ordonnancement de site de grille de calcul développée au cours de cette thèse. Nous détaillons son fonctionnement et les services qu'elle propose. Nous décrirons également le mécanisme de génération automatique de traces d'activités utilisées dans nos simulations d'ordonnancement via des modèles de queue Poisson exponentielle et MMPP (*Markov Modulated Poisson Process*). Nous présenterons enfin le mécanisme d'extraction autonomatique de traces d'activités des logs

réels de la grille EGEE.

Le chapitre 7 détaille et interprète les résultats expérimentaux obtenus à partir de simulations d'ordonnancement basées sur un ensemble de charges synthétiques et réelles. Nous tenterons ainsi de mettre en évidence le gain de l'apprentissage par renforcement pour le problème de l'ordonnancement des jobs et de l'élasticité de l'infrastructure de calcul. La question de robustesse du procédé proposé sera également abordée.

Chapitre 2

Autonomic Computing

La difficulté du processus de maintenance des systèmes logiciels ou physiques actuels s'explique par la complexité des problèmes de configuration, de fonctionnement et d'optimisation qu'ils entrainent. La résolution manuelle de ces problèmes demande des compétences de plus en plus importantes dans des domaines toujours plus variés. Il en résulte une augmentation des ressources déléguées à l'administration de ces systèmes. A l'initiative d'IBM [Hor01], la notion d'*Autonomic Computing* [1]a émergé ces dernières années afin de regrouper sous une même définition les recherches autour de la prise en automatique, par les systèmes eux-mêmes, des problèmes de d'optimisation, de gestion et de maintenance.

La définition d'une politique de contrôle pour l'allocation et le dimensionnement des ressources dans les grilles relève des domaines de l'optimisation et du déploiement. Cette catégorie d'applications est l'un des domaines applicatifs les plus actifs de l'autonomic computing. Ce chapitre présente le contexte de nos travaux. La section 2.1 détaille la notion d'*autonomic computing*. La section 2.2 reprendra les travaux récents dans ce domaine pour l'allocation de ressource en général et l'ordonnancement en particulier. Ces recherches montreront l'importance grandissante des travaux en apprentissage automatique dans les systèmes autonomes. La section 2.3 concluera ce chapitre passant en revue les questions nouvelles liées à la conception de tels systèmes.

2.1 Principes

2.1.1 Origines

L'évolution des systèmes informatiques a ouvert de nouvelles possibilités : le calcul distribué permet d'augmenter la rapidité des traitements, les systèmes d'information sont répartis de plus en plus fréquemment sur plusieurs sites. Ces évolutions vont de pair avec un accroissement de la complexité des systèmes informatiques, lié à l'hétérogénéïté des systèmes mis à disposition. Pour implémenter et maintenir de telles infrastructures, les connaissances nécessaires sont liée à de nombreux facteurs, dont en particulier l'asynchronisme, l'hétérogénéité, les comportements fautifs, et l'information incomplète [RTI02]. En effet, la complexité grandissante des systèmes les rend plus sensibles aux pannes et disfonctionnements : l'analyse des causes des erreurs est plus difficile du fait des inter-dépendances entre les éléments d'un système. Par exemple, la mise à jour d'une partie du système peut entraîner des incompatibilités avec le reste du et les défauts d'un composant logiciel peuvent entraîner une défaillance globale. Par ailleurs, ce phénomène d'inter-dépendance fait de l'amélioration des performances un processus difficile.

[1]la traduction française *informatique autonomique* étant peu usitée, nous conserverons la terminologie anglaise.

Partant de ce constat, J. Kephart et P. Horn, chercheurs d'IBM, ontxper introduit en 2001 le concept d'*Autonomic Computing* [Hor01]. Cette notion a comme but d'unifier l'ensemble des recherches concernant la gestion des systèmes par eux-mêmes. L'objectif à long terme est de concevoir des systèmes autonomes en ce qui concerne les tâches administratives récurrentes, qu'il s'agisse de la configuration, la maintenance ou l'optimisation des performances et du diagnostic de pannes. Le travail de l'ingénieur se bornera alors à définir des politiques de contrôle de ces systèmes à un niveau d'abstraction élevé. Les techniques issues de l'apprentissage automatique jouent un rôle essentiel dans cette discipline [KC03] car l'optimisation et l'adaptation à des situations nouvelles sont au coeur des applications de l'apprentissage automatique. Cette approche se présente comme un nouveau champs d'application pour l'apprentissage automatique, avec des difficultés propres qui rejoignent des thématiques classiques telles que la gestion des très grands espaces de données (réduction de dimension, sélection d'attributs), l'exploration, de manière sécurisée, de nouvelles politiques de contrôle (exploration vs exploitation), la résistance au bruit, etc.

Dans la suite de ce chapitre, nous présenterons une définition communément utilisée d'un système autonome. Nous passerons en revue plusieurs travaux de recherche récents en apprentissage automatique concernant les systèmes autonomes et plus particulièrement les problèmes d'allocation de ressources.

2.1.2 Caractéristiques d'un système autonome

L'état de l'art [NB07] définit l'*autonomic computing* par quatre critères fondamentaux nommés souvent regroupés sous la dénomination *self-** :

- *auto-configuration (self-configuration)* : un composant du système doit s'installer et se configurer automatiquement. Le système l'englobant doit prendre connaissance automatiquement de sa présence et s'adapter à son introduction.
- *auto-optimisation (self-optimisation)* : un système autonome doit être en mesure de s'optimiser seul, étant donné une politique de haut niveau précisant les objectifs de performance. Les difficultés sont d'identifier automatiquement les paramètres pertinents et de les ajuster de manière optimale. Les performances dépendent également de la planification des tâches nécessitant une prévision de la charge de travail. Le système doit intégrer un mécanisme de supervision de fonctionnement afin de remplir ces objectifs.
- *auto-protection (self-protection)* : On distingue ici les deux définitions classiques des atteintes portées à l'intégrité des systèmes d'information. La première concerne les attaques extérieures au système (intrusions et dégradations). Le système doit être capable de détecter ces attaques et de les empêcher. La seconde définition concerne

la prévision des erreurs en cascade responsables de dysfonctionnements généralisés du système considéré.

– *auto-récupération (self-healing)* : le système doit être capable de détecter une situation d'erreur ou critique. Il doit également être capable d'en diagnostiquer par lui-même les causes et d'y remédier dans la mesure du possible ou d'alerter l'administrateur.

L'idée première de l'autonomic computing est qu'un système puisse assurer les tâches récurrentes d'administration propres à son fonctionnement et qu'il puisse assurer ses fonctions de manière optimale avec un minimum d'action extérieure. Le système doit ainsi être conscient de lui-même et de ses composants. Il doit notamment connaître les ressources qui peuvent être partagées ainsi que celles défaillantes. Nos travaux se situent dans le domaine de l'auto-optimisation et dans une certaine mesure de l'auto-configuration.

2.1.3 Apprentissage automatique et autonomic computing

Plusieurs spécificités des applications aux systèmes autonomes sont d'intérêt majeur pour l'apprentissage automatique. En effet, le nombre d'états et le nombre de paramètres intervenant dans ces systèmes est très grand et l'ensemble des données est souvent creux par rapport à l'espace des exemples. La modélisation de tels systèmes est également une tâche difficile. Les techniques employées couvrent un grand pan de l'apprentissage automatique dont notamment l'apprentissage par renforcement [KLM96, SB98] qui sera l'objet de notre travail.

La recherche de fonctions d'objectifs, nommées également fonctions d'utilité [KD07], est une question pratique et théorique majeure pour l'emploi de l'apprentissage par renforcement dans le cadre de l'*autonomic computing*. Cette approche permet de focaliser le problème de conception de ces systèmes sur les besoins des utilisateurs finaux. Cette caractéristique est à mettre en opposition aux systèmes de décisions classiques, couramment nommés systèmes experts, constitués de règles fixes pré-établies gouvernant le comportement qu'un système doit adopté dans chaque situation rencontrée. L'emploi de fonctions d'objectif nous permettra, dans la suite de notre travail, de commander le comportement du système autonome par la combinaison de diverses mesures de performances et de rechercher ainsi une politique de contrôle maximisant les performances de l'infrastructure sur le long terme. Un avantage important de cette approche, vis à vis du formalisme des systèmes experts, est l'impossibilité placer le système autonome dans des états où la sélection de l'action est un problème indécidable [KD07]. Un second avantage est de pouvoir concevoir des systèmes optimisant des objectifs contradictoires et où la construction d'une politique de contrôle à base de règles est difficilement envisageable.

Une dernière question à aborder concerne la confiance à accorder à de tels systèmes. Dans la suite de cette présentation, on observera qu'un système autonome a besoin d'explorer un ensemble de comportements sous optimaux afin d'apprendre une politique de contrôle performante pour un objectif donné. Des mécanismes de calcul d'indice de confiance, notamment basée sur une connaissance préalable de l'environnement dans lequel évolue l'agent décisionnel [CSAW05] ont été proposé. Le principe est d'estimer, au préalable, des sous espaces de l'ensemble des couples d'état de l'environnement et des actions possibles de l'agent conduisant à des situations dangereuses pour lui même ou son environnement et de n'autoriser contrôleur à effectuer l'action désirée qu'avec le concentement d'un opérateur humain. Malheureusement, l'application de telles techniques reste complexe dans le cadre d'environnements peu connus ou difficilement modélisables. La question reste donc encore aujourd'hui posée.

2.2 Autonomic computing pour l'allocation de ressources

Nous présentons plusieurs travaux significatifs récents d'allocation de ressource où la méthodologie employée est basée sur le concept de l'autonomic computing et plus précisement le paradigme de l'apprentissage par renforcement. L'allocation de ressources, dont il est question dans ces travaux, porte dans tous les cas sur la capacité de calcul au sens strict par opposition par exemple aux transferts de données. En revanche, les contraintes et les objectifs sont très différents. Nous présenterons dans un premier temps les travaux de l'équipe de Gerald Tesauro au sujet de la gestion des tâches dans un centre de données accessible par un ensemble de services web. Nous analyserons ensuite les travaux du même auteur sur la régulation de la consommation de courant dans les salles serveur pilotée par le même principe général. Nous présenterons enfin les travaux de David Vengerov l'ordonnancement des tâches sur une machine parallèle.

2.2.1 Ordonnancement des tâches dans un centre de données

Le contexte de ce travail est la gestion des requêtes d'utilisateurs dans un parc de serveurs mettant à disposition des applications accessibles via le web. La finalité de ce travail [TJDB07, TWK05] est d'apprendre une politique de contrôle de bonne qualité sans nécessité de connaitre un modèle explicite du traffic ou de la fonction de performance à maximiser. On souhaite, par la même occasion, restreindre la quantité de connaissances nécessaires au sujet du système physique contrôlé.

Le problème posé est l'allocation de puissance de calcul, par le biais de serveurs dédiés, à un ensemble d'applications mis à la disposition des utilisateurs. Ainsi, le parc de serveurs

est une ressource fixe, à l'intérieur de laquelle l'arbitrage sera effectué entre les différents applicatifs. L'intérêt majeur de ce travail est de montrer la faisabilité de l'apprentissage par renforcement pour la recherche d'une politique de contrôle capable de maximiser une fonction de gain inversement dépendante du temps d'attente des utilisateurs dans ce contexte réel.

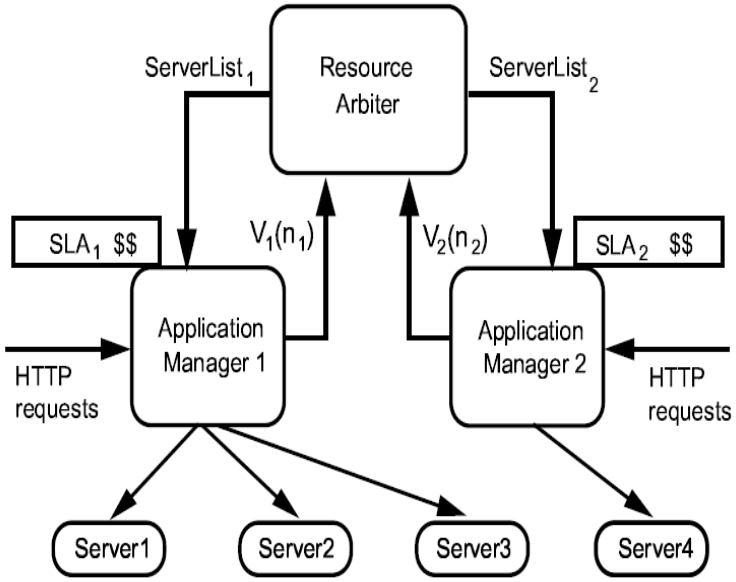

FIG. 2.1 – Architecture du système d'allocation de serveurs proposé [TWK05]

La figure 2.1 présente l'architecture de contrôle en charge de l'allocation des serveurs aux différents applicatifs devant être mis à disposition des utilisateurs finaux. Le système se compose de deux niveaux de contrôle. Tout d'abord, un *Application Manager* affilié à chaque applicatif disponible pour les utilisateurs du centre est chargé de répartir les demandes, pour l'applicatif correspondant, sur les serveurs lui étant affectés. Périodiquement, le *Ressource Arbiter* remet en cause l'affectation des serveurs dédiés à chaque applicatif et les transmet aux *Application Managers*.

Le parc est supposé homogène, le nombre n_i, de serveurs pour chaque applicatif, i, suffit à définir complètement le choix de modélisation. Dans ce modèle, les $V_i(n_i)$ représentent les gains accumulés depuis le dernier choix d'affectation. La définition de l'état se résume au taux d'arrivée à la seconde, λ, de requêtes d'utilisateur depuis la dernière prise de décision. Les valeurs des $V_i(n_i)$ sont calculées en fonction des demandes traitées et du *Service Level Agreement* (SLA) défini pour chaque applicatif. Un SLA représente un gain, en unité économique, associé à chaque exécution d'une demande d'utilisateur, suivant notamment le temps d'attente et la valeur intrinsecte de chaque applicatif. L'objet est de maximiser continuellement la fonction de gain qui correspond à la somme des gains passés. L'article identifie deux limitations majeures d'une stratégie d'apprentissage par

renforcement" : (1) en l'absence de connaissance du domaine ou de bonnes heuristiques, une politique d'apprentissage par renforcement classique est initialement mauvaise (2) le compromis exploration-exploitation contraint le apprenant à choisir des décisions sous optimales tout au long de son exécution afin d'espérer améliorer cette dernière. Afin de pallier aux difficultés d'initialisation du système, précédemment énoncées, politique déterministe est utilisée afin d'apprendre un modèle de contrôle avant de faire appel à un contrôleur par apprentissage renforcement qui sera capable d'adapter son comportement en ligne, tout au long de l'expérience. Cette méthode est appelée, dans ce travail, *hybrid reinforcement learning*. On peut ramener cette approche à un apprentissage par démonstration [CAN08] suivi d'un apprentissage par renforcement classique. L'idée proposée est de ne jamais placer en condition réelle un contrôleur dont les seraient imprévisibles. Pour cela, le modèle apprend une politique en observant le comportement d'un algorithme déterministe choisi qui sera utilisé à la mise en route du système d'allocation. Une fois la politique apprise, on utilise ce contrôleur par apprentissage par renforcement pour répartitir les serveurs du centre de données entre les différents applicatifs à tenir à disposition des utilisateurs.

Dans le contexte de l'allocation de ressources des grilles, plusieurs observations rendront ses propositions difficilement transposables. Tout d'abord, la dynamique des demandes s'avère plus complexe que dans ce cadre de centre de données. L'emploi de processus de Poisson modulés par une chaîne de Markov sera notamment choisi, dans le chapitre 6, pour la modélisation des délais d'inter-arrivée des demandes de calcul sur l'infrastructure de grille. De plus, la modélisation de l'état, bien qu'utilisant un nombre plus important de variables, restera moins parcimonieuse que celle proposée ici. En effet, les apports que nous constateront par l'utilisation d'un modèle de décision à mémoire attesteront de ce manque d'information sur le système administré. De même, l'emploi d'une politique déterministe pour l'entraînement préalable du modèle de décision ne sera pas abordé de part la difficulté à proposer une politique de contrôle déterministe efficace. De plus, le modèle de Tesauro prend pour hypothèse, inopérante dans notre cas, que la taille de l'ensemble des ressources du centre de données est inchangée au cours de l'expérience et que celles-ci sont homogènes. Enfin, comme présenté dans le chapitre 5, le problème d'allocation traité dans cette thèse est multi-objectif.

2.2.2 Régulation du courant sur un serveur

Dans ce travail [TDC+08], l'apprentissage par renforcement est utilisé pour le problème dual de la gestion des performances et de la consommation de courant d'un serveur de calcul. L'auteur applique une méthode d'apprentissage par renforcement en condition réelle sur un serveur exécutant des applications web commerciales. Un contrôleur de fréquence des processeurs est installé dans le système d'exploitation de la machine et le

tout est contrôlé par une politique chargée de maximiser une fonction de gain multi-critère dépendant de la consommation de courant et des performances applicatives du serveur. Les difficultés rencontrées dans ce travail sont le caractère multiobjectif de la fonction de gain à maximiser ainsi que le bruit inhérant aux capteurs, notamment de consommation de courant, utilisés par la politique de contrôle pour choisir ses actions.

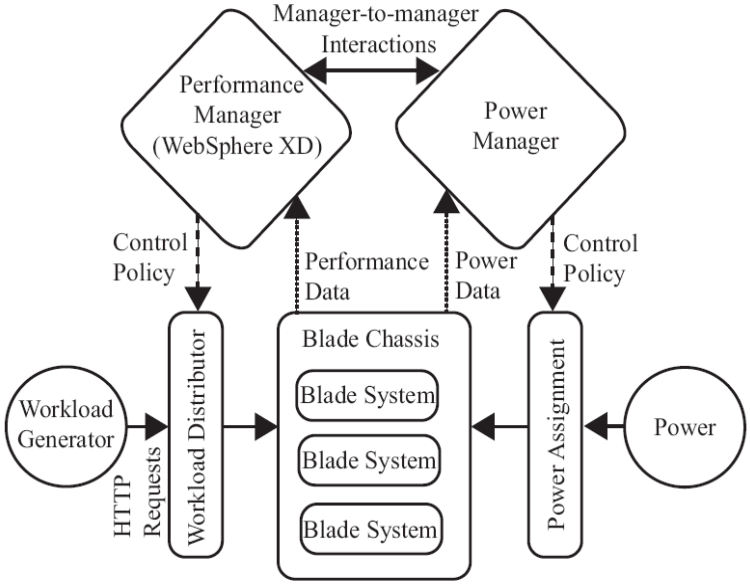

FIG. 2.2 – Architecture du système de régulation de courant mis en place dans (ref)

La figure 2.2 montre le système mis en place pour l'expérimentation. Le *blade chassis* est une armoire physique regroupant plusieurs serveurs IBM de type *Blade*. La politique de contrôle est maintenue par le *Performance Manager*, celle-ci gère la politique de distribution des demandes provenant d'un générateur de charge. Les demandes en question sont des requêtes HTTP. Le *Power Manager*, interagissant avec le *Performance Manager*, est en charge d'augmenter ou de diminuer la fréquence d'exécution des processeurs présents dans les serveurs. Pour ces deux aspects, des données de performance et de consommation de courant sont remontées aux deux contrôleurs et les contrôleurs sont capables de se transmettre leurs données de performance afin d'affiner leur politique de contrôle respective. Dans ce contexte, c'est encore une fois par le paradigme de l'apprentissage par renforcement que la politique de contrôle est apprise, les détails techniques de cette approche seront formalisés dans le chapitre 3.

2.2.3 Ordonnancement sur machine parallèle

David Vengerov et son équipe ont appliqué le problème de l'ordonnancement des tâches sur une machine parallèle[Ven05] sous l'hypothèse de travaux *malléables* (*moldable jobs*)

[CB01, SKS06]. Chaque travail est formée d'un ensemble de tâches, qui peuvent être partiellement ou totalement séquentialisées pour requérir moins de ressources. On note que Jensen [JLT85] fut le premier à proposer de formaliser le problème de l'ordonnancement comme un problème de contrôle en ligne associé à une fonction de gain à maximiser. Cette formulation est une extension du formalisme de l'ordonnancement basé sur une notion d'échéance (*deadline*) où le bénéfice est alors de 1 si le job respecte sa deadline et 0 sinon [JLT85]. Vengerov propose de modéliser le gain associé à la terminaison d'un travail comme une fonction linéaire par morceaux de son temps de présence dans le système et dont les paramètres dépendent de son temps d'exécution présumé. Connaissant cette fonction, il est possible d'en déduire une politique d'ordonnancement optimale de manière gloutonne suivant la connaissance de cette espérance de gain.

Une innovation technique de cette étude est l'utilisation d'une base de règles floues pour l'estimation du gain associé à l'ordonnancement d'un job. Une base de règles floues [KM96, Wan92] est une fonction f qui, à un vecteur d'entrée $x \in \mathbb{R}^k$, associe un scalaire y. Cette fonction est construite à partir de règles floues telles qu'une règle i est une fonction $f_i : \mathbb{R}^k \to \mathbb{R}$ qui associe, à un vecteur d'entrée $x \in \mathbb{R}^k$, un scalaire p_i. Vengerov formule les règles de la façon suivante.

Règle i : SI (x_1 est S_1^i) et (x_2 est S_2^i) et ... (x_k est S_k^i) ALORS (sortie = p_i)

Avec x_j, le j-ième composant de x, S_j^i est un paramète prédéfini de la règle i et p^i sa valeur de sortie. L'expression (x_j est S_j^i) correspond à une fonction d'appartenance $\mu : \mathbb{R}^n \to \mathbb{R}$ qui dépend de la nature de la variable x_j considérée.

Enfin, la sortie de la base de règles floues $f(x)$ est une moyenne pondérée des p^i :

$$y = f(x) = \frac{\sum_{i=1}^{M} p^i w^i(x)}{\sum_{i=1}^{M} w^i} \tag{2.1}$$

avec

$$w^i(x) = \prod_{j=1}^{K} \mu_{S_j^i}(x_j) \tag{2.2}$$

avec M, le nombre de règles de la base. Dans ce contexte, chaque règle floue peut être interprétée comme un intervalle de l'espace avec des bornes floues jointes. Tous les intervalles couvrent l'ensemble des valeurs de x. Lorsque x est observé, dans un intervalle i, la sortie recommandée par la règle i est p_i. La sortie de la base de règle est une somme

pondérée des p_i avec les poids indiquant la distance entre x est le centre de l'intervalle i. Un exemple de règle floue pourrait être :

Si le nombre de processeurs inactifs $x_1 > 10$, alors $p = 0$

A la condition que les fonction μ restent constantes et que les sorties p^i des règles floues soit modifiables, une base de règle floue devient équivalente à une combinaison linéaire de fonctions. Lorsque les fonctions $\mu()$ sont correctement choisies, la formulation précédente est un approximateur universel des fonctions continues réelles à tout degré de précision [Wan92]. Cette base de règle floue est utilisée pour définir une priorité à donner à chaque job et choisir à tout moment le job ayant la priorité la plus élevée à ordonnancer.

Ici, les ressources, c'est à dire le nombre de processeurs, restent fixes tout au long de l'expérience. Ceci constitue une différence majeure avec le contexte applicatif de nos travaux. Pour Vengerov, la question est de définir une partition de l'ensemble des processeurs disponibles pour un sous ensemble des jobs en attente de traitement. L'objectif est alors d'obtenir le meilleur compromis entre le degré de parallélisme effectif, et le délai d'attente avant de lancer le travail. Ceci constitue une différence majeure avec le contexte de nos travaux. Nous nous situons au niveau d'une charge de travail purement séquentielle qui peut d'ailleurs résulter de l'exécution de workflow parallèles.

2.3 Discussion

Les travaux présentés dans ce chapitre nous permettent de dégager plusieurs problématiques récurrentes dans le domaine de la conception de systèmes autonomes :

Tout d'abord, la nécessité de définir formellement les fonctions d'objectif est une première nouveauté dans le processus classique de conception des systèmes d'informations. Cette question peut se rapporter à un problème nouveau de conception logicielle où les mesures de performances escomptées apparaissent explicitement dans le logiciel développé et doivent être décrites sous des formes fermées [Tes07]. La notion d'identification issue de la conception orientée objet évolue ainsi vers une conception *orientée but* où les objectifs des applications développées sont désormais formulés explicitement. Dans certains cas, la combinaison de plusieurs fonctions de gain, nommées parfois fonctions d'utilité [Ven05, TK04], caractérisant des aspects différents d'un problème de contrôle posé, sera nécessaire. On parlera, dans ce dernier cas, de problèmes de contrôle multiobjectifs. Il sera également possible, lorsque le contexte applicatif le permet, de subdiviser ou hiérarchiser un problème de contrôle en sous-problèmes chacun affectés à la maximisation d'une fonction de gain [DCCC06, Die00].

La question du modèle d'apprentissage et de la méthode de recherche d'une politique

satisfaisante font également partie des nouveaux choix indispensables à la construction de ce type de services. L'interrogation se porte alors sur l'adéquation entre le problème posé, la modélisation de la politique de contrôle, et la méthode la plus appropriée pour l'apprentissage d'une politique de contrôle souhaitée. La description de l'environnement et de sa dynamique, ainsi que du champs d'action envisagé et la présence de bruit sont des éléments à considérer dans ce choix. Ces points seront développés dans le chapitre 3, dans le cadre du formalisme des processus de décision de Markov et de l'apprentissage par renforcement.

Une dernière interrogation, classique, porte sur la fiabilité et la sécurité de tels systèmes au champs décisionnel accru. La question est d'établir un protocole de simulation permettant de valider les comportements de ces algorithmes de contrôle par fois non-déterministes, comme nous le présenterons dans un chapitre suivant. La question de l'interprétation des résultats expérimentaux est alors posée, devant, dans ce contexte, aller au delà des mesures de performance synthétiques satisfaisantes sur des problèmes paradigmatiques (benchmarks) [Sut92]. La simulation permettra en particulier d'explorer en détail la dynamique du système. Nous aborderons cette question dans le chapitre 6.

Résumé

Nous avons défini dans ce premier chapitre le paradigme de l'*autonomic computing* justifié par le besoin croissant en personnel qualifié pour la mise en place, l'optimisation et la maintenance des systèmes informatiques toujours plus vastes. L'*autonomic computing* se définit comme approche de conception de systèmes informatiques capables de prendre en charge les différentes exigences de leur cycle de vie en production seule, depuis l'installation jusqu'à l'optimisation et la protection de leur propre intégrité et de celles des données qu'elles hébergent. Pour cela, l'apprentissage articifiel et plus spécifiquement l'apprentissage par renforcement offre un cadre théorique et applicatif pour la conception et l'implémentation de tels systèmes. Nous avons passé en revue plusieurs travaux aboutis de systèmes autonomes dans le domaine de l'allocation de ressource qui sera le coeur de cette thèse. Nous avons terminé cette présentation en mettant en évidence les questions nouvelles suscitées par cette approche récente de la conception des systèmes. Dans le chapitre suivant, nous présentons le formalisme de l'apprentissage par renforcement et plus précisément son application dans le cadre des domaines d'états et d'actions continus qui seront le cadre applicatif de nos travaux d'allocation de ressources.

Chapitre 3

Processus de décision de Markov et apprentissage par renforcement

3.1 Processus de décision de Markov

Le formalisme des processus de décision de Markov, ou MDP selon l'abréviation anglaise de *Markov Decision Processes* [Der70], est une des bases de réflexion les plus utilisées pour l'apprentissage par renforcement. Après avoir présenté les bases théoriques de cette approche, nous nous intéresserons plus particulièrement au cas des problèmes de contrôle dits continus et sans hypothèses de modèles de l'environnement et du gain.

Le formalisme des MDP permet de décrire un problème de contrôle où un agent évolue dans un environnement dans le but de maximiser un gain. Si on reprend l'exemple célèbre présent dans [SB98], un conducteur de voiture (l'agent) doit composer avec les lois de la physique, la carte ainsi que le traffic (l'environnement). Le but de l'agent peut être de minimiser son temps de trajet tout en évitant les accidents. Dans ce but, l'agent suit une politique de contrôle dont les atomes pourraient être : *tourner à gauche*, *à droite*, *accélérer*, *freiner*. L'efficacité d'une politique est définie par un critère de gain associé au problème de contrôle. Apprendre une politique de contrôle qui maximise, à long terme, le critère de gain choisi est le but des algorithmes d'apprentissage par renforcement.

Formellement, un MDP est un quadruplet (S, A, p, r) tel que

- S, l'espace d'état (discret ou continu)
- A, l'espace d'action (discret ou continu)
- p, la fonction de probabilité de transition
- r, la fonction de gain

Alors que la description du temps peut être discrète ou continue, le cas du temps discret sera considéré dans le reste de ce chapitre pour des raisons de simplicité. Par ailleurs, de nombreux problèmes de la vie réelle peuvent être décris dans le cadre du temps discret.

On définit tout d'abord une fonction de probabilité de transition $p : S \times A \times S \rightarrow \mathbb{R}^+$. Plus précisement,pour un état $s \in S$ et une action $a \in A$, $p(.|s, a)$ est la distribution de probabilité sur S décrivant les états suivants que l'on peut atteindre depuis s via une action a. En particulier

$$\forall s, \forall a \in A, \sum_{s' \in S} p(s'|s, a) = 1 \tag{3.1}$$

La fonction de gain r se définit de manière similaire, $r : S \times A \rightarrow \mathbb{R}$. Plus précisement, un état $s \in S$ et une action $a \in A$, $r(s, a)$ est le gain instantané collecté par l'agent choisissant une action a dans l'état s à un instant t. En fonction du problème de contrôle considéré, $r(s, a)$ peut être stochastique également.

La propriété markovienne est une caractéristique importante d'un MDP car les fonctions de gain et de transition dépendent uniquement de l'état et de l'action et non de l'histoire du système (ses actions et états passés). Formellement, un historique h_t dénote une séquence finie de paires d'état/action : $h_t = (s_0, a_0, ..., s_{t-1}, a_{t-1})$. Soit H_t, l'ensemble de toutes les historiques au temps t, la propriété markovienne est définie par

$$\forall t, \forall h_t \in H_t, \forall s_t, s_{t+1} \in S, \forall a_t \in A, P(s_{t+1}|h_t, s_t, a_t) = P(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t) \quad (3.2)$$

Sur ce dernier point, un mécanisme de mémoire sera nécessaire si la propriété markovienne du processus de décision considéré n'est plus assurée. En effet, dans ce cas, la fonction de gain ne dépendra plus seulement de l'état courant mais également de la suite des états passés, rencontrés par le système considéré. S'il on considère un problème de contrôle en environnement complexe, définir un processus de décision de Markov revient, par définition, à admettre que l'on possède une description de l'environnement suffisamment complète pour que celle-ci résume le passé de la simulation. Lorsque cette définition n'est pas vérifiée, il nous faut être en mesure d'inclure les états passés de l'environnement rencontré par l'agent décisionnel dans le processus de prise de décision. Une illustration célèbre est le problème du peigne de Tolmann [Tol32].

3.1.1 Politique de contrôle

La politique de contrôle modélise les décisions de l'agent. Une politique est une fonction π qui associe, à chaque pas de temps t et à chaque état $s \in S$, une distribution de probabilité sur les actions A. Une politique π est dite déterministe si $\forall t, \forall s \in S, \pi_t(s)$ est une distribution de Dirac sur A. Par ailleurs, une politique π est nommée stationnaire lorsqu'elle ne dépend pas du pas de temps t, ainsi $\exists \pi, \forall t, \forall s \in S, \pi_t(s) = \pi(s)$.

3.1.2 Critères de gain

Résoudre un problème de décision de Markov consiste à trouver une politique de contrôle qui optimise un critère de gain donné. Il est donc important de distinguer les critères de gain de ce que l'on nomme le gain instantané, $\forall s \in S, r(s)$. Les critères de gains les plus courants sont brièvement rappelés.

Horizon fini

Dans ce cas, l'intervalle de temps est limité, le temps pour effectuer une tâche est connu à l'avance. A noter que le nombre maximum de pas de temps autorisés pour effectuer une action (l'horizon) est déterminé. Ainsi, le critère à maximiser est l'espérance de la somme des gains sur tous les pas de temps inclus dans l'horizon fini que l'on aura préalablement défini.

$$E\left[\sum_{t=0}^{T-1} r_t\right] \tag{3.3}$$

γ-dégressif

Lorsqu'une tâche n'est pas limitée dans le temps, un second critère couramment employé consiste à diminuer exponentiellement les gains des pas de temps futurs. Cette formulation possède de bonnes propriétés de convergence. Soit $\gamma \in [0, 1]$, le facteur dégressif, le gain est défini par

$$E\left[\sum_{i=0}^{\infty} \gamma^t r_t\right] \tag{3.4}$$

Si l'on considère l'intervalle de définition de γ, ce critère définit également un horizon souple, $T = \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$. Dans ce cas, on parle d'horizon car $\lim_{t\to+\infty} \gamma^t r_t = 0$. La qualification de souple vient de la décroissance de l'impact des gains futurs, le *credit assignement*, sur le choix de l'action présente. Ainsi, les états suivants les plus proches dans le temps auront un impact plus important sur l'évaluation du gain courant.

Total

Le critère dit total correspond au cas limite du critère γ-dégressif où $\gamma = 1$, le critère total est donc défini comme

$$\lim_{T\to+\infty} E\left[\sum_{t=0}^{T-1} r_t\right] \tag{3.5}$$

Evidemment, ce critère fait uniquement sens dans le cas où le MDP considéré est épisodique, c'est à dire qu'il contient des états terminaux.

Moyenne

Il est également possible de s'intéresser uniquement au cas du gain moyen. Plus précisement, on choisit, pour estimer le gain, la moyenne à long terme des gains instantanés suivant une fenêtre de temps définie. Formellement, on définit le gain moyen comme

$$\lim_{T\to+\infty} \frac{1}{T} E\left[\sum_{t=0}^{T-1} r_t\right] \tag{3.6}$$

Dans la suite de ce chapitre, on concentrera notre étude sur les critéres γ-dégressif à horizon fini car il est le plus utilisé et qu'il reflètera le contexte applicatif développé dans les chapitres suivants.

3.1.3 Fonctions de valeur

Une notion fondamentale de l'apprentissage par renforcement est la fonction de valeur $V^\pi(.) : S \to \mathbb{R}$. Cette fonction associe à chaque état du MDP considéré une espérance de gain pour une politique de contrôle π suivie. Cette fonction permet de comparer différentes politiques et est utilisée dans tout les cas où l'on connait la fonction de transition du MDP considéré. Plusieurs définitions, en relation avec les différents critères de gain présentés précédemment, sont à préciser.

Dans le cas de l'horizon fini :

$$V^\pi(s) = E\left[\sum_{t=0}^{T-1} (r_t | s_0 = s, \pi)\right] \tag{3.7}$$

Dans le cas du critère γ-dégressif

$$V_\gamma^\pi(s) = E\left[\sum_{t=0}^{\infty} \left(\gamma^t r_t | s_0 = s, \pi\right)\right] \tag{3.8}$$

On peut également définir la fonction de valeur à tout instant t, dans le cas de l'horizon fini, on obtient :

$$V_{t,T}^\pi(s) = E\left[\sum_{t'=t}^{t+T} (r_{t'} | s_t = s, \pi)\right] \tag{3.9}$$

Dans le cas du critère γ-dégressif :

$$V^{\pi}_{t,\gamma}(s) = E\left[\sum_{t'=t}^{\infty}\left(\gamma^{t'} r_{t'} | s_t = s, \pi\right)\right] \tag{3.10}$$

L'espérance de gain d'un état peut aussi être définie par le biais d'une fonction Q : $S \times A \rightarrow \mathbb{R}$ où :

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} p(s_{t+1}|s_t, a_t) V^{\pi}(s_{t+1}) \tag{3.11}$$

La fonction V représente l'espérance de gain à long terme associée à un état de l'environnement considéré atteint par l'agent et Q représente cette même espérance de gain à long terme mais associée à chaque couple (état, action) pouvant être adopté par l'agent.

L'avantage principal de la fonction Q est d'offrir la possibilité de choisir l'action sans connaissance de la fonction de transition du MDP considéré. Le choix d'une décision optimale, au regard du critère de gain considéré, peut uniquement être basé sur l'optimisation de Q. Dans le cas de la fonction V, l'utilisation de la fonction de transition,p, est nécessaire. Nous verrons dans la suite de ce chapitre que de nombreux cas rendent impossible l'emploi de la fonction V pour l'apprentissage d'une politique de contrôle.

Pour une fonction V donnée, la politique de contrôle π dérivée de V est

$$\forall s_t, \pi(s_t) = \underset{a}{\operatorname{argmax}} \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) V(s_{t+1}) \tag{3.12}$$

Tandis que pour une fonction Q donnée, la politique de contrôle π dérivée de V est

$$\forall s_t, \pi(s_t) = \underset{a_t}{\operatorname{argmax}}\, Q(s_t, a_t) \tag{3.13}$$

3.1.4 Algorithmes d'apprentissage

Considérons le but de maximiser un des critères décrit dans la section précédente. On appelle π^* la politique de contrôle optimale, on note couramment $V^*(s)$, l'espérance de

gain réelle d'un état s, dans un MDP donné. C'est à dire, l'espérance de gain maximale à laquelle peut prétendre une politique dans le MDP considéré. Ainsi,

$$V^{\pi^*}(s) = V^*(s) \tag{3.14}$$

et

$$\forall s \in S, \forall \pi, V^{\pi}(s) \leq V^{\pi^*}(s) \tag{3.15}$$

Trouver π^* ou la fonction associée V^* sont équivalents en principe. Si V^* est donnée, alors

$$\forall s \in S, \pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \left(\sum_{s' \in S} p(s'|s,a) V^*(s') \right) \tag{3.16}$$

En pratique, V^* ne peut uniquement être connu qu'à une approximation près et plus particulièrement dans le cas continu.

Dans les chapitres suivants, on s'intéresse à l'application de ces méthodes lorsque les fonctions de transition et de gain sont inconnues. Ces méthodes sont couramment désignées sous le nom de *hors contexte*.

3.2 Apprentissage par renforcement hors contexte

Ces méthodes ne construisent, à la manière des méthodes de contrôle classique, ni assument, à la manière des méthodes d'approximation de la fonction V, de modèle explicite de l'environnement, ainsi la fonction de transition p et la fonction de gain r ne sont pas explicitement représentées ou utilisées lors de l'apprentissage d'un politique de contrôle. Dans un but pédagogique, nous débutons cette partie en rappelant un algorithme de base, TD(0), assumant cette connaissance et nous verrons comment étendre cet algorithme vers une version ne nécessitant plus une telle connaissance.

3.2.1 L'algorithme TD(0)

L'algorithme de différence temporelle [SB87] permet de combiner une approche de Monte Carlo et une approche de programmation dynamique. Cette algorithme a été largement utilisé pour son efficacité et sa généricité. L'algorithme fonctionne en mettant à jour itérativement les fonctions Q et V, ajustant la fonction selon la prédiction d'erreur.

L'algorithme 1 décrit le plus simple algorithme de Différence Temporelle, référencé comme TD(0) et est utilisé pour apprendre la fonction de valeur, V, d'une politique fixe donnée, π. Pour des raisons de simplicité, on considère le critère γ-dégressif et un espace d'état et d'action fini. En partant d'une connaissance de V arbitraire, l'algorithme affine sa connaissance de V par itération successive pour une politique de contrôle donnée.

Soit π une politique, parmi les paramètres de l'algorithme, on considère une série $\alpha_n : (\alpha_1, \ldots, \alpha_n)$ où chaque α_i règle la taille d'un pas de mis à jour. On peut montrer que V converge vers V^π [Sut88, SB98].

Algorithm 1 TD(0)

Require: un MDP, avec S et A fini
Require: horizon infini, γ-degréssif
Require: une politique π, un état initial $s_0 \in S$, une sequence α_n
1: initialiser $V_0(s)$ arbitrairement
2: initialiser $s = s_0$, $n = 0$
3: **while** vrai **do**
4: choisir une action a, observer le gain r et l'état suivant s'
5: $V_{n+1}(s) \leftarrow V_n + \alpha_n(r - V_n(s) + \gamma V_n(s'))$
6: $s \leftarrow s', n \leftarrow n + 1$
7: **end while**
$V = V_\gamma^\pi$

En pratique, on choisit le plus souvent une série constante, $\forall n \in \mathbb{N}, \alpha_n = \alpha$, où α est une constante petite, $0 < \alpha < 1$. Dans ce cas de figure, l'algorithme s'arrête lorsque la moyenne des n dernières mises à jour de V descend en de ça d'un seuil passé en paramètre de l'algorithme. Dans ce cas, la garantie de convergence n'est plus assurée, l'algorithme se termine dans un minimum local [Tes92]. L'algorithme reste, néamoins, empiriquement efficace dans ce cas [Tes02] [TJDB07].

3.2.2 L'algorithme SARSA

L'algorithme SARSA [Sut88, RN94], présenté dans cette section, est une adaptation de TD(0) permettant d'apprendre une politique optimisant, sans connaissance à priori de la fonction de transition, p, une fonction de gain d'un MDP défini. Cet algorithme que l'on référence aussi sous le nom de *On-policy learning* consiste à mettre à jour itérativement une politique en fonction des paires (état, action) rencontrées. A contrario, dans la méthode nommée *off-Policy learning*, les mises à jour sont indépendantes de la politique de contrôle courante. Dans ce second cas, la politique apprise peut différer de la politique utilisée durant la phase de sélection de l'action.

Formellement, on note π_n une politique agissant suivant Q_n, la connaissance de Q au pas n de l'algorithme. Une méthode couramment utilisée consiste à définir π_n comme une politique ϵ-*greedy* (avec $0 < \epsilon < 1$) de Q_n. Avec une probabilité $(1 - \epsilon)$, la meilleure action au regard de Q_n est choisie, et, avec une probabilité ϵ, au hasard, uniformément sur l'espace des A. Une sélection ϵ-*greedy* des actions sur Q_n garantie que toutes les actions soient sélectionnées un nombre infini de fois avec une probabilité de 1 quelque soit Q_n. Evidemment, une telle politique ne peut être optimale car une proportion non nulle des actions sélectionnées se révèleront être sous optimales. Il a cependant été montré, sous la condition que chaque paire état/action soit visitée un nombre infini de fois et que la valeur de ϵ diminue au cours de l'apprentissage, que SARSA converge vers la politique optimale avec une probabilité de 1 [Sut88]. La question est de choisir la vitesse de diminution de ϵ. En pratique, on conserve ϵ constant avec une valeur petite, dépendante de l'application. Dans ce cas, il a été montré que l'algorithme converge vers une politique non optimale, mais efficace au regard du critère maximisé [Tes92, Tes02, TJDB07].

Algorithm 2 SARSA

Require: MDP, avec S et A fini

Require: horizon infini, γ-degressif

1: Définir une politique, π dépendante de Q
2: Initialiser $Q(s, a)$ arbitrairement
3: Initialiser $s = s_0$, $n = 0$
4: **while** vrai **do**
5: choisir une action a, observer un gain r et le nouvel état s'
6: $a' = \pi_n(s')$
7: $Q_{n+1}(s, a) \leftarrow Q_n(s, a) + \alpha_n(r + \gamma Q_n(s', a') - Q_n(s, a))$
8: $s \leftarrow s', n \leftarrow n + 1$
9: **end while**

$Q = Q^*_\gamma$ si π explore chaque paire état-action un nombre infini de fois.

L'algorithme 2 décrit la procédure d'apprentissage dans le cas fini, en considérant un critère γ-dégressif. En partant d'une politique π_n, dépendante d'une estimation arbitraire de Q_n, l'algorithme effectue une suite d'actions, transitions d'état et collectes de gains immédiats. Itérativement la connaissance de Q est affiné en tout couple (s,a) rencontré. Cet algorithme implique que l'on conserve en mémoire un tableau des couples (s,a) rencontrés au cours des itérations de l'algorithme. Cette méthode suppose également que l'on ait traversé un nombre infini de fois l'ensemble des couples (s,a) de l'environnement exploré afin de converger vers une connaissance exacte de Q, qui nous permet donc d'obtenir une politique de contrôle optimale au sens du critère considéré.

3.2.3 L'algorithme du Q-Learning

On présente ici une variante d'apprentissage de la fonction Q [Wat89]. L'intéret du Q-Learning est sa propriété de convergence avec une probabilité de 1.

Algorithm 3 Q-Learning

Require: un MDP, avec S et A fini

Require: horizon infini, γ-degressif

1: Definir une politique π_n dépendante de Q_n
2: Initialiser $Q(s,a)$ arbitrairement
3: Initialiser $s = s_0$, $n = 0$
4: **while** vrai **do**
5: choisir une action a, observer un gain r et le nouvel état s'
6: $a' = \pi_n(s')$
7: $Q_{n+1}(s,a) \leftarrow Q_n(s,a) + \alpha(r + \gamma \max_{a'}(Q_n(s',a')) - Q_n(s,a))$
8: $s \leftarrow s', n \leftarrow n+1$
9: **end while**

$Q = Q^*_\gamma$ si π_n explore chaque paire état-action un nombre infini de fois et converge vers une politique gloutonne en Q_n

L'algorithme requiert toutefois une tâche de maximisation de Q lors de l'étape de mise à jour de Q qui n'est pas toujours possible en tout point si s et a sont définis dans $\mathbb{R}^n$.

3.3 L'apprentissage par renforcement en domaine continu

Nous consacrons cette partie à la question de l'apprentissage de politique de contrôle dans le cas où les espaces des états et des actions sont continus. En effet, les algorithmes précédemment présentés prennent pour hypothèse que les valeurs des fonctions V et Q

peuvent être conservées dans un tableau. Cette hypothèse devient inopérante dans le cas d'un espace continu, plus encore lorsque s et a sont des vecteurs, comme il en sera question dans notre contexte applicatif des chapitres suivants.

Dans cette section, nous présentons deux approches courantes pour l'utilisation de l'apprentissage par renforcement en domaine continu. La première méthode est utilisée lorsque l'état est défini sur $\mathbb{R}^s$ mais l'espace d'action reste discret. Ce cas est fréquent lorsqu'un agent évolue dans un environnement réaliste mais où ses possibilités d'interaction avec ce dernier sont limitées à un ensemble d'action fini. La seconde approche fera référence à un espace d'état et d'action défini respectivement sur $\mathbb{R}^s$ et $\mathbb{R}^a$. Nous présenterons alors une méthode couramment utilisée de régression de la fonction Q.

3.3.1 Discrétisation de l'espace d'état

Une solution triviale mais efficace consiste à passer d'une représentation continue à une représentation discrète de l'espace des états. Ceci correspond à une forme d'aggrégation où tous les états dans un sous espace du domaine d'état sont regroupés sous la forme d'un état discret. Dans [Doy00], l'état continu du pendule inverse est partitionné en un nombre fini d'états discrets. L'avantage de cette méthode est qu'un algorithme d'apprentissage par renforcement se basant sur une table de triplets (etat, action, gain), comme ceux présentés précédemment, peut être utilisé, mais avec un certain nombre d'inconvénients : si l'espace d'état est un processus de Markov, alors il est possible que la discrétisation rende le processus de décision considéré non-markovien [UV98]. Ce fait a pour conséquence que les preuves de convergence vers une politique de contrôle optimale ne sont plus garanties. Par ailleurs, le mécanisme d'aggrégation réduit considérablement l'ensemble des politiques possibles. La politique optimale pour l'espace d'état/action continu peut donc ne plus être présent dans l'ensemble. Dans ce cas, la politique trouvée peut entrainer de mauvaises performances dans le cas du système continu.

Parce que le choix de la méthode de discrétisation influence le résultat, les algorithmes développés utilisent une aggrégation adaptative. Certaines méthodes sont basées sur un apprentissage non supervisé, comme les k-plus-proches voisins[MHLM09] ou les cartes auto organisatrices [ST94, Smi02]. L'avantages de ces méthodes de quantisation adaptative est qu'elles permettent d'offrir de meilleurs résultats en terme d'apprentissage de politique de contrôle [WCS06, Smi02]. Cependant, indépendamment de la méthode de discrétisation choisie, lorsqu'un large état est scindé, la probabilité d'être dans un des états résultants devient plus faible. La conséquence est que l'estimation de valeur pour ces états perd en fiabilité.

3.3.2 Apprentissage continu

Lorsque les espaces des états et des actions sont continus, l'utilisation d'un tableau collectant les valeurs des triplets (état, action, gain) rencontrés au cours des expériences n'est pas envisageable, le système souffre de ce que Bellman nomma *the curse of dimensionality* [Bel57, Bel61]. On considère donc l'utilisation d'un modèle de régression pour l'apprentissage de la fonction Q d'un processus de décision de Markov. On s'intéresse particulièrement à ce cas de figure car il est le plus souvent rencontré dans les problèmes de contrôle réels. Il correspond, entre autre, au problème de l'ordonnancement des tâches et du dimensionnement des ressources dans les sites de grilles de calcul.

De nombreuses expériences concluantes ont déjà permis d'affirmer la viabilité d'une telle méthode [tH01, BEDB07]. Ainsi, bien que les preuves de convergences de ces méthodes soient encore restreintes à des hypothèses fortes, les résultats sont utilisables [Tes02]. Après avoir rappeler les bases de la régression linéaire généralisée, nous présentons, dans la suite de ce chapitre, deux modèles de régression que nous nous proposerons d'utiliser dans ce contexte.

3.4 Modèles de régression

Lorsque les domaines d'état et d'action ne peuvent être réduit à un ensemble de valeurs discrètes, la fonction Q doit être estimée par le biais d'un modèle de régression. Le choix de l'hypothèse est cruciale et doit correspondre à une nécessite d'expressivité liée au problème de décision traité. L'utilisation de modèles de régression linéaires [SB98, dMBA08, PLT+08] a été proposé à plusieurs reprises. Toutefois, afin de capturer la non linéarité des applications, l'utilisation de réseaux de neurones multi couche [RHW86] a été proposé [Tes02]. Enfin, nous introduirons un modèle de régression non linéaire à mémoire, l'Echo State Network [Jae02], qui nous permettra d'appréhender le caractère non-markovien des processus de décisions étudiés dans la suite de ce travail. La partie expérimentale nous permettra de valider cette hypothèse.

3.4.1 Modèle de régression linéaire généralisée

Un approximateur de fonction linéaire généralisé peut être formellement défini comme un ensemble de fonctions indexé par un vecteur $\mathbf{w}$ de valeurs scalaires nommées poids

$$f_\theta(\mathbf{x}) = w_0 + \sum_{i=1}^{\ell} w_i \phi_i(\mathbf{x}) \tag{3.17}$$

avec ϕ_i, une fonction de base, permettant de s'abstraire du caractère linéaire de l'estimateur, et θ regroupant les poids ainsi que les paramètres éventuels des fonctions de bases utilisées. Un exemple de fonction de base couramment utilisée, avec $\mathbf{x} : (x_1, \ell, x_\alpha)$, est la fonction gaussienne, et μ et Σ préfinis

$$\phi(\mathbf{x}) = exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right) \tag{3.18}$$

La fonction sigmoide est également une fonction de base courante

$$\phi(\mathbf{x}) = \sum_{i=1}^{\alpha}\left(\frac{1}{1+e^{-x_i}}\right) \tag{3.19}$$

La motivation principale de l'utilisation de ces modèles est l'acquisition d'une certaine forme de généralisation. Un exemple classique de généralisation est l'interpolation ou extrapolation de données expérimentales et notamment, dans le cadre de l'apprentissage par renforcement, d'une fonction Q préalablement échantillonnée via un tableau de triplet (états, actions, gains).

Principe de l'apprentissage

Afin d'approximer une fonction cible donnée, il est nécessaire de trouver un ensemble de poids approprié. Le problème est que changer un poid aura tendance à altérer la sortie de la fonction dans l'ensemble de l'espace d'entrée. Une solution possible consiste donc à minimiser une fonction qui mesure la qualité de l'approximation. Dans une majorité de cas, on dispose d'un ensemble fini d'échantillons d'entrée/sortie dit d'apprentissage. Le but est de trouver une fonction qui les approxime correctement tout en assurant une capacité de généralisation. En effet, l'approximation doit être en mesure d'évaluer correctement des points, issus de la fonction cible, mais ne figurant pas dans l'échantillon d'apprentissage.

On nomme ces échantillons $(\mathbf{x}_i, y_i)$ avec $i \in 1, ..., n$. Dans cette situation, l'erreur quadratique peut être utilisée :

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2 \tag{3.20}$$

Entrainer un estimateur de fonction se ramène souvent à trouver un vecteur des paramètres θ composé de $\mathbf{w} = (w_0, w_1, \ldots, w_\ell) \in \mathbb{R}^\ell$, et des paramètres des fonctions de

base choisis, qui minimisent $E(\theta)$. Ce problème d'optimisation est classique et beaucoup de techniques ont été développées afin de le résoudre. L'approche la plus commune est la descente de gradient. Cette méthode consiste à estimer l'*altitude* du paysage de la fonction E en fonction des valeurs des poids $\mathbf{w}$, et, afin de trouver un minimum, descendre suivant le gradient de l'erreur jusqu'à atteindre un minimum. Cette méthode ne convergera pas vers un minimum global de E, mais uniquement vers un minimum local. Dans de nombreux cas, un minimum local est satisfaisant lorsque l'on est en mesure de fournir une valeur initiale de $\mathbf{w}$ correctement choisie. L'algorithme classique permet d'effectuer une descente de gradient. Il consiste à fixer une taille de pas avec un paramètre $\nu \in [0, 1]$ appelé pas d'apprentissage. Les poids ajoutent itérativement la valeur

$$\nabla \mathbf{w} = -\nu \frac{\partial E}{\partial \mathbf{w}} \tag{3.21}$$

On répète cette opération jusqu'à que le critère de terminaison soit atteint.

Algorithme batch et incrémental

Lors d'un apprentissage supervisé, la fonction d'erreur est très souvent définie comme la somme des valeurs d'erreur obtenues sur un nombre fini p d'exemples d'apprentissage composés de couples (entrée,sortie). L'algorithme présenté dans la section précédente est qualifié de *batch* car le gradient de l'erreur est évalué sur l'ensemble complet d'apprentissage. Une autre méthode consiste à modifier les poids directement grâce au calcul de E_i, tel que,

$$E_i(\theta) = \frac{1}{2} \left(f_\theta(x_i) - y_i \right)^2 \tag{3.22}$$

On nomme cette seconde approche l'apprentissage incrémental, elle est également appelée apprentissage en ligne ou apprentissage stochastique.

L'algorithme 4 résume le principe de l'apprentissage batch, on calcule le gradient $\mathbf{g}$ sur l'ensemble d'apprentissage complet avant de l'appliquer aux poids du modèle.

L'algorithme 5 résume le principe de l'apprentissage incrémental. Le gradient est estimé sur chaque exemple et appliqué directement aux poids du modèle.

Le choix de la meilleure technique est une question délicate, la réponse dépend du problème d'apprentissage considéré. Cependant un ensemble de point est à prendre en considération, comme mentionné dans [BLC05].

Algorithm 4 Apprentissage batch

Require: $\mathbf{w}$ aléatoirement initialisé

repeat
  $\mathbf{g} \leftarrow \mathbf{0}$
  for $i = 1$ to p **do**
    $\mathbf{g} \leftarrow \mathbf{g} + \frac{\partial E_i}{\partial \mathbf{w}}$
  end for
  $\mathbf{w} \leftarrow \mathbf{w} - \nu \mathbf{g}$
until critère de fin satisfait

Algorithm 5 Apprentissage incrémental

Require: $\mathbf{w}$ aléatoirement initialisé

repeat
  $i \leftarrow$ valeur aléatoire entre 1 et p
  $\mathbf{w} \leftarrow \mathbf{w} - \nu \frac{\partial E_i}{\partial \mathbf{w}}$
until critère de fin satisfait

Avantages de l'apprentissage batch

1. Les conditions de convergence sont connues. La présence de bruit en apprentissage incrémental entraîne une constante fluctuation des poids autour des minimums locaux et une absence de convergence en une valeur constante stable. Ce phénomène n'apparait pas en apprentissage batch.
2. De nombreuses méthodes d'accélération fonctionnent uniquement dans le cadre d'un apprentissage batch. En particulier, des algorithmes comme le gradient conjugé.
3. L'analyse théorique de la vitesse de converge est plus simple. Ce phénomène est notamment relié à l'absence de bruit dans l'apprentissage batch.

Avantages de l'apprentissage incrémental

1. L'apprentissage incrémental est généralement plus rapide que l'apprentissage batch, spécifiquement lorsque l'ensemble d'apprentissage est redondant. Dans ce cas, lorsque l'ensemble d'apprentissage contient des couples d'entrée/sortie similaires, l'apprentissage batch perd du temps à calculer et à ajouter un gradient similaire avant de mettre à jour les poids du modèle.
2. L'apprentissage incrémental trouve souvent une meilleure solution. La raison est que l'aléatoire intrinsèque à l'apprentissage incrémental crée du bruit dans la mise à jour des poids. Ce bruit s'avère utile pour sortir de mauvais minima locaux [Wat87].
3. L'apprentissage incrémental est en mesure de prendre en considération les changements dans l'ensemble d'apprentissage. Un exemple typique est l'apprentissage

d'une fonction de gain en apprentissage par renforcement. La connaissance de la fonction évolue en fonction des explorations effectuées par l'agent, et ce quelque soit l'algorithme d'apprentissage par renforcement adopté. L'apprentissage en ligne est en mesure d'ajuster plus rapidement sa modèlisation de la fonction cible qu'un apprentissage batch.

Nous présentons désormais les deux modèles paramétriques utilisés dans le cadre de nos problèmes de décision définis dans un espace d'états et d'actions continu.

Limitations

Dans ce début de partie, nous avons concentré notre étude sur les modèles linéaires généralisés, utilisant une combinaison linéaire de fonctions de base éventuellement non linéaires. Nous avons vu que l'hypothèse de linéarité des paramètres nous conduit à un ensemble de propriétés incluant la présence d'une solution optimale accessible. Par ailleurs, l'utilisation de fonctions de base non linéaires nous permet de modéliser arbitrairement les relations non linéaires éventuellement présentes dans les données d'entrée. Néanmoins, la limitation de ces modèles vient du fait que les paramètres des fonctions de base doivent être fixés avant que l'ensemble de donnée d'entrée ne soit présenté [Bis06]. En conséquence, le nombre de fonctions de base nécessaires tend à grandir exponentiellement en fonction de la taille de l'ensemble de données traité.

Deux propriétés des ensembles de données réels nous permettrons de pallier à cette difficulté. Tout d'abord, $\mathbf{x}$ tend à pouvoir se définir dans un espace de dimension inférieur à la dimension de l'espace de donnée considéré. Les réseaux de neurones, qui utilisent des fonctions de bases sigmoidales, peuvent adapter leurs paramètres, tel que l'espace de définition des fonctions de bases. La seconde propriété utile est que la variable cible peut avoir un nombre restreint de dépendance avec les vecteurs propres de l'espace généré par les fonctions de base. Le réseau de neurones exploite cette propriété en choisissant les variables d'entrées auxquelles les fonctions de bases correspondent.

3.4.2 Réseaux de neurones multicouche

Un réseau de neurone multi couche est un graphe de noeuds, appelés neurones, connectés entre eux par des liens pondérés. Ces noeuds et liens forment un graphe acyclique dirigé. Les neurones reçoivent des valeurs d'entrée et produisent des valeurs de sortie. Un réseau de neurone multicouche est un approximateur de fonction paramétrique. Le gradient de l'erreur en fonction des poids des liens peut être calculé grâce à l'algorithme de rétro propagation du gradient [LeC85, RHW86]. On nomme $\mathbf{x}$, le vecteur d'entrée d'un neurone

défini dans $\mathbb{R}$, $\mathbf{y}$ son vecteur de sortie et $\mathbf{w}$, les poids définis dans $\mathbb{R}^d$, la forme fermée associée à un neurone est

$$y = w_0 + \theta\left(\sum_{j=1}^{d} w_j x_j\right) \tag{3.23}$$

avec $\theta : \mathbb{R} \to \mathbb{R}$ une fonction d'activation, couramment sigmoidale :

$$\theta(a) = \frac{1}{1 + e^{-a}} \tag{3.24}$$

La figure 3.1 présente un neurone composant un réseau de neurone multicouche.

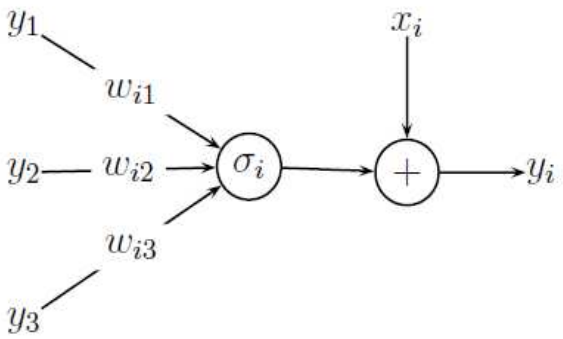

FIG. 3.1 – Neurone composant un réseau de neurone multi couche.

En apprentissage par renforcement, les approximateurs linéaires ont parfois été préférés aux modèles neuronaux car ces modèles sont plus complexes à paramétrer [MS04, MR07]. Par ailleurs, il existe peu de résultats théoriques de convergence au sujet de l'apprentissage par renforcement incorporant l'utilisation d'un réseau de neurones pour l'approximation d'une fonction d'espérance de gain. Cependant, de nombreuses applications ont depuis une dizaine d'années rencontré des succès notables, tel Tesauro [Tes02] et son joueur de backgammon basé sur un réseau de neurones multicouche qui aujourd'hui reste le meilleur algorithme du monde en la matière et fait partie des 10 meilleurs joueurs de la planète aux cotés de champions humains. Une qualité importante des réseaux de neurones est leur capacité à traiter des entrées de grande dimension bruitées. Barron [Bar94] prouva que ce modèle ne souffrait pas de la problématique de dimensionalité, à la différence des approximateurs de fonctions linéaires. Bien que leurs complexités les rendent délicats à manier, les réseaux de neurones multicouche représentent un modèle d'apprentissage efficace. En effet, il possède d'excellentes capacités de généralisation, pouvant aider à résoudre des problèmes d'apprentissage par renforcement continu en grande dimension en terme de description de l'environnement ou de l'action.

3.4.3 Echo State Network (ESN)

Le formalisme des ESN définit à la fois une architecture et un principe d'apprentissage supervisé pour les réseaux de neurones récurrents [Jae02]. En pratique, les réseaux récurrents sont peu utilisés du fait de la faible rapidité de convergence, y compris dans le cas de réseau à faible nombre de neurones. Partant de ce constat, l'idée principale des ESN est d'utiliser un large réseau de neurones récurrent généré aléatoirement et dont la topologie et les poids restent fixes au cours d'une utilisation. Les entrées passent par ce réseau que l'on nomme *réservoir*. Celui-ci produit une réponse non linéaire. Les sorties obtenues sont ensuite combinées linéairement et comparées à la sortie désirée. Les poids de sortie sont ainsi adaptés. Dans ce cas, le nombre de connexions mis à jour durant la phase d'apprentissage est très restreint, accelérant ainsi drastiquement la vitesse de convergence de la procédure d'apprentissage. Les sorties représentent généralement une faible proportion des neurones présents dans le réservoir. Il est également possible d'ajouter des connexions directes entre les entrées et les sorties du modèle.

La figure 3.2 schématise le fonctionnement d'un ESN. Dans certains cas, il est possible que la sortie soit connectée au réservoir. On appelle ce type de connexion, *feedback*, elles sont couramment utilisées lors de l'apprentissage de séries temporelles [SH06]. Les flèches tracées en pointillé à la sortie du réservoir représentent les poids appris par régression linéaire alors que les liens en ligne sont les poids du modèle fixé au démarrage de l'expérience. On note clairement que la proportion des poids appris est minime par rapport au nombre de ceux présents dans le réservoir. Les liens en ligne non pointillés sont les poids du modèle fixés au démarrage de l'expérience.

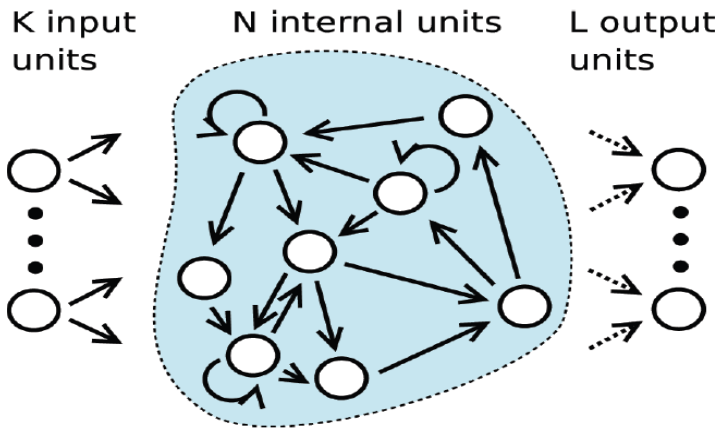

Fig. 3.2 – Schéma général d'un Echo State Network [Jae02]

L'intérêt grandissant pour les ESN s'explique par le fait qu'uniquement les poids de sortie du modèle sont entrainés durant la phase d'apprentissage. Une tâche de régression

s'appuyant sur un ESN peut s'effectuer par un algorithme de régression linéaire, comme présenté dans une sous-section précédente. Ceci, bien qu'un ESN soit en mesure d'offrir une capacité d'expressivité supérieure à un modèle linéaire. Une autre justification de la pertinence de ce modèle a été proposée dans [SS05], où Schiller and Steil ont montré que dans la méthode traditionnelle d'apprentissage des réseaux de neurones récurrents, où tous les poids sont adaptés, les changements dominant s'effectuent majoritairement sur les poids de sortie. L'idée principale de ce modèle est partagée avec celle des *Machines à Etat Liquide* développées indépendamment par Maass [MNM02]. Ces modèles sont fréquemment aggrégés sous le nom de *Reservoir Computing*.

Formalisation

Un ESN est composé de k unités d'entrée, de n unités internes et ℓ unités de sortie. L'activation des unités d'entrée, internes et de sortie à un pas de temps t sont respectivement $\mathbf{x}^t = (x_1^t, \ldots, x_k^t)$, $\mathbf{h}^t = (h_1^t, \ldots, h_n^t)$ et $\mathbf{y}^t = (y_1^t, \ldots, y_\ell^t)$. Les poids symbolisant les liens entre les différentes unités sont conservés dans trois matrices, tout d'abord $k \times n$ poids dans la matrice d'entrée W^{in} qui relie les unités d'entrée x_i aux unités internes h_j de l'ESN ; $n \times n$ poids dans la matrice interne W^{res} qui relie les unités internes entre elles, matérialisant ainsi la récurrence du réseau ainsi que son mécanisme de mémorisation. Enfin, $n \times \ell$ poids de la matrice W^{out} reliant les unités du réservoir h_i et les unités de sortie y_j.

La valeur des unités internes calculée comme suit

$$h_i^{t+1} = f\left(W_i^{in}\mathbf{x}^t + W_i^{res}\mathbf{h}^t\right) \tag{3.25}$$

avec f, une fonction de sortie des unités internes, le plus couramment une fonction sigmoidale. Les sorties sont calculées de la façon suivante

$$\mathbf{y}^{t+1} = W^{out}\mathbf{h}^{t+1} \tag{3.26}$$

Ainsi, les deux principales différences avec un réseau de neurones multi couche classique sont, tout d'abord, que les seuls poids modifiés durant l'apprentissage sont les poids de la matrice de sortie par simple régression linéaire. Les autres poids sont fixés au démarrage de l'expérience et restent inchangés. Ensuite, le réservoir, matérialisé par la matrice W^{res} est spécifié par deux paramètres : ρ le taux de connectivité du réservoir et la plus grande

valeur propre α de la matrice W^{res} nommé facteur d'amortissement [Jae02]. Formellement, les poids du réservoir sont initialisés à 0 avec une probabilité de $1-\rho$. Sinon, ceux-ci sont initialisés à a ou $-a$, où a est tel que α soit en accord avec le facteur d'amortissement prédéfini, généralement $\alpha = 0.95$. Le contrôle de ce facteur permet d'assurer un comportement stable du modèle [Jae02].

Procédure d'apprentissage

La procédure d'apprentissage classique d'un ESN se décompose en trois étapes successives :

Etape 1

Contruire un réseau de neurone aléatoire.

- Créer un réservoir aléatoire, choisir une fonction f définissant le modèle de neurone, tel que linéaire, sigmoïdale ou gaussien. Le réservoir de taille n est dépendant de la tâche. Les exemples de la littérature font fréquemment allusion à un ordre de grandeur d'une à quelques centaines de neurones[MP08, Ven07a].
- Initialiser aléatoirement les poids de la matrice d'entrée W^{in}, ainsi toutes les entrées sont connectées à tous les neurones du réservoir.
- Initialiser aléatoirement les poids de la matrice d'entrée W^{res} en fonction du paramètre d'amortissement ρ prédéfini.
- Initialiser aléatoirement les poids de la matrice de sortie W^{out}. Si la tâche requiert un lien de type *feedback*, définir des poids générés aléatoirement entre la sortie du modèle et le réservoir.

Etape 2

Présenter l'ensemble d'apprentissage D séquentiellement à l'entrée du réservoir. Pour chaque élément de l'ensemble d'apprentissage, on conservera en mémoire l'état des neurones du réservoir, $\mathbf{h}^t$, pour l'élément suivant.

Etape 3

Mettre à jour les poids de sortie par une régression linéaire des poids de la sortie espérée $\mathbf{y}$ dans les états du réservoir $\mathbf{x}$ et utiliser ces poids comme connexion du réservoir vers la sortie du modèle.

Dans le contexte de l'apprentissage par renforcement, l'apprentissage de la fonction d'espérance de gain Q s'effectue en présentant dans l'ordre d'apparition dans la simulation les triplets {état, action, gain} à l'ESN. Il est donc possible de bénéficier de la propriété de mémorisation de ce modèle [MCP+07]. Formellement, le modèle ESN permet de s'abstraire du caractère markovien du MDP considéré, comme montré dans [SGL06]. Concrètement, ce modèle nous permet de prendre un charge une famille de problèmes de contrôle plus générale que les MDP simples, tels que les POMDP (Partially Observable Markov Decision Process) [Son71, KLC98]. L'idée essentielle est l'utilisation d'un modèle à mémoire afin de pallier au manque d'information sur l'environnement. En effet, si on considère un problème de contrôle en environnement complexe, définir un processus de décision de Markov revient, par définition, à admettre que l'on possède une description de l'environnement suffisamment complète pour que celle-ci résume le passé de la simulation (cf. section3.1). Lorsque cette définition n'est pas vérifiée, il nous faut être en mesure d'inclure les états passés de l'environnement rencontré par l'agent décisionnel dans le processus de prise de décisions. Une illustration célèbre est le problème du peigne de Tolmann [Tol32]. Les expériences du chapitre 7 mettront cette difficulté en évidence dans le cadre des problèmes de contrôle abordés.

Discussion

De part le mécanisme de mémorisation présent dans ce second modèle, les états du réservoir reflètent, à tout instant, les traces des entrées passées. Ceci peut être interprété comme une mémoire à court terme [Jae01]. Concernant la propriété d'approximation universelle, les ESNs peuvent réaliser, avec une mémoire bornée, toute fonction non linéaire [MNM02]. L'intuition à la base de l'apprentissage des poids de sortie est que les neurones cachés constituant le réservoir sont en mesure de modéliser une large famille de fonctions. La fonction cible peut ainsi être approximée par une somme pondérée des sorties de celui-ci. L'innovation des Echo State Network et plus généralement du *reservoir computing* est de ramener à une régression linéaire, donc à un problème d'optimisation bien posé, la modélisation de fonctions complexes. Le nombre N de neurones cachés, le ratio de connectivité ρ et le facteur d'amortissement α sont les *hyper-paramètres* de ce modèle.

Le formalisme des Echo State Network a déjà obtenu des résultats satisfaisants pour des tâches comprennant de l'information à caractère temporel, telles que des problèmes de prédiction de série temporelle ou de contrôle non linéaire, exploitant le fait que la mémoire du système est encodée dans l'état des neurones du réservoir. Pour justifier l'emploi de ce type de modèle dans le cadre de l'apprentissage de fonction d'espérance de gain en apprentissage par renforcement, Bush propose une hiérarchisation des problèmes d'apprentissage par renforcement en fonction en fonction de leur difficulté de résolution et de

la définition du problème considéré [Bus08]. La figure 3.3 résume les principaux éléments de cette classification.

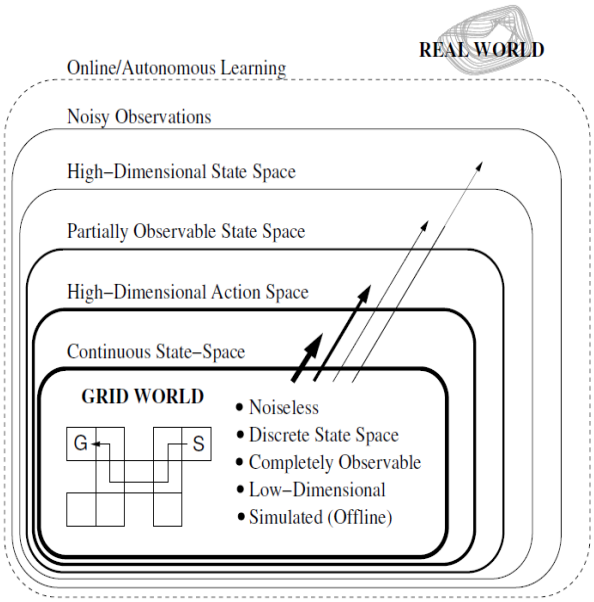

FIG. 3.3 – Classification hiérarchique des processus de décision de Markov proposé par Bush [Bus08]

Après le passage en domaine continu de la description des variables d'état et d'action, l'absence de description complète de l'environnement constitue une difficulté majeure dans cette classification. Dans ce cas, le problème de décision est qualifié de *partiellement observable*. Par définition, la propriété markovienne n'est plus vérifiée dans ce cas car l'état courant ne résume plus l'ensemble des transitions passées d'une expérience donnée (cf. section 3.1). De là, un mécanisme de mémoire s'avére au moins utile voir indispensable pour assurer la production d'une politique de contrôle utilisable [WSY07]. Cette idée a également été proposée par Lorincz et al. [SGL06] et prouvée comme convergente sous des conditions simples.

Résumé

Nous avons détaillé, dans ce chapitre, les éléments fondamentaux du paradigme de l'apprentissage par renforcement. Ce cadre théorique permet la recherche efficace de politiques de contrôle, par le biais d'un agent inter-agissant avec un environnement, dans le but de maximiser une fonction de gain. Nous avons détaillé les différentes formules de gain couramment utilisées et nous nous sommes focalisés sur le cas d'une maximisation de gain γ-dégressif, qui sera le contexte théorique de notre travail sur les grilles de calcul.

Nous nous sommes ensuite intéressés au cas où l'environnement et l'action sont des vecteurs réels, et nous avons introduit plusieurs modèles de régression capable de gérer cette difficulté, le réseau de neurone multi couche et l'Echo State Network. Nous avons terminé notre présentation sur l'apport de l'ESN, comme modèle à mémoire, capable de passer outre une représentation non exhausive de l'environnement nécessitant l'utilisation d'un modèle à mémoire du fait du caractère non-markovien du processus de décision résultant. Dans le chapitre suivant, nous présentons le cadre applicatif de notre travail, la grille de calcul EGEE.

Chapitre 4

La grille EGEE

Nous présentons dans ce chapitre les éléments principaux du concept de grille illustrés par l'infrastructure EGEE, support de notre travail expérimental. Nous préciserons également les enjeux justifiant l'existence de telles infrastructures de calcul. Nous nous intéresserons ensuite plus particulièrement au traitement des jobs sur cette plateforme et au processus d'ordonnancement auquel ceux-ci sont soumis. Nous pouvons débuter notre description par une citation, datée de 1998, de Carl Kesserman et Ian Foster :

une grille computationnelle est une infrastructure matérielle et logicielle qui fournit un accès fiable, largement accessible et économique à une capacité de calcul haute performance.

La principale motivation pour l'implémentation d'une architecture de type grille est la mise en commun de grandes puissances de calcul via un accès standardisé, sécurisé et transparent aux ressources physiques mises en jeu. Techniquement, une grille est un service permettant de partager des ressources de calcul et de stockage de données sur l'Internet. Néanmoins, une grille ne peut être résumée à un ensemble d'ordinateurs partageant de l'information. Les grilles permettent de mettre en commun les ressources et les capacités d'ordinateurs disséminées, auxquelles les utilisateurs peuvent accéder depuis tout point du globe de manière unifiée. Les grilles transforment ainsi des réseaux d'ordinateurs en vastes ensembles de ressources intégrées permettant d'utiliser des applications ayant de lourds besoins de calcul ou de stockage de données. Partant de ce besoin, il a donc été nécessaire de concevoir des systèmes capables de traiter ces gigantesques masses de données alors qu'il n'existe aucun ordinateur central assez puissant pour les prendre seul en charge : La solution a été de faire collaborer des machines de façon décentralisée. Ce modèle de calcul coopératif, a été développé entre autres, dans le projet européen Datagrid [BMM03] et s'inscrit plus particulièrement dans le prolongement des travaux de MetaComputing [SC92].

L'union européenne s'apprête à fédérer toutes ces grilles dans une superstructure, pour cela elle a créé le projet «Enabling grid for e-science » (EGEE) qui rassemble les experts de plus de vingt-sept pays. Le projet EGEE intégre les efforts nationaux, régionaux et thématiques dans une infrastructure pour la recherche européenne. Cette infrastructure est construite sur le réseau GEANT [AAC$^+$08], qui relie les réseaux académiques nationaux ou NRENs (National Research and Educational Network). Actuellement, 33 NRENs sont impliqués, avec des débits variant de 34 Mo/s à 10 Gb/s. EGEE exploite l'expertise acquise par d'autres projets de grille comme EDG et assure l'interopérabilité avec d'autres grilles comme *Cyberinfrastructure*, de la NSF américaine, afin d'établir une grille mondiale.

Après avoir rappelé les éléments principaux de l'organisation globale de l'infrastructure EGEE, nous nous intéresserons plus particulirèrement aux mécanismes de gestion des jobs soumis par les utilisateurs. Nous détaillerons ensuite deux types ordonnanceurs locaux, utilisés dans les sites physiques composant la grille EGEE, Condor[GM08] et

Maui/openPBS/PBS[CDO07]. Ces ordonnanceurs administrent indépendamment les clusters de calcul constituant cette infrastructure de calcul mondiale. Cette étude nous permettra de mettre en évidence deux mécanismes de spécification des objectifs des politiques d'ordonnancement que sont l'*appareillement* et la *prioritisation*. Nous montrerons en quoi ces mécanismes, bien que fonctionnels, ne permettent pas de définir explicitement et de manière non ambigüe les objectifs des politiques d'ordonnancement couramment souhaitées. Nous aborderons ensuite la question de l'intéractivité dans ces infrastructures. Enfin, nous terminerons ce chapitre par une mise en parallèle des notions de grille et de *cloud computing*, nous tenterons de mettre en évidence la complémentarité de ces deux concepts.

4.1 EGEE, Organisation générale

EGEE, la plus grande et la plus utilisée des infrastructures de grille multidisciplinaires au monde, est devenue une infrastructure critique pour un grand nombre de chercheurs européens. De nombreux défis sont associés à ce projet, tels que optimiser et simplifier l'usage de la plus grande grille européenne opérationnelle en l'exploitant de manière continue, en soutenant un nombre toujours croissant de communautés d'utilisateurs d'une part et en conséquence de ressources de traitement et de stockage de données d'autre part. En marge des applications scientifiques, l'organisation d'EGEE étend les services de l'infrastructure à l'industrie. Plusieurs sociétés se sont engagées à ce jour à collaborer avec le projet au titre de premiers associés commerciaux d'EGEE (EBA) afin de rendre l'infrastructure de calcul distribué de la grille plus conviviale, plus efficace et plus sûre dans un contexte industriel.

Les disciplines scientifiques utilisant aujourd'hui les services d'EGEE sont diverses. On compte notamment des travaux en science de la Terre, physique des hautes énergie, bioinformatique et astrophysique. Les utilisateurs d'EGEE sont répartis en organisations virtuelles (VO). Ces entités abstraites, comparables à des groupes d'utilisateurs UNIX, permettent d'agréger également les institutions et les ressources sous un même nom de domaine. On compte à l'heure actuelle près de 150 VOs actives. Matériellement parlant, EGEE compte aujourd'hui 200 sites de calcul en production disséminés tout autour du monde avec un total de 30.000 processeurs et 20 Peta octets de capacité de stockage. Ces chiffres connaissent un accroissement quasi-constant. Chaque site peut choisir quelles organisations virtuelles supportées et quelles proportions des ressources mettre à disposition. Ainsi, les utilisateurs n'ont généralement pas accès à tous les sites de la plateforme EGEE. Ces sites varient beaucoup en terme de capacité de traitement et de stockage. Une petite partie de la grille est par ailleurs dédiée à la mise en préproduction des nouvelles versions

de l'intergiciel GLite [Ba08] de la grille afin d'y être testé par les sites et les utilisateurs. Cette procédure permet de détecter d'éventuels problèmes d'utilisation ou de déploiement.

Indépendemment de la taille des ressources ou de leur localisation matérielle, la grille peut se caractériser par l'agrégation de ressources qui n'appartiennent pas au même domaine. En conséquence, l'architecture d'une grille implique un principe général de subsidiarité : la grille agrège des ressources qui disposent d'une certaine autonomie, en terme d'environnement logiciel et de politique de gestion des tâches et des utilisateurs, et fournit essentiellement avant tout des services de découverte et d'allocation des ressources, avec tout ce qui l'accompagne (sécurité d'une part, surveillance des ressources d'autre part). En particulier, la sécurité et confidentialité, pour les utilisateurs de grille et pour les utilisateurs non grille, doivent être assurées sans compromettre la transparence des accès, en se fondant sur les dispositifs locaux. En conséquence de cela, une gestion centralisée de ce type d'infrastructure n'est pas envisageable.

Par ailleurs, le modèle d'exploitation des infrastructures de l'e-science (centres de calcul haute performance et grilles) reste actuellement dominé par le scénario du *best effort* avec le débit des jobs à forte demande de calcul comme première mesure de performance. Au sein de ces infrastructures, un ordonnanceur batch est utilisé, accompagné d'une étape préliminaire de mise en correspondance des jobs et des ressources dans le cas des grandes grilles de calcul. Les ordonnanceurs batchs satisfont efficacement des contraintes d'équité prédéfinies, au prix d'une configuration logicielle complexe. Bien qu'en principe ces systèmes supportent la réservation de ressources, ce mécanisme est rarement utilisé, à la fois parce qu'il est associé à des problèmes d'utilisation [SCJG00] causés par le besoin de bloquer les jobs de type *best effort* afin de réserver des intervalles de temps pour des réservations futures, mais aussi parce que les réservations à l'avance ne correspondent tout simplement pas aux requêtes d'accès non planifiées. Les ordonnanceurs batchs fournissent des services basiques pour l'exécution immédiate, tel que dans l'ordonnanceur PBS [FMR07]. L'expérience concrète dans un environnement supportant ce mécanisme prouve que les difficultés de configuration impliquées par un tel système limitent sévèrement l'applicabilité de ce service.

4.2 Composants de la grille EGEE

EGEE est contruite autour d'un ensemble important d'éléments logiciels et matériels interdépendants. Au commencement, l'utilisateur interagit avec la grille via une interface utilisateur (UI) lui permettant les actions suivantes :

– Interroger le système d'information pour connaître les ressources disponibles.

- Manipuler des fichiers : les déposer sur des unités de stockage (SE), les enregistrer sur des catalogues et les dupliquer.
- Soumettre des jobs.

Chaque site constituant la grille est en mesure de mettre à disposition une ou plusieurs des ressources suivantes :

- UI, User Interface : interface homme-machine, accessible bien l'Internet, depuis laquelle un utilisateur soumet un job.
- CE, Computing Element : site de calcul comprenant
  - un GK, GateKeeper : élément frontale qui assure la correspondance entre un utilisateur de la grille et un compte disponible sur le CE pour la VO de l'utilisateur.
  - un ou plusieurs WN (Worker Nodes) : machines effectives de calcul.
- SE, Storage Element : élément de stockage. Il s'agit d'un serveur de disques (éventuellement de cassettes).
- MON, Monitoring Node : Noeud de contrôle chargé de l'exécution des applications serveurs du système d'information pour le site, ainsi que le service d'optimisation des duplications de fichiers (ROS, Replica Optimization Service).

D'autres services globaux, non spécifiques à un site, sont nécessaires au fonctionnement de la grille, parmi lesquels :

- RB (Resource Broker) : courtier de ressources, cette machine recherche, sur la grille, l'élément de calcul le plus adéquat pour l'exécution de chaque job, en prenant en compte la proximité des données éventuellement requises.
- RLS, Replica Location Service : ce service assure la maintenance des catalogues contenant la localisation des fichiers originaux et de leurs copies, sur un ou plusieurs éléments de stockage (SE), pour une VO donnée.
- IC, Information Catalog : ce service d'information délivre des informations sur l'état de la Grille concernant son infrastructure et ses applications, principalement utilisées par le RB.
- LB, Logging & Bookkeeping : ce service de journalisation, construit sur une base de données, récupère les événements relatifs à la vie des jobs, tel que la soumission ou le CE d'exécution, et les rend accessibles aux utilisateurs.

4.3 EGEE, l'intergiciel GLite

GLite est le *Workload Management System* (WMS) de l'infrastructure EGEE. Il permet aux utilisateurs de soumettre leurs jobs et effectue toutes les tâches requises pour permettre leur exécution sans exposer les utilisateurs à la complexité de la grille. Il en va

de la responsabilité de l'utilisateur de décrire ses jobs et les besoins de celui-ci en terme de puissance de calcul, de mémoire, ainsi que les fichiers et paramètres requis pour son exécution. Le système sera en mesure de mettre à disposition les fichiers générés par le job une fois celui-ci terminé. Après avoir accepté les jobs utilisateur, GLite est donc en charge de les assigner au site de grille le plus approprié et de retourner les résultats.

Les jobs à soumettre sont décrits en utilisant le *Job Description Langage* JDL [Ba08] qui spécifie quel fichier exécuter ainsi que les paramètres associés et les fichiers d'entrée éventuellement nécessaires. Le processus de choix du CE dans lequel le job sera envoyé pour exécution est nommé *matchmaking*. Il sélectionne dans l'ensemble des CE disponibles, celui qui remplit le mieux les conditions exprimées par l'utilisateur par l'intermédiaire de son fichier de description de job.

```
# hello.jdl
Executable = "/bin/echo" ;
Arguments = " Hello world" ;
VirtualOrganisation = "eo" ;
StdOutput = "hello.out" ;
StdError = "hello.err" ;
OutputSandbox = "hello.out","hello.err" ;
```

Ce script JDL demande l'exécution de la commande *"/bin/echo Hello world"*, et la récupération des sortie et erreur standard dans les fichiers *"hello.out"* et *"hello.err"*. L'organisation virtuelle, *"eo"*, de l'utilisateur concerné, est renseignée. Le paramètre *"OutputSandbox"* renseigne de la totalité des fichiers de sortie du programme qui devront être mis à disposition de l'utilisateur une fois le job terminé. Dans cet exemple, seulement les deux fichiers de sortie standard et d'erreur seront rappatriés de la grille.

Finalement, le LB suit les différents états du cycle de vie des jobs soumis sur EGEE. Il collecte les évenements de tous les composants du WMS et enregistre les différents statuts ainsi que l'historique des jobs. Ce qui vient d'être présenté décrit la part que prend l'intergiciel GLite dans l'ordonnancement, et indirectement ce qu'il ne fait pas. L'intergiciel Glite se charge du placement du travail, logiquement au niveau du CE : une demande de calcul traitée par le WMS aboutit dans une file d'attente d'un site de grille. Après cela, l'intergiciel se borne à surveiller l'évolution du job. Ses seuls interventions auront trait à l'interaction avec les autres couches de l'intergiciel, éventuellement avec l'utilisateur final (par exemple si celui-ci décide d'annuler l'éxecution d'un de ses jobs). L'intergiciel n'est responsable ni du placement final du travail, sur les processeurs, ni de l'ordonnancement temporel des travaux, plus particulièrement de la décision de la date de lancement. Ces deux composantes de l'ordonnancement sont déléguées à la politique de site. Il s'agit donc d'une organisation décentralisée conforme au principe de subsidiarité. Sous gLite, EGEE

combine des ressources de calcul et de stockage distribuées en une unique infrastructure de production disponible pour tous les utilisateurs d'EGEE. Chaque site participant configure et maintien un système de traitement par lot et rend ses ressources disponibles à la grille par le biais d'un service, le *gatekeeper*. La politique d'ordonnancement de chaque site est défini par l'administrateur. Par conséquent, la politique générale d'ordonnancement est à la fois hautement distribuée et hautement variable.

4.4 Cycle de vie d'un job dans l'infrastructure EGEE

Nous énumérerons succintement, dans cette partie, les différentes étapes depuis la soumission d'un job sur l'infrastructure EGEE jusqu'à sa terminaison. A noter que pour chacune de ces étapes, un évènement est enregistré dans le système de *Logging and Book keeping* (LB).

1. Après avoir obtenu un certificat depuis une autorité de certification agrée par EGEE, s'être enregistré dans une organisation virtuelle et avoir obtenu un certificat utilisateur, l'utilisateur est près à utiliser la plateforme EGEE. L'utilisateur se loggue alors sur une machine particulière faisant office de *User Interface* (UI) depuis laquelle il sera en mesure de lancer ses exécutions sur la grille.
2. L'utilisateur soumet un job depuis l'UI vers le WMS, pour mémoire GLite. La description du job à lancer s'effectue dans un fichier via le *Job Description Langage*(JDL). Le statut du job est désormais *SUBMITTED*.
3. Le WMS cherche le meilleur CE disponible pour exécuter le job. Pour ce faire, il interroge le (BDII) afin de déterminer pour chaque CE candidat le statut de ses ressources computationnelles et de stockage ainsi que le catalogue de fichier du site afin de déterminer si tous les fichiers requis par le job sont présents et accessibles. Le statut du job est désormais *WAITING*.
4. Le WMS prépare le job pour submission dans le CE sélectionné, générant notamment un script permettant le lancement du job accompagné des paramètres nécessaires. Le statut du job est désormais *READY*.
5. Le CE reçoit une requête et envoie le job pour exécution dans le *Local Ressource Management System*. Le statut du job est *SCHEDULED*.
6. Le LRMS prend en charge l'exécution du job dans un *Local Worker Node*,une machine physique où l'exécution du job s'effectura. Le statut du job est *RUNNING*.
7. Si le job se termine sans erreur, la sortie est transférée sur le noeud WMS du CE. Le statut du job est *DONE*.

8. A ce point, l'utilisateur peut récupérer les sorties de son job depuis l'UI. Le statut du job est *CLEARED*.
9. A tout moment, l'utilisateur peut envoyer des requètes depuis l'UI sur le LB pour s'informer du statue de ses jobs. il peut également s'informer des statut des ressources par le biais du BDII.
10. Si le site auquel un job est envoyé n'est pas en mesure de l'accepter ou de l'exécuter, le job peut automatiquement être ressoumis à un autre CE qui satisfait les exigences de l'utilisateur. Après un nombre prédéfini de tentatives, le job obtient le statut *ABORTED*. Les utilisateurs peuvent obtenir des informations sur l'historique d'un job via une requête au LB.

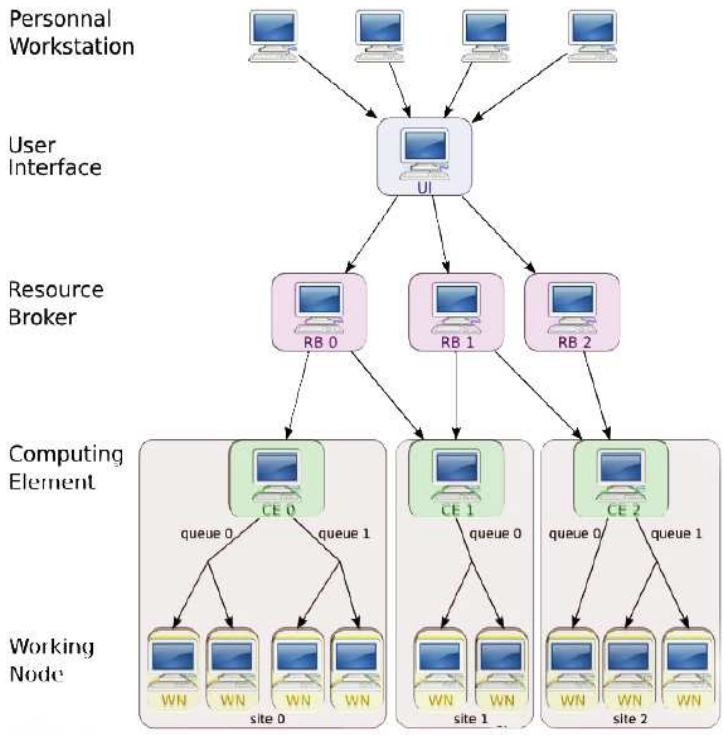

FIG. 4.1 – Soumission d'un job dans la plateforme EGEE

La figure 4.1 résume l'organisation générale des éléments constituant l'essentiel du chemin effectué par chaque job depuis sa soumission sur l'infrastructure par le biais de l'UI jusqu'à sa terminaison. Dans cette thèse, nous nous posons le problème de l'ordonnancement au sein d'un site de grille de calcul (CE). Nous remarquons que cette décision est un élément de la chaîne de décision qui conduit chaque demande de job depuis la machine utilisateur jusqu'au noeud de traitement chargé de son exécution effective. Il aurait également été intéressant de se poser la question de l'ordonnancement au niveau du ressource broker. En effet, celui-ci consiste à l'heure actuelle essentiellement à effectuer un mécanisme d'appareillement entre les ressources disponibles sur chaque CE considéré et

les demandes de l'utilisateur exprimées par le biais de son fichier de description de job. Les notions de concurrence ou d'évaluation des performances des décisions prises à ce niveau ne sont pas utilisées à l'heure actuelle.

Dans la suite de ce chapitre, nous présentons deux mécanismes d'ordonnancement pouvant être utilisés sur un site de grille. Nous focaliserons notre étude sur leurs mécanismes respectifs de spécification des objectifs d'ordonnancement.

4.5 Ordonnancement par appariement, Condor

Le projet Condor [LLM88, LL90] a développé un système d'ordonnancement de jobs dont l'objectif originel était d'utiliser les quanta de temps de calcul d'ensemble de machines de bureau libres afin de constituer une puissance de calcul utilisable. Ce système est aujourd'hui également utilisé pour le contrôle d'ensembles de machines pour le calcul haute performance, le concept d'ordonnancement sous jacent est appelé, appariement ou *matchmaking* [PL95]. Ce système indépendant de l'intergiciel GLite est ainsi utilisé comme ordonnanceur local au sein de certains CE composant EGEE. Les défis rencontrés sont de plusieurs ordres, le premier étant la mise en relation des ressources nécessaires avec les jobs en attente sur le CE administré. Tel, au niveau du RB, chargé de définir à quel CE affecté chaque job, l'ordonnanceur d'un CE doit vérifier les compatibilités entre systèmes d'exploitation et logiciel requis par chaque job en attente. Par ailleurs, l'outil est capable d'orienter la politique d'ordonnancement grâce à un langage de demandes et de préférence quant à l'utilisation des machines et au placement des jobs. Dans le cadre de l'ordonnancement au sein d'un CE, on admet communément que les ressources du parc sont quasi homogènes. Concrêtement, la politique adoptée au sein des sites est de type FIFO avec prioritisation éventuelle de certains profils particuliers de job que nous allons aborder.

Afin d'exprimer une politique de contrôle qu'il souhaite voir appliquée, dans le cadre de ce type d'ordonnancement, l'administrateur du CE peut définir pour chaque machine du cluster ses préférences en terme d'affectation des jobs. Un parallèle peut être fait avec le Job Description Langage utilisé par l'utilisateur de la grille pour renseigner ses jobs. L'administrateur d'un CE définit des recommandations, nommées *ClassAd*, pour *Classified Advertisement* [LBRT97], par exemple :

- **Demande** des jobs sans activité clavier
- **Prefere** des jobs de l'organisation virtuelle Y
- **Interdit** l'exécution des jobs de l'utilisateur W

Parallèlement, les requêtes des utilisateurs sont transcrites dans ce language :

- **Demande** une plateforme Linux
- **Prefere** une machine avec 2 Giga octects de mémoire vive
- **Prefere** une machine placé dans le CE *X*

Concrêtement, l'expression suivante présente une suite de demande pour un job :

```
( ( target.Memory > 8192 ) && ( target.Arch == "INTEL" )
&&
( target.OpSys == "LINUX" ) && ( target.Disk >= 10go ))
```

L'expression suivante est extraite d'une déclaration de machine constituant le cluster de calcul administré dans ce langague *ClassAd* :

```
((Architecture = "INTEL") &&
(OpSys = "RH 6.1") &&
(PhysicalMemory = 256) &&
(LocalDiskSpace = 100) &&
(TotalCPUs = 4) &&
(TotalJobs = 15) &&
(AuthorizedUser = "XYZ") &&
(RunTimeEnvironment = "CMS3.2"," EO4.2"))
```

Le fichier descriptif renseigne sur les caractéristiques physiques de la machine du cluster ainsi qu'un ensemble d'utilisateurs ayant l'autorisation de l'utiliser pour l'exécution de leurs jobs. Il est également possible d'utiliser le champs *AuthorizedVO* afin de spécifier un ensemble de VO autorisées à utiliser une machine donnée.

Au niveau des utilisateurs de l'infrastructure EGEE, ce language n'est pas directement utilisé car, par souci d'interopérabilité entre les utilisateurs et les sites, le fichier rédigé en JDL par les utilisateurs est traduit par un module de l'intergiciel GLite en *ClassAd* si le CE affecté pour le job est ordonnancé par Condor.

Grâce à ces directives, l'administrateur peut exprimer une politique de contrôle en accord avec un ensemble d'objectifs tels qu'une équité prédéfinie entre différentes organisations virtuelles ou encore une prioritisation des jobs d'une durée d'exécution inférieure à un seuil de temps prédéfini. Cependant, le système ne définit pas une forme fermée de fonction de gain à optimiser. Si on souhaite définir une priorité pour les jobs courts et leur éviter un temps d'attente trop important en queue, une politique possible mais sous optimale serait de forcer un sous ensemble des machines du cluster à ne traiter que les jobs répondant à ce critère. De même, si l'administrateur d'un CE souhaite définir une parité entre plusieurs organisations virtuelles, il doit, encore une fois, forcer un ratio de ces machines à ne traiter que les jobs provenant d'une VO donnée. Ces choix sont sous-optimaux car le système ne prend pas en considération les fluctuations éventuelles du flot

de demandes. Dans ces conditions, il est possible qu'une suite de plusieurs jours se produisent sans demande provenant d'une organisation virtuelle à laquelle on aurait affectué un sous ensemble du cluster. Ce problème se manifestera également dans le cas de la prise en charge préférencielle des jobs à faible durée d'exécution par le biais de ce mécanisme d'affection de machines.

Enfin, dans un cadre multi objectif, il est très difficile d'exprimer une politique capable de pondérer exactement des objectifs prédéfinis. Les politiques engendrées par ce langage de commande sont donc dans de nombreux cas, tels que ceux énumérés ici, sous optimales et non adaptatives.

4.6 Ordonnancement à prioritisation, Maui

Maui est un logiciel ordonnancement à code source ouvert pour les cluters et les super calculateurs. Il permet la mise en place de politiques d'ordonnancement, de réservation de ressources et d'équité. Il est actuellement utilisé dans une majorité des sites constituant la grille EGEE ainsi que dans de nombreuses institutions publiques et privées à travers le monde [JSC01, Jac02].

La spécification des politiques d'ordonnancement dans ce système se base sur un mécanisme de prioritisation. Ce procédé permet de déterminer quels jobs privilégier à chaque décision d'ordonnancement et selon quels critères par l'intermédiaire du calcul de sa priorité. Dans le cas de l'ordonnancement de grille, un site se doit le plus souvent de satisfaire plusieurs objectifs : la maximisation de l'utilisation de l'infrastructure de calcul, la mise en place de préférences pour des utilisateurs ou des groupes d'utilisateurs donnés ou s'assurer qu'aucun job ne reste en queue plus d'un certain temps. L'approche utilisée par Maui consiste à accorder des poidss à différents descripteurs de jobs et de ressources prédéfinis et d'assigner des priorités à certaines valeurs de ces descripteurs afin d'en déduire une priorité à utiliser au moment du choix des jobs à ordonnancer [Jac04]. Le calcul des priorités dans Maui permet de pondérer des composants associés à des aspects différents des jobs traités. Le but est de permettre la spécification de priorités à grain fin. Le calcul de la priorité p de chaque job s'effectue par sommation des valeurs des différents composants décrivant le job selon la formule

$$p = \sum_{i=1}^{n} C_i \sum_{j=1}^{m} S_j V_j \tag{4.1}$$

Avec $C \in \mathbb{R}$ le poids attribué à un composant, tel que la part de l'équité ou de l'utilisateur dans le calcul de la priorité. $S \in \mathbb{R}$ correspond à un sous composant tel que

l'identité d'un utilisateur. $V \in \mathbb{R}$ est la priorité associée à la valeur d'un sous composant considéré, telle que la priorité d'un utilisateur nommé *john*. Dans la suite de cette section, trois composants et sous composants principaux du système sont décrits.

4.6.1 Prioritisation par profil utilisateur

Un premier composant du système permet à l'administrateur d'un site de définir des priorités basées sur l'identité des utilisateurs ou des groupes d'utilisateurs des jobs.

Le calcul des priorités se calcule selon la formule

```
p += CREDWEIGHT . (
USERWEIGHT . JobUserPriority +
GROUPWEIGHT . JobGroupPriority +
CLASSWEIGHT . JobClassPriority)
```

Le fichier de configuration suivant donne un exemple de configuration de priorité basé sur ce mécanisme.

```
CREDWEIGHT 1
USERWEIGHT 1
GROUPWEIGHT 1

USERCFG[john] PRIORITY=2000
USERCFG[paul] PRIORITY=-1000

GROUPCFG[staff] PRIORITY=10000
```

Dans cette configuration, le poidss attribué à cet aspect de la prioritisation d'un job est de 1 (¨*CREDWEIGHT 1*¨). Le poids attribué à la prioritisation de l'utilisateur est de 1 également (¨*USERWEIGHT 1*¨) et de même pour le groupe (¨*GROUPWEIGHT 1*¨). Ensuite, des priorités particulières sont spécifiées pour les utilisateurs *John* et *Paul*. Une priorité particulière a également été affectée au groupe *staff*.

4.6.2 Prioritisation pour l'équité

Le composant d'équité du mécanisme de prioritisation permet de favoriser des jobs selon l'historique d'utilisation du site. La prioritisation peut être basée sur l'historique d'utilisation des machines par les utilisateurs et les groupes. Le calcul de la priorité est

```
p += FSWEIGHT . (
FSUSERWEIGHT . DeltaUserFSUsage +
FSGROUPWEIGHT . DeltaGroupFSUsage +
FSACCOUNTWEIGHT . DeltaAccountFSUsage +
FSQOSWEIGHT . DeltaQOSFSUsage +
FSCLASSWEIGHT . DeltaClassFSUsage)
```

Toutes les variables terminant par *'WEIGHT'* sont spécifiées indépendamment des utilisateurs, *FSUSERWEIGHT*, et des groupes, *FSGROUPWEIGHT*, elles permettent de définir une priorité globale sur un aspect d'un job tel que l'identité du propriétaire, des groupes ou des classes d'utilisateurs. Les variables *'Delta*Usage'* sont à définir pour chaque groupe et utilisateur afin de définir leur part d'activité de l'infrastructure à leur accorder. Ce mécanisme permet d'influencer la distribution des groupes ou des utilisateurs des jobs exécutés sur le cluster administré. Toutefois, ce mécanisme peut favoriser mais en aucun cas imposer la distribution des jobs exécutés sur le cluster administré.

4.6.3 Prioritisation par requêtes de ressources

Pondérer les priorités des jobs en fonction de la quantité de ressource demandée permet au site de favoriser certains profils de job. Ce type de priorité peut, par exemple, indirectement permettre d'améliorer l'utilisation ou l'équité du site.

La prioritisation basée sur la demande de ressources est utile dans un ensemble de cas types :

1. lorsque la favorisation des jobs à lourde demande de ressources est une priorité de la politique globale suivie par le site.
2. lorsque un objectif du site administé consiste à favoriser les jobs à faible temps de réponse qu'il est possible de détecter par leur faible demande de ressources.

Le calcul de cette partie de la valeur de priorité de chaque job est définie selon la formule suivante :

```
p += RESWEIGHT . (
NODEWEIGHT . TotalNodesRequested +
PROCWEIGHT . TotalProcessorsRequested +
MEMWEIGHT . TotalMemoryRequested +
SWAPWEIGHT . TotalSwapRequested +
DISKWEIGHT . TotalDiskRequested)
```

Encore une fois, les variables terminant par *'WEIGHT'* sont spécifiées indépendamment des jobs. Les variables de type 'Total*Requested' sont les demandes effectuées par chaque

job. A noter, que les poidss peuvent être définis dans l'intervalle $[0, 1]$ permettant de rendre inversement proportionelle la demande de ressources et la priorité affectée aux jobs. Les variables considérées sont respectivement, le nombre de noeuds de calcul souhaité, le nombre de processeurs, la quantité de mémoire vive, la quantité de mémoire swap, la quantité de disque de stockage.

Dans ce second cas, des politiques plus fines peuvent être spécifiées par l'administrateur système. Cependant, ce mécanisme de déclaration des priorités, très proche du fonctionnement concret de l'ordonnanceur rend rapidement difficile la déclaration de politique d'ordonnancement de haut niveau, notamment lorsque le nombre d'organisations virtuelles, d'utilisateur et de profil de job, courts et longs sont à prendre en considération. La difficulté majeure de ce mécanisme ainsi que de la méthode précédemment présentée est l'impossibilité de définir explicitement les fonctions d'objectifs que le système se doit de maximiser et ceux, indépendamment du système administré ou des caractéristiques du flux de demandes rencontrés.

4.7 Grilles de calcul et interactivité

On peut distinguer deux types d'interactivité, reliés aux temps de réaction attendus par les utilisateurs. Les utilisateurs exigent un temps de réponse faible de la part des applications interactives comme les outils de visualisation scientifiques, les éditeurs d'images, les outils de modélisation basés sur des simulations physiques et les jeux. De telles applications réagissent à l'arrivée apériodique de messages générés par les actions externes. En réponse à chaque message, l'application exécute une tâche qui produit un retour visuel destiné à l'utilisateur. Ce retour concourt à déterminer les actions ultérieures de l'utilisateur, et ainsi, les tâches subséquentes. L'apériodicité de l'arrivée des messages est le résultat des actions humaines dans la boucle de contrôle. Plus précisément, on peut distinguer deux situations liées aux attentes de l'utilisateur concernant les temps de réponse dans un contexte interactif :

- L'interactivité au sens faible : la visualisation des résultats, par exemple à travers une interface Web, et les tests précédant les travaux à effectuer.
- L'interactivité au sens fort : la réalité augmentée ou virtuelle et la conduite des calculs informatiques, où l'utilisateur n'attend pas la fin des calculs, mais vérifie leur pertinence et modifie les données durant leur execution.

4.7.1 Qualité de service

Les systèmes décrits dans les deux parties précédentes ne permettent pas de spécifier explicitement une qualité de service, ils correspondent donc à une vision sans garantie (*best effort*) des systèmes par lots classiques. Quelques tentatives ont eu lieu dans le cadre de Globus pour implanter un mécanisme de réservation des ressources. Dans le modèle de production décrit par [Pa04], le point essentiel est l'introduction de la description de la qualité de service comme une des caractéristiques possibles d'une requête d'exécution d'un job. Certaines propositions ont ciblé la réalisation de cette qualité de service par une négociation dynamique entre les entités contribuant à l'ordonnancement sur grilles. Il s'agit d'apparier les ressources que gère une entité d'ordonnancement avec les requêtes en décrivant des contraintes de qualité de services souples. Par exemple, une fenêtre temporelle d'exécution plutôt que des valeurs fixes qui correspondraient à la réservation. L'objectif de la proposition est la définition d'un formalisme (*Service Level Agreement*, SLA), permettant de spécifier les protocoles de négociation relatifs à la qualité de service. La fonctionnalité et l'implantation de ces protocoles restent toutefois une question ouverte.

Les travaux sur la réalisation de la qualité de service sur grille sont encore assez peu nombreux. GARA [FFR+04] vise à exposer l'information rassemblée par un environnement de surveillance (*monitoring*) et de prédiction de performance comme NWS [WSH99]. Au niveau des algorithmes d'ordonnancement, des heuristiques sont développées dans un contexte multiagents, sur la base du raffinement dynamique de solutions initiales basées sur des informations inexactes.

4.7.2 Structure de la charge de travail

Une application interactive fournit une borne supérieure du délai d'exécution (*deadline*), ou de débit en calculs élémentaires, que l'environnement de grille doit satisfaire. De ce point de vue, l'ordonnancement des applications interactives sur grille relève de l'ordonnancement temps réel sur multiprocesseurs. Le problème est en réalité plus complexe, à cause de la structure des applications interactives d'une part, et leur coexistence avec des travaux de type lots d'autre part.

- Les applications interactives doivent partager les ressources de la grille avec d'autres applications, qui n'ont pas de contraintes de deadline.
- Le niveau intra-requête, qui correspond à l'exécution parallèle des composants d'une application interactive, présent dans certaines applications de visualisation avancée [Bed00], est d'un grain trop fin pour être pris en compte par les mécanismes d'allocation globaux.

4.8 Grilles de calcul et Cloud computing

Une grande partie des données expérimentales ainsi que des motivations pour ce travail proviennent du monde des grilles de calcul. Quel sera l'impact de l'apparition des modèles de production data-center et clouds sur les infrastructures de grilles. Dans les limites de ce document, nous ne pouvons qu'effleurer cette question ; une étude récente [FZRL09] présente une argumentation approfondie pointant plutôt vers la convergence de ces deux modèles. De façon très générale, la sociologie et l'économie de l'e-science rendent peu probable un modèle d'exploitation où les utilisateurs finaux embarqueraient leurs tâches de calcul dans des images sur un pool de ressources donné utilisable de manière indifférenciée. De ce point de vue, plusieurs projets en cours (par exemple StratusLab) s'orientent plutôt vers le modèle *Infrastructure as a Service* (IAaS), avec la migration d'un intergiciel (*middleware*) offrant des fonctionnalités sophistiquées d'authentification, de contrôle de partage et de protection des données, sur une infrastructure virtualisée.

Pour le problème considéré ici, qui est la réactivité, l'impact de la virtualisation, qui est la technologie première de soutien de mise en fonction des clouds, est considérable. La virtualisation apporte un avantage majeur en comparaison des grilles : la "quantification" des allocations. Un job soumis sur une infrastructure de grille doit s'exécuter jusqu'à sa complétion, sauf si un mécanisme de "checkpoint" ou point d'arrêt, très rare, est mis en place. Avec capacité de suspendre, migrer et reprendre les tâches de calcul encapsulées dans des machines virtuelles, le processus d'allocation de ressources peut être scindé en quanta de temps. On pourrait alors envisager de traiter le besoin de réactivité en adaptant des méthodologies de temps réel souple (*soft real time*), allouant des quanta de temps afin de respecter les deadlines imposées avec une erreur bornée. Néanmoins, même pour les tâches périodiques à temps d'exécution connu, la coexistence de stratégies de type *best effort* et temps réel reste une question ouverte. Comme on le verra dans les chapitres consacrés aux expériences, les processus d'arrivée et de service sur une grille sont fort loin d'être aussi simples, mettant en doute la capacité de produire un ordonnanceur généraliste a priori même dans un contexte virtualisé. On peut cependant noter que des travaux récents [PH08] utilisent une boucle de rétroaction afin d'orienter l'allocation des machines virtuelles dans ce cas. Notons enfin que cette problématique sera probablement complètement renouvelée par la banalisation des GPU, dont les relations avec la virtualisation sont encore inexplorées. Dans la suite de ce travail, nous proposerons un mécanisme de gestion des ressources d'un site de calcul administré de la grille EGEE permettant d'optimiser l'utilisation de ce dernier par la mise à disposition de quantités de ressources adaptées à la demande.

Résumé

Dans ce troisième chapitre, nous avons présenté le cadre applicatif de notre travail expérimental, la grille de calcul EGEE. Cette plateforme de calcul haute performance distribuée tout autour du globe offre la possibilité à plusieurs dizaines de communautés scientifiques diverses d'effectuer des expériences où la puissance de calcul ainsi que l'espace mémoire nécessaire est trop important pour être effectuée sur une machine simple. Nous avons présenté le fonctionnement hiérarchique de gestion des demandes utilisateurs et nous nous sommes intéressés plus spécifiquement au fonctionnement de Condor et Maui, deux ordonnanceurs de site interagissant avec l'intergiciel d'EGEE, GLite. Nous avons mis en évidence les difficultés inhérentes de ces systèmes à produire des politiques de contrôle capable d'optimiser les objectifs de haut niveau des administrateurs et des utilisateurs de l'infrastructure. Nous avons mis en évidence, dans un premier temps, que les choix de conception quant à l'édition des préférences des administrateurs des sites composant la grille et des utiliseurs ne permettait de spécifier exactement ces objectifs. Nous avons montré dans un second temps que la spécification de priorités ne parvient également pas à satisfaire ce besoin. Nous nous sommes enfin intéressé à deux questions fondamentales de ce travail, l'interactivité dans les infrastructures de grille et le dimensionnement dynamique des ressources mises à disposition. Dans le chapitre suivant, nous formalisons le problème de l'ordonnancement des tâches dans un site de grille ainsi que le dimensionnement de celui ci comme deux problèmes d'apprentissage par renforcement et proposons une méthode de résolution via l'apprentissage de politiques de contrôle en domaine d'état et d'action continu.

Chapitre 5

Formalisation du problème de l'ordonnancement des tâches dans les grilles de calcul

5.1 Introduction

Dans ce chapitre, les deux problèmes de contrôle abordés au cours de cette thèse sont formalisés par le biais du paradigme de l'apprentissage par renforcement. Une première partie sera consacrée à l'ordonnancement des tâches dans les grilles de calcul haute performance puis, une seconde partie traitera du problème de l'allocation dynamique des ressources d'un cluster administré. Comme décrit dans le chapitre 3, formaliser un problème d'apprentissage par renforcement consiste à définir les trois éléments fondamentaux du processus de décision associé. Une attention particulière sera portée aux fonctions d'objectif choisies. En effet, pour le problème de l'ordonnancement, une fonction d'objectif n'est ni directe ni triviale.

5.2 Ordonnancement des tâches

Le problème que nous adressons ici est l'ordonnancement des demandes de traitement arrivant sur un site de grille de calcul (Computing Element, dans la terminologie EGEE). Les CEs sont généralement composés d'un ensemble de machines uniformes ou quasi uniformes en termes de puissance de traitement et de mémoire. Une fois les jobs acheminés dans la queue du site par le Ressource Broker (voir chapitre 4), l'ordonnanceur local du site est le composant logiciel chargé de déterminer quand et sur quelle machine les exécuter. On admet dans notre travail qu'une machine ne traite qu'un job à la fois sans possibilité de préemption. Cette hypothèse est réaliste, elle s'applique fréquemment dans le cadre des sites de l'infrastructure EGEE.

L'ordonnancement au sein des sites peut donc être considéré comme un agent devant prendre des décisions, que sont les placements des jobs présents en file d'attente, dans un environnement qui est le site de grille considéré. L'agent d'ordonnancement reçoit un gain, après chaque terminaison de job, correspondant à une valeur de performance pour l'ordonnancement effectué. Toute la question est désormais de définir :

1. les variables qui décriront l'environnement considéré
2. l'action, matérialisée par un job en attente dans une queue
3. le gain, correspondant à une fonction d'objectif à définir

On considère un site comme 1) un ensemble de queues où des jobs sont en attente de traitement et 2) un ensemble de machines toutes aptes à exécuter les jobs arrivés sur le site administré. Cette dernière hypothèse est réaliste car, comme décrit dans le chapitre 4, le ressource broker est en charge d'affecter les demandes d'exécution de job aux sites capables de les effectuer. De là, nous définissons le processus de décision de Markov suivant

- Soit S, l'ensemble des états du système, composé des queues et du cluster de calcul
- Soit A, l'ensemble des jobs en attente
- Soit $T : S \times A \rightarrow S'$, une fonction de transition d'un état du cluster à un autre à la terminaison d'un job ou dès l'arrivée d'un job dans la queue si le cluster n'est pas pleinement alloué
- Soit $R : S \rightarrow \mathbb{R}$, une fonction de gain, calculée à la fin de chaque exécution de jobs. Cette fonction sera, dans le cadre de notre travail, une combinaison linéaire de plusieurs fonctions élémentaires

Dans la suite de ce chapitre, nous revenons successivement sur les différents éléments de ce processus de décision de Markov.

5.2.1 Définition de l'environnement

Dans le contexte de l'apprentissage de politique de contrôle pour l'ordonnancement des tâches dans les grilles de calcul, la notion d'environnement se matérialise par toutes variables pouvant décrire un état dans lequel se trouve le système considéré, tels que le cluster du site administré et la file d'attente. Vengerov proposa un ensemble de variables permettant de caractériser un système de calcul dans le cadre d'un apprentissage de politique d'ordonnancement pour une machine unique [Ven07b]. Nous étendons cette approche au cas des grilles de calcul. Ainsi, le système étudié comporte :

- le cluster, l'ensemble des machines faisant office d'unité d'exécution des jobs
- les files d'attentes, regroupant les jobs près à être exécutés.

Un ensemble de variables réelles est déduit de ces deux éléments. Celles-ci seront définit comme la description d'un état de l'environnement. On nommera t_q la date d'arrivé en queue d'un job, t_b sa date d'ordonnancement marquant ainsi le début de son exécution et t_e la date de fin d'exécution de celui-ci. On notera t la date courante.

- La durée, d_f restante avant qu'un job se termine sur le cluster, $d_f = \min(t_e - t)$.
- Le ratio de machines inactives, r_i^t. Soit n, le nombre total de machine constituant le cluster et n_i^t le nombre de machine inactive à instant t : $r_i^t = \frac{n_i^t}{n}$.
- La charge de travail présente dans les queues du site c^t, à un instant t. Soit e_i^t, la durée d'exécution d'un job i présent en queue à une date t, avec $i \in 1, ..., q$. Ainsi, $c^t = \sum_{i=1}^{q} e_i^t$.
- Soit V^t, un vecteur du ratio du nombre de jobs présents en file d'attente pour chacune des d organisations virtuelles considérées, à une date t. Ainsi, $V^t = v_1^t, \ldots, v_d^t$ avec v_i^t le ratio du nombre de job en file d'attente à un instant t appartenant à la VO i.

Dans la suite de ce travail, on admettra que le processus de décision proposé considère l'existence d'une queue unique, regroupant l'ensemble des queues du site de grille administré. Par ailleurs, plusieurs des variables énoncées requièrent une connaissance à tout instant de la durée d'exécution de chaque job. Nous discuterons dans le chapitre suivant de la manière dont cette difficulté sera appréhendée concrêtement. Dans cette description théorique, nous admettons que cette valeur est connue.

5.2.2 Définition de l'action

Le choix du prochain un job à placer sur une machine libre du cluster de calcul administré constitue l'action de notre processus de décision de Markov. Formellement, chaque action est un vecteur réel de descripteurs du job considéré et de la machine du cluster chargée de son exécution. Dans un contexte de grille de calcul, on admettra que les machines du cluster sont identiques. Ainsi, l'utilisation de descripteurs du job est suffisante.

Les variables suivantes sont utilisées pour définir l'action :

- V^j, un vecteur de l'organisation virtuelle à laquelle correspond l'utilisateur ayant soumis la tâche, matérialisée par un nombre. On considère n organisations virtuelles, $V^j = \{v_1^j, \ldots, v_n^j\}$ avec $v_i^j = 1$ si le job j appartient à l'organisation virtuelle i et $v_i^j = 0$ sinon. Ce choix de représentation creuse des variables modales est courant [MES08, MBPS09, Pey09].
- La durée d'exécution de la tâche, t_e^j, pour un job j. On admet que cette valeur est définie à la génération des traces d'activité.
- Une variable binaire b, indiquant si la durée d'exécution est inférieure ou non à un seuil, τ, exprimé en seconde, permettant de distinguer les jobs batchs, à durée d'exécution long, et les jobs interatifs, à durée d'exécution court. L'appartenance d'un job j est donc établie en fonction de τ et de la valeur de t_e^j.

La question de la connaissance du temps d'exécution des jobs est encore une fois posée, nous traiterons en détail cette question dans le chapitre 6. On remarque également que l'organisation virtuelle est proposée comme descripteur de job et non l'utilisateur propriétaire de chaque job. Dans la mesure où nous déciderons de modéliser la fonction d'espérance de gain comme un modèle de réseau de neurones, il serait difficile d'employer comme variable descriptive l'utilisateur à cause de son type modale et de sa grande dimension. En effet, on compte un total de 15000 utilisateurs fréquents sur l'infrastructure de grille. Dans le cas des sites généralistes, un nombre important des utilisateurs de la grille peut avoir l'occasion d'utiliser un site proposant un large éventail de suites logicielles couramment utilisées. Un vecteur creux de cette taille n'est pas envisageable dans ce contexte. Par ailleurs, l'identité des utilisateurs peut correspondre à un utilisateur unique dans certains cas ou à un

groupement d'utilisateur, suivant notamment l'organisation virtuelle d'appartenance. Une solution à ce problème pourrait être de regrouper préalablement les utilisateurs suivant des caractéristiques de comportement. Cette solution ne sera toutefois envisageable que sous l'hypothèse d'une certaine pérennité des comportements de chaque utilisateur.

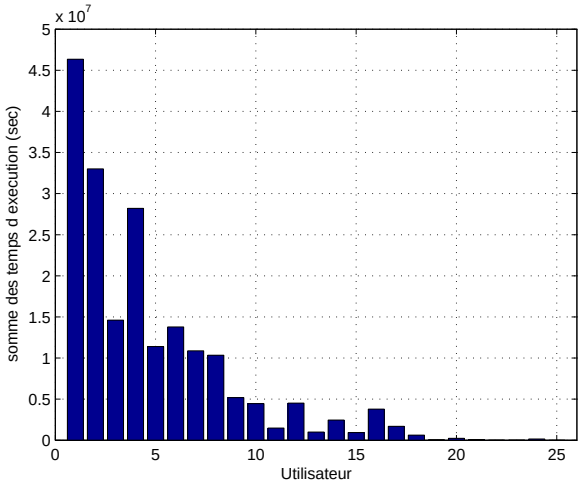

FIG. 5.1 – Sommes des temps d'exécution des jobs triées par utilisateur sur un échantillon de 10000 jobs extrait des traces d'activité d'un site de grille EGEE

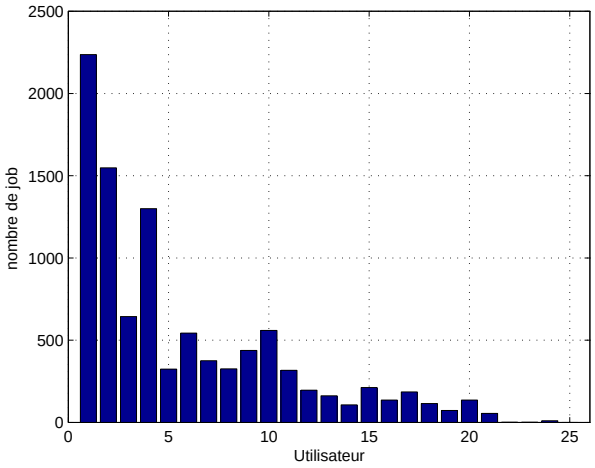

FIG. 5.2 – Nombres de jobs triés par utilisateur sur un échantillon de 10000 jobs extrait des traces d'activité d'un site de grille EGEE

Les figures 5.1 et 5.2 présentent respectivement les sommes des temps d'exécution et le nombre de job d'un échantillon de 10000 jobs extrait des traces d'activité d'un site de grille d'EGEE. On remarque que 5 utilisateurs sur les 25 considérés dans l'échantillon comptabilisent à eux seuls près de 50% de l'activité totale du site sur la période d'enregistrement.

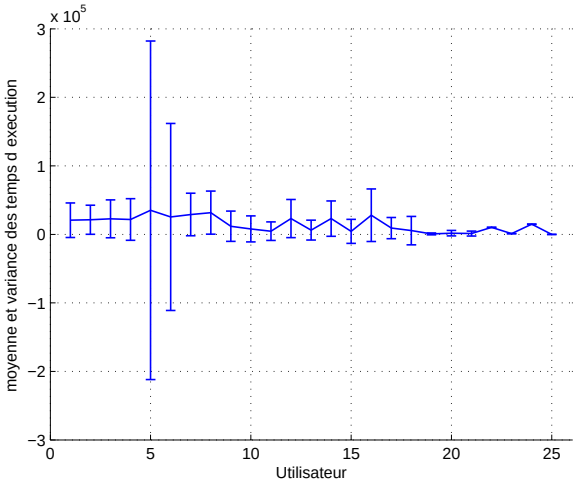

FIG. 5.3 – Moyennes et variances des durées d'exécution des jobs triées par utilisateur sur un échantillon de 10000 jobs extrait des traces d'activité d'un site de grille EGEE

La figure 5.3 présente les durées d'exécution moyenne entourées par la variance de la durée d'exécution des jobs de chacun des utilisateurs. On remarque que ces durées moyennes restent relativement comparables mais connaissent pour les premiers utilisateurs de la liste une variance près de 3 fois supérieure aux utilisateurs de fin de liste. Cette hétérogénéité de la définition d'utilisateur complexifierai l'apprentissage d'une politique de contrôle d'ordonnancement sans garantie d'offrir un gain significatif si elle était ajoutée à la définition des jobs.

5.2.3 Fonctions de gain

L'ensemble des fonctions d'objectif standards dans le domaine de l'ordonnancement des tâches est considérable [T'K06, Leu04]. Certaines sont spécifiques au domaine des grilles. Dans cette section, l'emploi de la réactivité pour notre problème d'ordonnancement est décrit et justifié.

Réactivité

La réactivité associée à un job est le rapport entre sa durée d'exécution et le temps total passé dans le système. Cette mesure permet de pénaliser les délais d'attente en fonction des temps d'exécution de chacun des jobs et offre donc une fonction de coût réaliste si l'activité du site est en partie composée de jobs à courte durée d'exécution. Intuitivement, l'objectif est de pénaliser de manière équivalente une attente de plusieurs minutes subit par un job comptant moins d'un quart d'heure d'exécution et l'attente d'une heure d'un job

requièrant une journée complète d'exécution. De cette manière, nous souhaitons remédier à la difficulté actuelle des grilles de calcul de prendre en charge efficacement les jobs interactifs, à faible durée d'exécution.

Soit T_e, la durée d'exécution d'un job, soit T_w la délai d'un job avant le début de son exécution, on définit la réactivité W de ce job comme

$$W = \frac{T_\mathrm{e}}{T_\mathrm{e} + T_\mathrm{w}} \tag{5.1}$$

Concernant l'efficacité d'un ordonnancement les indicateurs les plus courants sont : le temps d'attente, le makespan, la taille moyenne de la file d'attente ainsi que le nombre moyen d'utilisateurs présents dans le système au cours du laps de temps donné [T'K06]. Cependant, ces indicateurs sont inappropriés dans le cadre de notre problème. En effet, le makespan, très utilisé dans les travaux de planification [JSOS03, JSOS99, SW98] est inadéquat dans le cadre d'un ordonnancement de grille de calcul où le système décisionnel tend à être utilisé sans interruption. En outre, la différence essentielle réside dans le fait que la planification prend pour acquis un ensemble de connaissances absentes dans le cas des problèmes d'ordonnancement, notamment :

- la fonction de transition
- l'ensemble des actions en tout état de l'environnement disponibles au départ de l'expérience
- la fonction de gain

Concernant le nombre d'utilisateurs présents dans le système, ni la minimisation, ni la maximisation de cette valeur n'est pertinent pour notre problème car l'objet est d'adapter la politique d'ordonnancement d'un système de grille aux demandes d'utilisateurs en fonction de leur durée d'exécution présumée. Enfin, l'utilisation du temps d'attente moyen ne permet pas la mise en place d'une qualité de service différenciée.

Équité

Les grilles de calcul sont généralement construites par la mise en commun de moyens financiers issus de différentes institutions. Dans ce contexte, chaque participant demande à obtenir une part équitable de l'utilisation du cluster en fonction de son investissement dans la grille. Cette problèmatique est un second enjeux auquel doit faire face un ordonnanceur de site de grille. L'équité représente la différence entre l'allocation actuelle des ressources et la définition préalable du partage entre chaque organisation virtuelle. Le processus d'allocation doit être tel que le service reçu par chaque organisation virtuelle

est proportionnel à ce partage prédéfini. S'il y a m organisations virtuelles autorisées à utiliser un site de calcul donné, le partage est généralement exprimé comme un vecteur de dimension m du pourcentage des ressources totales du cluster, $z = (z_1, \ldots, z_m)$. Soit S_{kj}, la fraction du service total reçue par la VO k à la date d'ordonnancement du job j, la distance entre l'allocation optimale et l'allocation actuelle est une bonne mesure du défaut d'équité. Cette distance est défini par

$$D_j = \max_k (w_k - S_{kj})_+ \tag{5.2}$$

avec $x_+ = x$ si $x > 0$ et 0 sinon.

Cette mesure est bornée par $M = \max_k(z_k)$. On en déduit une mesure gain normalisée par transformation linéaire. Soit F_j, le gain d'équité associé à l'ordonnancement d'un job j

$$F_j = -\frac{D_j}{M} + 1 \tag{5.3}$$

5.2.4 Principe de rétention en ordonnancement

Dans cette sous section, nous discutons de la nature des décisions effectuées par le processus d'ordonnancement afin d'apporter une interprétation générale aux résultats que nous obtiendrons dans les expérimentations du chapitre 7.

Dans le cas d'une politique de type FIFO, les jobs sont ordonnancés, par définition, suivant leur ordre d'arrivée. Dans le cas d'un ordonnancement impliquant une quelconque autre décision, déterministe ou stochastique, et dans la mesure où aucun mécanisme de préemption n'est assuré dans le système administré, une décision d'ordonnancement consiste à suivre ou à modifier cet ordre d'ordonnancement FIFO. Sachant que le système n'a aucun contrôle sur la date et l'ordre d'arrivée des jobs, une modification de cet ordre d'ordonnancement FIFO revient uniquement à retenir ou non un job en queue au profit d'un autre job arrivé après ce premier job car son espérance de gain est estimée comme plus élevée, dans le cas d'un système par apprentissage par renforcement. Une décision de l'ordonnanceur correspond donc à un choix de rétention d'un job au profit d'un autre dans l'optique de maximiser une fonction de gain définie.

Dans le cas de la réactivité comme fonction de gain, le comportement attendu d'un ordonnanceur par apprentissage par renforcement est de privilégier les jobs les plus courts afin d'assurer un temps d'attente minimal à ces derniers et donc un gain de réactivité

maximale. En fonction de la distribution des durées d'exécution des ensembles de jobs considérés, on s'attend donc à une augmentation de la variance du gain en réactivité pour les jobs les plus longs car ces derniers devront avoir tendance à être retenu en queue au profit des jobs plus courts.

5.3 Dimensionnement dynamique des ressources

Nous traitons, dans cette partie, de la question de l'adaption dynamique du nombre de machine du cluster de calcul administré en fonction du flux de demande. La figure 5.4 présente une mesure d'utilisation effectuée sur un site de la grille EGEE sur la période du 24 Juillet au 12 Septembre 2006. La série temporelle est calculée selon la mesure d'utilisation, définie dans l'intervalle $[0, 1]$, définie ci-après à la date de fin des 10000 jobs constituant cette trace d'activité. On remarque une sous utilisation de la grille qui s'explique par une volonté de l'administration du site d'être en mesure d'appréhender les périodes éventuelles de forte demande comme le montre le pic à 0.6 présent au début de la série. Dans cette section, nous considérons le problème du dimensionnement des ressources dans un site de grille de calcul comme un apprentissage par renforcement indépendant du problème de l'ordonnancement des tâches présenté précédemment. Le dimensionnement dynamique se définit comme la modification en cours d'utilisation du nombre d'unités de traitement, de machines, constituant le cluster du site de grille administré. L'objectif sera donc d'améliorer l'utilisation globale d'un site de grille administrée.

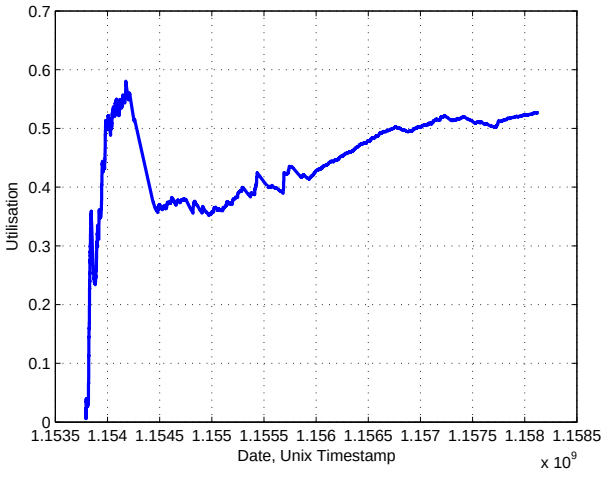

FIG. 5.4 – Série temporelle de l'utilisation pour l'ordonnancement d'EGEE

Nous définissons un processus de décision de Markov dédié à cette question. Néanmoins, le MDP proposé n'est pas en tout point différent de celui définit précédemment pour le problème de l'ordonnancement. En effet, nous conservons les mêmes variables pour définir

l'environnement car elles sont des valeurs disponibles et cohérentes pour le problème de l'ordonnancement et du dimensionnement.

5.3.1 Définition de l'action

Le contrôleur de dimensionnement est en charge de choisir, pour une période de temps correspondant à un nombre de job ordonnancé, le nombre de machine, a_e, à placer en état de disponibilité pour les jobs arrivant en file d'attente du cluster. L'action est donc une variable dans $\{N, \ldots, M\}$, avec N et M les nombres respectivement minimum et maximum de machine pouvant être mises à disposition des utilisateurs. Une difficulté est à noter, l'action se limite également au nombre de machines en exécution au moment de la prise de décision de dimensionnement. En effet, le nombre de machine ne peut être diminuer en deçà du nombre actuellement en phase d'exécution d'un job. En effet, cela obligerait la mise en place d'un système de préemption des jobs, que nous supposons ne pas prendre en charge dans ce modèle.

5.3.2 Fonction de gain

Utilisation

Généralement, l'utilisation est le ratio entre une quantité de ressource disponible et la quantité de ressource effectivement utilisée dans un intervalle de temps donné. Soit $(t_1, \ldots, t_N)$, des instants successifs de prise de décision du contôleur de dimensionnement, avec $t_1 = 0$. Soit P_k, le nombre de processeurs alloués dans l'intervalle $[t_k, t_{k+1}]$. Soit f_n, la somme des temps d'exécution des jobs ayant été traités dans l'intervalle $[t_1, t_n]$, le gain d'utilisation, U_n, au temps t_n, est défini par

$$U_n = \frac{f_n}{\sum_{k=0}^{n} P_k(t_{k+1} - t_k)} \tag{5.4}$$

5.4 Pondération des fonctions d'objectif

Pour les fonctions d'objectifs considérées dans les deux problèmes de contrôles décrit précédemment, toutes les fonctions de gain sont bornées dans $[0, 1]$. Dans les deux problèmes, la fonction de gain effectivement utilisée est une combinaison linéaire de deux des trois fonctions de gain présentées.

Dans le cas de l'ordonnancement des jobs, la fonction de gain effective, R_s, est défini par

$$R_s = \lambda_s W + (1 - \lambda_s) F \tag{5.5}$$

Et dans le cas du dimensionnement du site, la fonction de gain utilisée, R_e, est défini par

$$R_e = \lambda_e W + (1 - \lambda_e) U \tag{5.6}$$

avec $\lambda_s, \lambda_e \in [0, 1]$, les coefficients permettant de contrôler le compromis entre les différentes fonctions de gain. Ce choix de conception nous permet, par ailleurs, de qualifier les problèmes de contrôle considérés comme multi objectifs.

5.5 Organisation des processus de décision dans l'infrastructure administrée

Nous passon en revue les points communs et les divergences des deux processus de décision abordés dans ce travail. Le processus de décision chargé de l'ordonnancement des tâches modifie l'état de la queue et du cluster alors que le processus de dimensionnement agit uniquement sur du cluster, en modifiant le nombre de machines actives. Les deux processus de décision sont strictement indépendants, il utilisent toutefois des informations communes de l'état du système considéré.

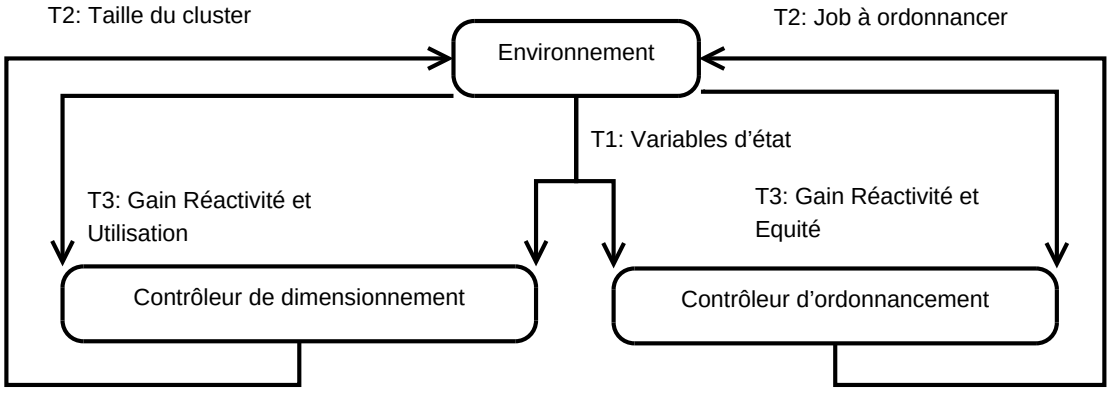

FIG. 5.5 – Organisation des processus de décision dans l'infrastructure de site.

La fréquence de décision des deux processus est différente. Comme résumé dans la figure 5.5, les jobs sont ordonnancés dès qu'une machine du cluster est libérée et qu'un

job est en attente en queue. Le processus de dimension n'agit, quant à lui, que lorsqu'un nombre m, fixé, de jobs a été exécuté depuis la dernière décision de dimensionnement.

Les variables d'ajustement des fonctions d'objectifs des processus d'ordonnancement et de dimensionnement, respectivement λ_s et λ_e ont été défini dans une section précédente et peuvent, bien évidemment, avoir des valeurs différentes, suivant les volontés de l'administration système du site considéré. On précisera les valeurs jugées pertinente lors des expériences du chapitre 7.

Enfin, les fréquences de ré-apprentissage des fonctions d'espérance de gain des deux modèles sont différentes. Ces valeurs correspondent au nombre de décisions devant être prise par chacun des deux MDPs avant de relancer une procédure de ré-estimation de la fonction de gain d'espérance de gain correspondante. En effet, les fréquences des actions effectués par les processus de décision étant elles même différentes, celles ci le sont également. On précisera les valeurs jugées dans le chapitre 6.

Résumé

Dans ce quatrième chapitre, nous avons formalisé les problèmes de l'ordonnancement des tâches dans un site de grille ainsi que celui du dimensionnement dynamique du cluster administré comme deux problèmes d'apprentissage par renforcement en domaine continu. Nous avons proposé les éléments des processus de décision associés à chacun de ces problèmes. L'accent a été mis sur la justification des fonctions de gain proposées que sont, la réactivité, l'équité et l'utilisation. Ensuite, nous avons proposé l'emploi de deux modèles de régression afin d'apprendre les fonctions d'espérance de gain associées à la recherche d'une politique de contrôle optimale pour ces deux problèmes, le réseau de neurone multi couche et l'Echo State Network. Nous avons enfin orienter notre présentation sur l'utilité des modèles à mémoire et en particulier de l'ESN lorsque la propriété markovienne du processus de décision considéré est remise en cause. Ce phénomène s'explique par le manque de variable décrivant l'état courant de l'environnement, ne permettant donc plus de résumer le passé des actions et états visité par l'agent décisionnel, comme expliqué dans le chapitre 3. Dans le chapitre suivant, nous présentons la plateforme de simulation d'ordonnancement de site de grille développée au cours de cette thèse ainsi que des détails de l'implémentation des algorithmes d'apprentissage utilisés.

Chapitre 6

Mise en oeuvre et environnement d'expériences

Ce chapitre présente le protocole d'expérimentation des modèles exposés dans le chapitre précedent ainsi que les développements logiciels associés. La première partie de ce chapitre détaille l'organisation et le fonctionnement du simulateur de site de grille de calcul développé pendant cette thèse et utilisé au cours des expériences présentées dans le chapitre 7. La notion d'interactivité ainsi que les post-traitements effectués seront explicités. Dans une seconde partie, l'implémentation des algorithmes d'apprentissage par renforcement pour l'allocation de ressources et le dimensionnement sera détaillée. Ces algorithmes ont été dissociés de la partie simulation afin de les rendre exportables simplement vers des contextes différents, éventuellement réels. La troisième partie de ce chapitre décrit les traces synthétiques utilisées au cours des expérimentations. Plusieurs listes de temps d'arrivée et de durées d'exécution ont été générées suivant différentes distributions statistiques. Celles-ci ont permis d'obtenir des flux de demandes aux propriétés représentatives des comportements d'un flux de demandes observés sur la grille EGEE. La dernière partie décrit et analyse une trace d'activité réelle extraite de l'applicatif de supervision de site de la grille EGEE. La méthode utilisée pour l'extraction des informations à partir des journaux (*logs*) de l'applicatif sera présentée.

6.1 Description du simulateur

6.1.1 Présentation générale

Les expériences ont été effectuées via un simulateur de système de batch développé au cours de cette thèse. La motivation principale de ce développement a été de se munir d'un simulateur d'évènements discrets capable de reproduire la dynamique d'un site de grille de calcul. Concrètement, un système de batch se compose d'une file d'attente et d'un ensemble de machines identiques que l'on nommera *cluster* dans le reste du chapitre. Cette plateforme a été conçut de manière générique afin d'être en mesure de modéliser, notamment, les sites composant la grille d'EGEE. Toutefois, le dispositif couvre également tout type de système batch à file d'attente unique.

Un format d'entrée correspondant à la modélisation du chapitre précédent a été défini. Ainsi, les simulations produites peuvent se baser sur des fichiers d'entrée représentés dans un format générique de flot d'activité. Ces fichiers d'entrée sont :

- soit extrait de traces réelles produites par un applicatif de supervision utilisé par plusieurs sites de la grille EGEE (traces *réelles*) ;
- soit générés à partir de distributions statistiques choisies (traces *synthétiques*).

Dans ce contexte, la qualité d'un ordonnancement produit par un algorithme implémenté au sein de la plateforme de simulation a pu être estimée. En effet, en rejouant dans ce

simulateur une trace d'activité synthétique ou réelle, c'est à dire en respectant les ordres d'arrivée en queue et d'ordonnancement d'une trace donnée, il est possible d'estimer les valeurs de gain obtenues par l'algorithme d'ordonnancement à tout moment. Les gains mentionnés dans ce chapitre correspondent à ceux définis dans la section 5.2.3 et 5.3.2.

Le fonctionnement général de la plateforme de simulation se décompose en trois parties :

1. Simulation du mécanisme de gestion des jobs (gestion de la file d'attente, affectation des jobs aux machines)
2. Algorithmes d'ordonnancement et de dimensionnement
3. Procédure d'estimation des fonctions d'espérance de gain pour les algorithmes d'ordonnancement et de dimensionnement basés sur l'apprentissage par renforcement

La figure 6.1 présente les différents modules logiciels développés et leurs inter-dépendances. Autour du module assurant la simulation de la dynamique du site, ont été implémentés un module chargé de l'ordonnancement des jobs suivant une politique donnée, éventuellement basé sur un mécanisme d'apprentissage par renforcement. De même, un module chargé du dimensionnement du cluster de calcul du site ainsi qu'un module dédié à l'implémentation des algorithmes d'apprentissage des fonctions Q, respectivement pour l'ordonnancement et le dimensionnement, ont été developpés.

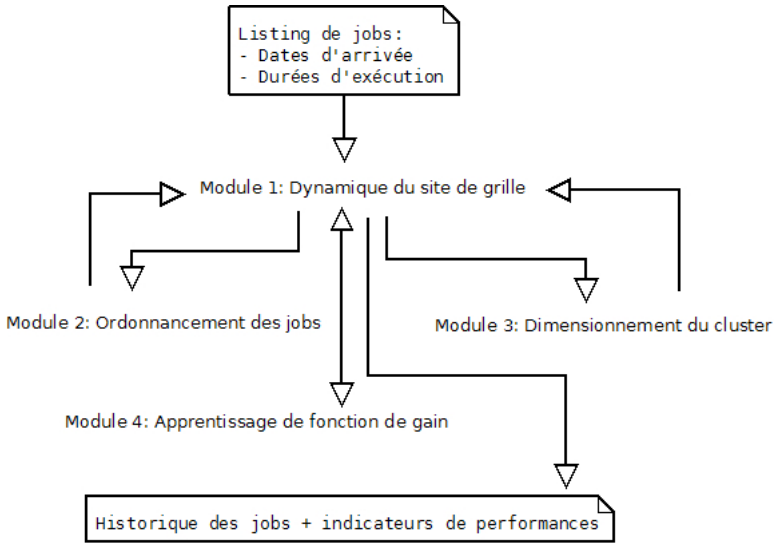

FIG. 6.1 – Diagramme fonctionnel de la plateforme de simulation

Deux ensembles d'éléments logiciels sont à distinguer. Tout d'abord, les modules logiciels propres à la simulation de la dynamique de l'environnement, constitué de la file d'attention et du cluster, ainsi que le module chargé de la collecte, et de la production de statistiques, des résultats expérimentaux. Le simulateur incorpore également les modules

logiciels implémentant les algorithmes d'apprentissage par renforcement proposés dans le chapitre 5.

6.1.2 Le format des entrées

Le simulateur prend en entrée une liste de descripteurs de jobs représentant un flot de requêtes. Chaque descripteur est un triplet représentant :

- La date d'arrivée en file d'attente
- La durée d'exécution
- L'organisation virtuelle d'affiliation

Le format ne comprend pas de typage d'interactif ou batch, ceci est expliqué au paragraphe suivant. Les durées sont exprimées en secondes ; les organisations sont définies par une valeur modale sur $\mathbb{N}$.

Cette définition des entrées introduit implicitement un modèle où le comportement de l'utilisateur en termes de soumission n'est pas conditionné par le comportement de la grille en termes de temps de réponse. Il s'agit d'une hypothèse simplificatrice : de façon générale, les performances de l'ordonnanceur peuvent avoir un impact sur le comportement de soumission de l'utilisateur, en particulier pour les jobs réactifs et pour les travaux de mise au point. Toutefois, une analyse comportementale de l'utilisateur sort du cadre de ce travail.

6.1.3 Définition de l'interactivité

Pour toute simulation, les jobs doivent être étiquetés comme interactif ou batch, afin de mettre en place des situations où des demandes de qualité de service différenciée seraient présentes. Bien évidement, cette information doit être générée dans les traces sythétiques. Elle n'est pas non plus présente dans les traces réelles : l'attribut INTERACTIVE dans les requêtes gLite correspond à une fonctionnalité complètement différente, qui consiste à déporter l'entrée/sortie standard du job vers la machine de soumission. Dans ce travail, la définition de job interactif correspond aux tâches dont la durée d'exécution est inférieure à une valeur donnée. Dans toutes les expériences, cette durée sera toujours de 15 minutes, 900 secondes.

La VO de chaque job aurait pu constituer un indice sur cette catégorisation. Dans les traces utilisées, chaque organisation virtuelle présente une distribution large de temps d'exécution, bien que spécifique. Par exemple, l'organisation virtuelle *Atlas*, une expérience de physique, est dominée par des jobs longs, mais les jobs interactifs représentent une

fraction significative de l'échantillon, alors que l'organisation virtuelle *Biomed* lance majoritairement des jobs de courte durée d'exécution. Ce phénomène se vérifie sur les données complètes d'EGEE [PGRKL10].

6.1.4 Dynamique du site de grille

Ce bloc incorpore les procédures, au sens algorithmique, effectuant l'alimentation et le vidage de la queue en fonction des dates d'arrivée des jobs. Cet élément est également en charge de la mise à jour de l'état du site de grille de calcul en fonction des décisions de placement de l'ordonnanceur et de la durée d'exécution des jobs en activité sur le cluster.

Ce composant central du simulateur est déterministe et doit être décorrélé des procédures décisionnelles liées à l'ordonnancement et au dimensionnement du cluster. Le pseudo-code 6 décrit son fonctionnement. Un pas de l'algorithme de simulation de site correspond à l'arrivée d'un nouveau job dans la queue ou à la terminaison d'un job en exécution dans le cluster. La simulation prend fin lorsque tous les jobs présents dans le listing de départ ont été exécutés et achevés. Dans cet algorithme, il est admis, comme pour la grille EGEE, qu'un unique job est exécuté à un instant donné sur une machine du cluster. La boucle de la ligne 4 à 18 de l'algorithme sera définie comme un pas de simulation.

L'algorithme général de simulation (6) prend en entrée la liste descriptive des jobs à simuler. Après l'initialisation du cluster et de la queue, qui correspond à la création des structures en mémoire, la double boucle principale de l'algorithme démarre. Tant que la liste complète des jobs n'a pas été traitée, les jobs présents en queue sont placés dans le cluster de calcul et exécutés dans l'ordre défini par la politique d'ordonnancement (*Ordonnance_Job*), pouvant être déterministe (FIFO) ou commandée par un mécanisme d'apprentissage par renforcement. Une fois que la totalité des machines libres est en activité ou que la queue est vidée, l'algorithme de dimensionnement (*Dimensionnement*) du cluster est invoqué lorsqu'un nombre m de jobs, prédéfini (c.f. section 5.3). La date du prochain pas de simulation est ensuite calculée (*Calcul_Nouvelle_Etape*). Cette étape, correspond à la date de la prochaine fin d'exécution ou de la prochaine arrivée d'un job en queue. Une fois cette nouvelle étape calculée, les jobs terminés, c'est dire ceux dont la date de fin d'exécution est égale à la date courante de simulation, sont supprimés. Les gains (*rewards*) associés à chacun de ces jobs sont calculés (voir sections 5.2.3 et 5.3.2) et stockés en mémoire accompagnés des valeurs des variables d'état au moment de la prise de décision. Enfin, les nouveaux jobs arrivés, c'est à dire, les jobs dont la date d'arrivée correspond au temps courant de la simulation, sont placés dans la queue. La *liste_param* représente les paramètres de configurations qui seront précisés dans les algorithmes 7 et 8.

Algorithm 6 Simulation d'un site de grille

Require: liste_param

```
cluster ← Initialisation_Cluster()
queue ← Initialisation_Queue(list_job)
nb_jobs_traités ← 0
temps_courant ← 0
{mécanisme de simulation}
while queue non vide do
    while Machine_Libre(cluster) > 0 do
        lst_jobs_choisis ← Ordonnance(liste_param, temps_courant)
        Supprimer_Job(queue, list_jobs_choisis)
    end while
    if (nb_job_traités, n) % 0 then
        Dimensionne(liste_param)
    end if
    temps_courant ← Calcul_Nouvelle_Etape(cluster)
    Supprimer les jobs terminés du cluster
    Incrémentation (nb_job_fin)
    {Comptabilité des gains}
    Calculer le gain de réactivité des jobs terminés
    Calculer le gain d'équité des jobs terminés
    Calculer le gain d'utilité
    Stocker les gains
    Placer les jobs arrivant dans la queue
end while
```

6.1.5 Post-traitements

Pendant une simulation, une trace détaillée de l'activité du superviseur est enregistrée. Plus précisément, pour chaque job, sont sauvegardés :

- Les éléments du descripteur
  - Date de mise en queue
  - Durée d'exécution
  - Organisation virtuelle
  - Type de job (interactif ou batch)
- Les caractéristiques de l'ordonnancement
  - Date de début d'exécution
  - Date de fin d'exécution
  - Numéro de machine sur laquelle le job a été exécuté
- Les indicateurs utilisés pour la prise de décision
  - Gain en réactivité
  - Gain en utilisation
  - Gain d'équité

Les éléments du descripteur sont extraits du fichier d'entrée de la simulation, les autres informations sont produites par le simulateur. Ces attributs sont utilisés pour analyser la qualité de l'ordonnancement effectué. Ils permettent aussi de valider l'exactitude du simulateur, en vérifiant la consistance des résultats produits. Par exemple, la durée d'exécution est extraite du fichier d'entrée de la simulation et comparée avec les dates de début et de fin d'exécution extraites de la simulation. De même, les gains en réactivité, équité, utilisation ainsi que les valeurs de la réactivité sont calculés pendant la simulation et peuvent ensuite être recalculés pour vérification. On vérifie ainsi le bon fonctionnement du simulateur. Le numéro de machine ayant exécuté chaque job est utilisé pour établir des journaux d'exécution sur chacune d'entre elle et ainsi observer l'activité du cluster.

Une suite logicielle a été développée pour faciliter la mesure et le tracé d'indicateur de performance. Les indicateurs suivant ont été extraits :

- Durée d'attente
- Réactivité
- Équité

Les mesures statistiques classiques ont été utilisées, le choix de la médiane permettra d'analyser des résultats où la valeur de variance serait trop importante pour déduire des conclusions des moyennes observées.

- Moyenne empirique

– Variance
– Médiane

Les représentations suivantes donnent une vision plus précise du comportement adopté par le superviseur au cours d'une simulation et permettrons d'apporter des interprétations aux décisions prises par les différents ordonnanceurs étudiés.

– Distributions cumulatives
– Distributions cumulatives inverses
– Séries temporelles

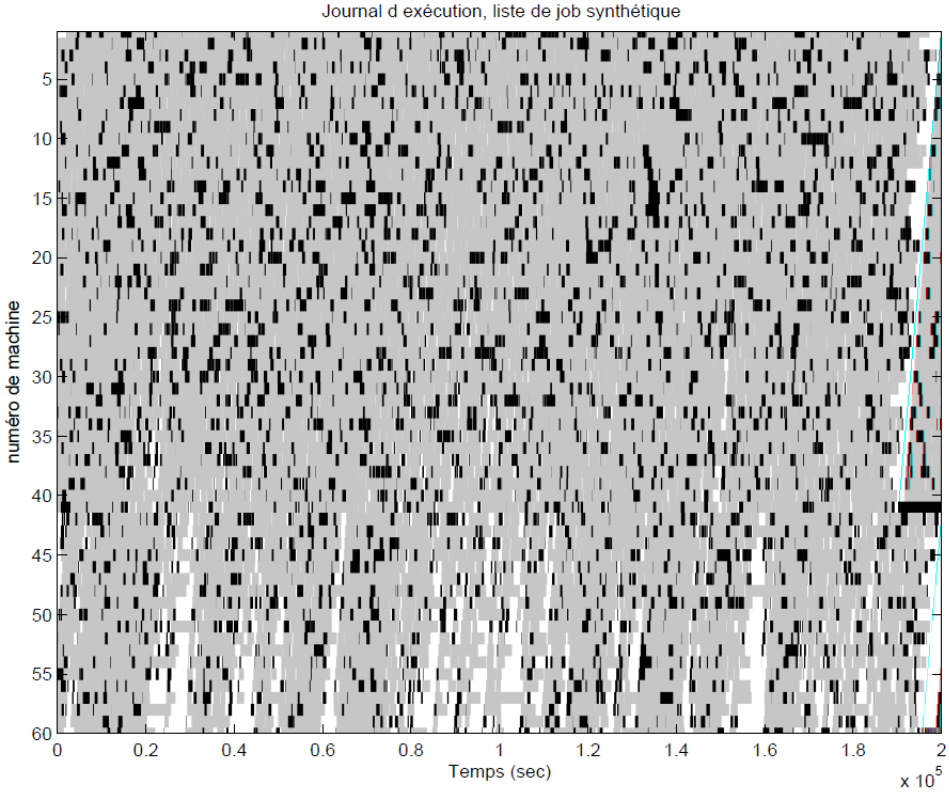

FIG. 6.2 – Exemple de journal d'une simulation d'ordonnancement

La construction de journaux a été permis grâce au champs *machine* présent dans les logs du simulateur. Il est ainsi possible de visualiser l'activité des différentes machines au cours d'une simulation ainsi que le placement des jobs interactifs et batchs sur ces machines. La figure 6.2 est un journal d'exécution d'une expérience utilisant un flot de demande synthétique de type Poisson exponentiel et dont le contrôleur d'ordonnancement est un réseau de neurones multi-couche. Pour chaque machine représentée en ordonnée, on note, à chaque seconde de la simulation, représentée en abscisse, son état. En noir, la machine exécute un job interactif, en gris un job long et en blanc que la machine est inactive.

6.2 Implémentation des algorithmes d'apprentissage par renforcement

6.2.1 Ordonnancement des jobs

L'ordonnanceur a pour rôle d'affecter à une machine inactive, un job présent en queue. Dans le cas d'un ordonnancement par apprentissage par renforcement, ce choix est basé sur les paramètres appris du modèle d'estimation de la fonction d'espérance de gain (Q), tout au long d'une simulation (voir section 3.3).

L'algorithme de décision utilise, en entrée, les caractéristiques de l'état du site ainsi que le descripteur de chaque job candidat à l'ordonnancement. Ce choix du job à ordonnancer est une procédure de type ϵ-greedy. Le gain espéré pour l'ordonnancement d'un job est estimé par le modèle statistique chargé de modéliser la fonction d'espérance de gain Q, c'est à dire un réseau de neurones ou un Echo State Network, comme présenté dans la section 3.4. Une fois l'estimation effectuée pour l'ensemble des jobs présents dans la queue, le job présentant la meilleur espérance de gain est choisi avec une probabilité $(1 - \epsilon)$, sinon un job candidat est choisi au hasard dans la queue. Comme expliqué dans la section 3.2.2, avec une valeur pour ϵ proche de 1, cette méthode permet de conserver un comportement d'exploration nécessaire à l'apprentissage d'une politique de contrôle optimale. Le paramètre ϵ est extrait de *liste_param*.

Algorithm 7 *Ordonnance* (invoqué dans l'algorithme 6)

Require: liste_param

```
1: modèle ← Apprendre(table_Q)
2: while nb_machine_libre(cluster) > 0 do
3:    state_var ← Calcul_Variable_Etat(cluster, queue, tps_courant)
4:    liste_Q_estim ← [ ]
5:    for i=1 to taille(queue) do
6:       Q_est ← Estimer_Q(state_var, job, modèle)
7:       liste_Q_estim ← Ajouter(liste_Q_estim, job, Q_est)
8:    end for
9:    meilleur_job ← Sélectionne_Job(liste_Q_estim)
10:   Choisir_Job_ϵ-Greedy (ϵ, meilleur_job)
11:   table_Q ← Ajouter(table_Q, state_var, selected_job)
12: end while
```

Le pseudo-code 7 décrit l'algorithme général d'ordonnancement implémenté dans le simulateur. Pour chaque machine libre du cluster, à un pas de simulation donné, l'espérance de gain (*Q_est*) associée au choix d'un job présent en queue est calculée (*Estimer_Q*). Dans

l'algorithme, cette évaluation se base sur le modèle statistique, *modèle*, et sur une mesure de variables d'environnement extraites de l'état du cluster, *state_var*, de la queue et enfin du job. Les jobs ordonnancés sont sélectionnés (*Select_Job* et *Choisir_Job_ϵ-Greedy*) selon une procédure de type ϵ-gloutonne (voir section 3.2.2). L'algorithme présenté ici est générique, il ne dépend pas des variables choisies pour la description des job, de l'environnement ou encore du modèle statistique employé pour l'estimation de l'espérance de gain. Avant que le choix ne soit appliqué, les couples {états, actions} seront stockés et concaténés au gain obtenu lorsque le job ordonnancé aura fini son exécution.

6.2.2 Dimensionnement du cluster

Le système conçu est capable d'adapter le nombre de machines d'un centre de calcul administré à l'intensité de la charge à traiter. Cette question du dimensionnement dynamique des ressources a été présenté en section 5.3. L'objectif du système est double, on compte maximiser l'utilisation du site et maintenir une qualité de service suffisante pour les utilisateurs. Toutefois, ces objectifs sont divergents : à l'extrême, une utilisation optimale est garantie lorsque le cluster est réduit à une unique machine alors que la satisfaction des utilisateurs demanderait un nombre non borné de machines afin d'éviter tout temps d'attente avant le démarrage de chaque exécution. L'algorithme de dimensionnement est un apprentissage par renforcement dont le gain est une somme pondérée de l'utilisation mesurée du site et de la réactivité estimée de ce dernier (voir section 5.4).

Concrètement, la procédure d'apprentissage se décompose de la même manière que décrite dans le système d'ordonnancement. Un échantillonnage de la fonction Q est effectuée, c'est à dire, des triplets {état, action, gain} sont collectés au cours de la simulation en conservant en mémoire les variables mesurées sur le site. Dans ce MDP, l'action consiste à choisir le nombre de machines à mettre en activité, le gain correspondant à chaque action est obtenu au pas suivant de la simulation.

Rappelons que le dimensionnement du site s'effectue tous les n jobs ordonnancés et une phase de régression de la fonction de gain de ce système est effectuée tous les m jobs ordonnancés, avec $m > n$. Il serait possible de lancer la procédure de régression de la fonction de gain après chaque acquisition de gain pour un {état, action} passé, c'est à dire $m = n$. Cette solution n'a pas été adoptée pour des raisons de performances, bien que valide [TDC+08].

Le pseudo code 8 détaille la procédure de dimensionnement dynamique du cluster de calcul implémentée. L'estimation de la fonction d'espérance de gain pour le problème du dimensionnement du cluster (*modèle*) est utilisée afin de choisir le nombre de machines représentant la meilleure espérance de gain dans l'ensemble des configurations légitimes.

Algorithm 8 Dimensionne

Require: liste_param

```
state_var ← Calcul_Variable_Etat(cluster, queue, tps_courant)
if nb_machines_occupees > nbre_machine_minimal then
    nb_machine_minimal ← nb_machine_occupees
end if
list_esperance_gain ← [ ]
for i ← nb_machine_minimale :nbre_machine_maximale do
    espérance_gain ← Calcul_Espérance(state_vars, i, modèle)
    Ajout(liste_esperance_gain, i, espérance)
end for
nouvelle_taille_cluster ← Dimensionnement_Cluster_ϵ-Greedy(ϵ, liste_espérance_gain)
nouvelle_taille_cluster
```

En effet, le panel des actions n'est pas toujours disponible. Il est impossible de diminuer le nombre de machines en deça d'une valeur inférieure au nombre de jobs en exécution au moment de la prise de décision (*nb_machine_minimal*) car les jobs ne peuvent pas être suspendus. Le nombre de machine minimal et maximal représente l'écart dans lequel la taille du cluster est autorisé à varier et est extrait de *liste_param*. Après avoir calculé l'espérance de gain associée à chaque nombre de machines pouvant s'appliquer, grâce au modèle de la fonction Q associée (*Calcul_Espérance*), on choisit, par une procédure de type ϵ-*greedy* (voir section 3.2.2), la nouvelle taille de cluster à appliquer (*Dimensionnement_Cluster_*ϵ-*Greedy*).

6.2.3 Estimation des fonctions d'espérance de gain

Comme présenté dans le chapitre 5, deux politiques de contrôle distinctes sont apprises, la politique de contrôle associée au problème de l'ordonnancement et la politique associée au problème du dimensionnement du site. Dans les deux cas, un modèle paramétrique de la fonction Q associée est régressé à partir d'un échantillonnage de cette fonction. L'état et l'action du système considéré sont tous deux des vecteurs réels. La section 3.3.1 a justifié qu'un apprentissage en tableau n'est pas envisageable et qu'il s'agit donc d'une régression de cette fonction ; pour distinguer l'apprentissage de la fonction Q, de l'apprentissage général, dans suite, on l'appelera toujours estimation.

L'algorithme d'apprentissage se décompose en deux phases :

1. Echantillonnage de la fonction Q
2. Régression de la fonction Q

La suite de cette section décrit ces deux étapes de l'algorithme d'apprentissage.

Echantillonnage de la fonction Q

On rappelle que la fonction $Q : S, A \rightarrow \mathbb{R}$ modélise l'espérance de gain à long terme associée au choix d'une action A quand l'environnement considéré est dans un état S. Dans le superviseur, l'échantillonnage d'une fonction Q consiste à stocker en mémoire des triplets {état, action, gain} d'une simulation du processus de décision de Markov considéré.

Dans le cas de l'ordonnancement, l'échantillonnage consiste à collecter les valeurs des variables d'état du site administré, les valeurs des variables de description de chaque job ordonnancé et du gain obtenu. Pour le problème du dimensionnement, on conserve en mémoire, à titre d'action, le nombre de machines que l'algorithme de dimensionnement a choisi de conserver en activité. Une fois qu'un job termine son exécution, le gain est calculé puis associé pour former le triplet {état, action, gain}. C'est en utilisant un ensemble de ces triplets que le modèle de Q, c'est à dire le réseau de neurones ou l'Echo State Network, est paramétré et ensuite utilisé durant la procédure de décision associée. Dans ce contexte, l'emploi du terme *échantillonnage* est approprié car la procédure décrite consiste à enregistrer des points dans l'espace des {état, action, gain} possibles de la fonction d'espérance de gain Q afin de l'estimer.

Un sous-échantillon de taille bornée des triplets est utilisé lors de chaque procédure de régression. L'utilisation de l'ensemble des triplets rencontrés au cours d'une ou plusieurs expériences est rendue difficile pour des questions de temps d'apprentissage et d'occupation mémoire [Doy00, Ven07b]. Par ailleurs, on souhaite paramétrer le modèle d'apprentissage grâce à la dynamique récente du système considéré, le site de grille. Conserver un historique trop important de l'activité d'un site administré, comme échantillon d'apprentissage de la fonction Q, altère l'efficacité du système d'ordonnancement car un nombre trop important de triplets augmente le risque de bruit dans l'échantillon. Ce choix peut conduire également à une politique sous optimale dans le cas d'un flux de demandes instationnaire, oscillant entre des périodes de demandes restreintes et importantes [BLHS97].

Régression de la fonction Q

L'essentiel de la procédure de régression a été explicité dans le chapitre 5. Nous précisons ici l'instanciation des paramètres. Deux modèles de régression ont été expérimentés pour l'apprentissage de la fonction Q, un réseau de neurone multi-couches à rétro-propagation stochastique du gradient d'erreur et un Echo State Network. Le réseau de neurone multi-couches comporte 10 neurones dans sa couche cachée, 13 neurones en entrée et 1 neurone en sortie. Le choix de cette architecture a été défini empiriquement et inspiré des travaux récent en apprentissage par renforcement en domaine continu [TDC+08].

De même, l'Echo State Network, développé selon [Jae02], est composé de 13 neurones en entrée, de 100 neurones dans son réservoir avec un taux de connectivité de 15% intra-réservoir et de 10% extra-réservoir. Les connexions intra-réservoir de l'ESN sont binaires.

6.3 Génération des traces synthétiques

Une liste de jobs synthétiques est d'abord définie par les séries temporelles associées au processus d'arrivée et au processus de service. Le premier contrôle la génération des délais d'inter-arrivée entre chaque couple de jobs consécutifs en file d'attente. Le processus de service régit l'espérance de la durée d'exécution pour chaque job. Dans la suite de cette section, deux modèles de couple, processus d'arrivée et de service, sont présentés. Ceux-ci ont été utilisé afin de produire les listes de jobs utilisées dans une partie des expériences.

Les charges synthétiques générées sont basées sur deux modèles statistiques :
- Le modèle Poisson/Exponentiel
- Le modèle Markov-Modulated Poisson Process/Exponentiel

La suite de cette section rappelle les caractéristiques de ces modèles. L'autre composante nécessaire à la définition d'une trace synthétique dans le contexte grille est la distribution des affiliations aux VOs, qui sera également présentée.

6.3.1 Le modèle Poisson/Exponentiel

Définitions

Dans ce premier modèle, la distribution des délais d'inter-arrivée suit une loi de Poisson et la distribution des temps de service, c'est à dire les durées d'exécution des jobs, suit une loi exponentielle. Cette configuration correspond donc au modèle classique Markov/Markov/P-serveur [DH98]. On rappelle ici la définition de ces lois usuelles.

Définition 6.3.1.1 *une variable aléatoire X suit une loi exponentielle de paramètre $\mu > 0$ si pour tout réel $t \geq 0$, $P(X \leq t) = \int_0^t \mu e^{-\mu u} du$; ou ce qui est équivalent, si pour tout réel $t \geq 0$, $P(X > t) = e^{-\mu t}$.*

Définition 6.3.1.2 *Une variable aléatoire X suit une loi de Poisson de paramètre $\lambda > 0$ si pour tout entier n, $P(X = n) = \frac{\lambda^n}{n!} e^{-\lambda}$.*

On définit le facteur d'utilisation d'un flot de demandes suivant les deux lois énoncées précedemment et selon le rapport $\rho = \frac{\lambda}{\mu P}$ des paramètres des distributions de Poisson et

exponentielle, avec P le nombre de serveur du système considéré (queue M/M/P). Par définition, si le facteur d'utilisation ρ est supérieur ou égal à 1, le délai d'attente en queue est infini et le système est surchargé.

Choix de paramètres expérimentaux

Une expérience sera caractérisée par, d'une part le taux f de jobs interactifs (rappelons qu'un job sera étiqueté interactif si sa durée d'exécution ne dépasse un seuil défini $\tau = 900$ secondes), d'autre part par le facteur d'utilisation du système rho. La valeur de μ ne dépend que du taux de jobs interactifs, et se déduit immédiatement de la définition de la distribution exponentielle 6.3.1.1

$$\mu = \frac{-log(1-f)}{\tau}. \tag{6.1}$$

On en déduit le taux d'arrivée λ, pour une valeur du facteur d'utilisation ρ

$$\lambda = \mu\rho P \tag{6.2}$$

Il sera possible dans les expériences d'évaluer le comportement de l'ordonnanceur avec des traces synthétiques modélisant une activité chargée mais théoriquement faisable où l'on fera varier le taux de jobs interactifs. La question sera alors d'étudier la gestion par l'ordonnanceur de ce type de job. L'objectif principal est de vérifier que dans le cas de jobs interactifs majoritaires ou minoritaires, les ordonnanceurs par apprentissage par renforcement sont capables d'assurer à ces derniers une qualité de service supérieure aux algorithmes choisis comme référence et admissible pour les utilisateurs de grille.

expérience	f	ρ	μ	λ	esp. attente (sec)	resp. jobs inter.
PE-1	0.1	0.95	1.17E-4	8.9E-3	1205	0.42
PE-2	0.1	0.99	1.17E-4	9.3E-3	15529	0.05
PE-3	0.5	0.99	7.70E-4	6.1E-2	1474	0.37
PE-4	0.9	0.99	2.56E-3	2.03E-3	431	0.67

TAB. 6.1 – Charges produites par une distribution Poisson exponentielle pour $P = 80$

Le tableau 6.1 présente les quatre paramétrisations des distributions Poisson et exponentielle qui ont été utilisées. Dans les quatre cas, on conserve un facteur d'utilisation de l'ordre de 95 à 99% et $P = 80$. Ce choix permet d'assurer des jobs en permanence dans

la queue et que l'ordonnanceur dispose effectivement d'un éventail de décisions possibles. Le taux d'interactifs varie pour chaque facteur d'utilisation afin d'observer si, notamment lorsque peu de jobs interactifs sont présents, le système parvient à produire une politique de contrôle capable de fournir une qualité de service suffisante à ces derniers.

Dans les quatre cas considérés, les délais d'attente moyens pour un ordonnancement FIFO sont suffisamment longs pour permettre à l'ordonnancement par apprentissage par renforcement d'apporter une amélioration. Plus précisement, ces paramètres ont été choisi après estimation des espérances des délais d'attente déduites de la formule d'Erlang [Tak69, Arh77] et calculées par [Wil, Wil99]. De plus, ces distributions représentent des cas où une politique FIFO est inefficace pour la gestion des jobs interactifs. En effet, on calcule, à la fin de la table 6.1, la réactivité des jobs interactifs, s'il on admet que ceux-ci ont tous une durée d'exécution de 900 secondes.

FIXME : on calcule une borne supérieure de la réactivité moyenne des jobs interactifs (préciser formule et justification)

FIXME : rappelons que larger is better et que c'est borné par 1. Donc, l'expérence 2 représent un cas extreme ou la réactivité est extrement mauvaise tandis que dans les expérience 1 3 et 4 le temps d'attente et d'execution sont du meme ordre de grandeur.

6.3.2 Le modèle MMPP/Exponentiel

Définition

Un MMPP (Markov-modulated Poisson Process) [FMH93] alterne des processus de Poisson dont le paramètre λ est défini par l'état d'une chaîne de Markov. La chaîne de Markov *module* le processus de Poisson. Cette modulation introduit des corrélations dans les délais d'inter-arrivées successifs générés par le processus. Un MMPP est un cas simple de MAP (Markovian arrival process) où l'on modélise un processus d'arrivée par une chaîne de Markov.

Le principal avantage d'utiliser un tel modèle pour la génération de délais d'inter-arrivée est la production d'un trafic à intensité transitoire. Il est possible de définir, par exemple, un SPP [Bha92] (Switched Poisson Process), ce MMPP à deux état comprend un état dont la valeur associée de λ est égale à 0. Comme l'indique le nom, on défini un processus d'arrivée avec des périodes de totale inactivité. Il est encore possible de définir un MMPP comportant des états à très forte charge correspondant à des délais d'inter-arrivée très courts et des périodes de faible charge à l'inverse.

Dans ce modèle, une distribution exponentielle est conservée pour les temps de services.

L'aspect transition du MMPP permet de créer des phases de forte charge, dans les listes de jobs, nécessaires pour tester la fiabilité de la méthode d'ordonnancement proposée. En effet, ce type de comportement en situation réelle se retrouve en situation réelle.

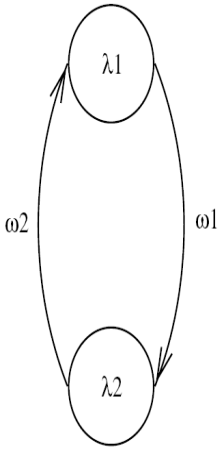

FIG. 6.3 – Processus de Poisson modulé par une chaîne de Markov à deux états

La figure 6.3 présente un MMPP à deux états avec les valeurs λ_1 et λ_2 associées. ω_1 et ω_2 représente les probabilités de transition d'un délai inter-arrivé de paramètre $lambda_1$ vers un délais d'inter arrivée de paramètre λ_2.

Choix des paramètres expérimentaux

Une question est désormais de déterminer les paramètres de ce modèle. Une première méthode serait de les apprendre à partir d'une trace réelle. On produit dans ce cas un modèle synthétique capable de reproduire la dynamique transitoire de la grille. Dans la littérature, plusieurs méthodes basées sur une optimisation de type *EM* pour estimer les paramètres d'un MMPP ont été proposée [Sit91, ER09] mais toutes souffrent d'un problème de coût de calcul et de fiabilité.

Ici, les paramètres $lambda_1$ et $lambda_2$ ont été déterminés selon la méthode utilisée dans la section précédente utilisé pour paramètriser les modèles M/M/P (poisson exponentiel). L'implémentation Matlab de Hui Li [LM07] pour la génération de délais d'inter-arrivées issus du modèle MMPP a été utilisée. $lambda_1$ et $lambda_2$ ont donc été déterminées en fonction des facteurs d'utilisation et des taux d'interactifs souhaites.

Les deux modèles choisis ont des taux de charge importants, $\rho \in \{0.95, 0.99\}$. Ces valeurs ont été choisies dans une démarche identique à celle présentée pour les choix des paramètres des modèles de Poisson et exponentiel. La proportion des jobs interactifs souhaitée est fixée et la valeur de μ est déduite, sachant que le nombre de machines $P = 80$

est fixe tout au long des expériences utilisant ces flux. On en déduit ainsi les valeurs de λ_1 et λ_2 pour les deux MMPP considérés. Les probabilités de transition sera choisi pour exhiber des flots de données au comportement très différenciés.

expérience	f	μ	λ_1	λ_2	ρ_1	ρ_2	esp. attente (sec)	resp. jobs inter.
MMPP-1	0.1	1.17E-4	0.89E-2	0.93E-2	0.95	0.99	4069	0.18
MMPP-2	0.5	7.70E-4	5.85E-2	6.10E-2	0.95	0.99	435	0.67

TAB. 6.2 – Propriété des charges produites par distributions MMPP/Exponentielle.

La table 6.2 décrit les paramètres des modèles de flot de demandes employant un MMPP à deux états pour la génération des temps d'inter-arrivée. Les probabilités de transition sont $\omega_1 = 0.2$ et $\omega_2 = 0.8$ afin de modéliser comprennant des pics de charge. Les espérances de durée d'attente sont calculées pour chaque état du MMPP correspondant, suivant la méthode décrite dans la section 6.3.1. Les valeurs de réactivité moyennes, sous l'hypothèse d'un ordonnancement FIFO et d'une classe de jobs interactifs uniquement composée de tâches ayant un délai d'exécution de 900 secondes, sont faibles dans les deux cas. On les calcul selon la forume

$$\omega_1 * esp\_attente_{\omega_1} + \omega_2 * esp\_attente_{\omega_2}$$

6.3.3 Caractéristiques des charges synthétiques

Dans ce paragraphe nous appelerons charge, les séries temporelles des sommes des temps de service aggrégées par quart d'heure. Les figures 6.3.3 montrent la charge pour chacune des traces générées. Bien que la distribution des temps de service reste identique au modèle Poisson Exponentiel chargé, le modèle MMPP produit des fluctuations dans la quantité de demande de calculs. Il permet de générer des périodes de demandes intenses intercallées d'intervalles de temps à demandes modérées. Des listes de 10000 jobs sont générées, ce qui est la limite de la quantité traitable en simulation.

La figure 6.3.3 présente la somme des durées d'exécution par quart d'heure de traces de 10000 jobs générées via les MMPP présentés dans la section précédente. L'amplitude des transitions entre les valeurs est d'un ordre de grandeur supérieur à celles observées dans le cas des charges générées par des distributions Poisson exponentielles simples. Par exemple, le cas MMPP-1 présente plusieurs cas d'augmentation importante de charge, de l'ordre de 130 heures de calcul, en l'espace d'une demi heure. A contratio, les évolutions de charge observées dans le cas PE-4 ne dépasse pas 5 heures sur un même intervalle de temps d'une demi heure.

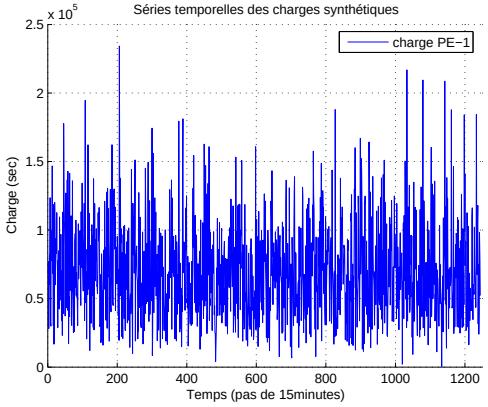

FIG. 6.4 – Série temporelle de la charge PE-1

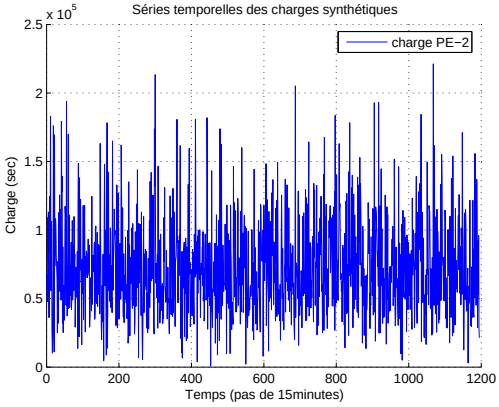

FIG. 6.5 – Série temporelle de la charge PE-2

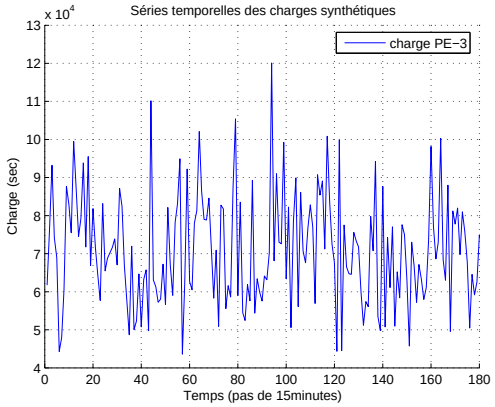

FIG. 6.6 – Série temporelle de la charge PE-3

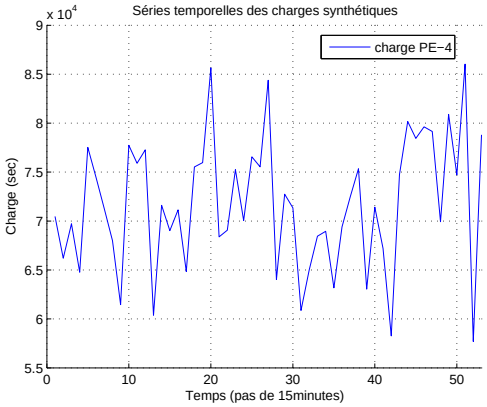

FIG. 6.7 – Série temporelle de la charge PE-4

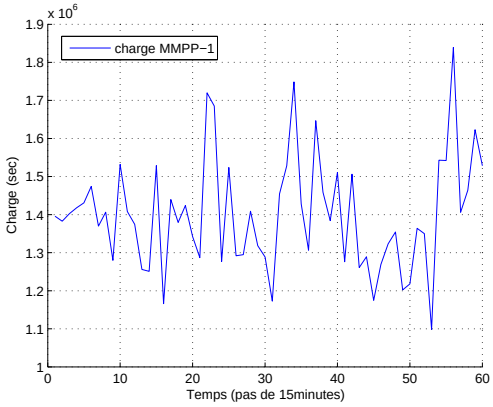

FIG. 6.8 – Série temporelle de la charge MMPP-1

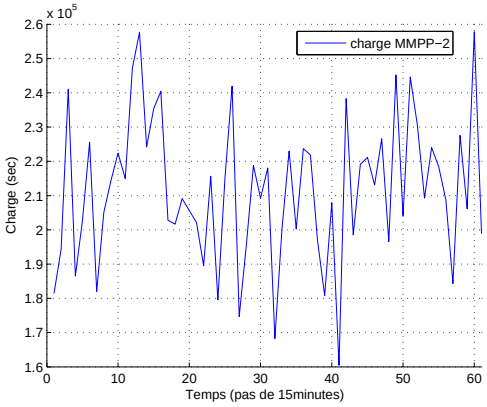

FIG. 6.9 – Série temporelle de la charge MMPP-2

Le modèle permet ainsi la création d'une distribution des délais d'inter-arrivée à queue lourde observée dans les traces réelles de flot de demande de grille de calcul [LMW06]. La figure 6.3.3 présente les histogrammes des délais d'inter-arrivée des charges MMPP, illustrant ce phénomène.

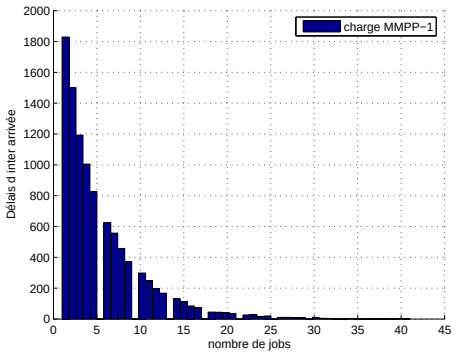

FIG. 6.10 – Distribution des délais d'inter-arrivée de la charge MMPP-1

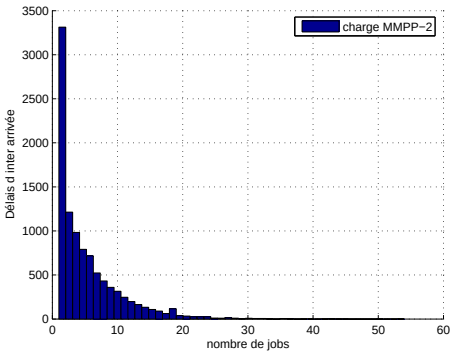

FIG. 6.11 – Distribution des délais d'inter-arrivée de la charge MMPP-2

6.3.4 Distribution des organisations virtuelles

Pour être compatible avec le modèle exposé dans le chapitre 5, le format des jobs contient un attribut organisation virtuel. Pour générer cet attribut dans les distributions synthétiques nous avons utilisé les distributions des VOs correspondant aux traces EGEE. Nous renvoyons à la section précédente pour la présentation de cette distribution.

Dans l'infrastructure EGEE, chaque job appartient à une organisation virtuelle. L'infrastructure EGEE compte plus d'une centaine d'organisations virtuelles et chaque administrateur de site de la grille EGEE est en droit de privilégier des VOs affiliées plus particulièrement au site dont il a la responsibilité.

Dans un souci de réalisme, des distributions de VOs correspondant aux activités enregistrées au sein d'EGEE ont été utilisées. Dans le cas des listes jobs réelles, la VO de chaque jobs a été conservée. Dans le cas des traces synthétiques, la distribution observée dans les traces réelles a été reprise. Il est important de noter que cette distribution est propre au centre de calcul d'où les traces d'exécution sont tirées. Il appartiendra à l'administrateur système du site supervisé de définir les objectifs d'équité en fonction de site.

Dans le cas du site du Laboratoire de l'Accélérateur Linéaire de l'Université Paris Sud / CNRS, une part importante de la demande de service provient d'une organisation virtuelle, Atlas, groupement d'expérimentation de la physique des particules.

Organisation	Proportion
Atlas	53%
Biomed	2%
CMS	17%
DZERO	8%
ILC	1%
LHCB	3%
autres VOs	16%

TAB. 6.3 – Distribution des VOS dans le site LAL de la grille EGEE

L'apprentissage d'un tel système peut être simplifié en regroupant les organisations virtuelles ayant les taux de présence les plus faibles. L'objectif est d'obtenir des distributions de poids homogènes. En effet, la fonction de gain d'équité présentée dans le chapitre 4 est calculée en fonction de l'écart le plus important observé dans la proportion des différentes organisations virtuelles prises en compte. Dans le cas où une organisation virtuelle présente une fréquence d'apparition trop restreinte, le gain d'équité risque d'être anormalement faible du fait de la non-présence de cette VOs dans la distributions des jobs au cours d'une partie importante de l'expérience. Ainsi, en regroupant les organisations virtuelles ayant les fréquences d'apparitions les plus faibles dans la liste des jobs, ont rétabli des mesures de faishare plus réaliste et on facilite l'apprentissage d'une politique de contrôle efficace.

Organisation	Proportion
Atlas	53%
Biomed DZERO LHCB ILC	14%
CMS	17%
Autres VOs	16%

TAB. 6.4 – Distribution uniformisée des VOS dans le site LAL de la grille EGEE

La table 6.7 présente les poids d'équité où ont été regroupées les organisations virtuelles présentant les taux d'apparition les plus faibles de la table 6.6.

6.4 Les traces réelles

6.4.1 Source des données

La charge réelle utilisée dans les expériences est extraite des logs TORQUE [Sta06] d'un site de calcul de la grille EGEE situé au Laboratoire de l'Accélérateur Linéaire de

```
03/18/2006 02 :17 :19 ;E ;333.grid10.lal.in2p3.fr ;user=dtes
group=dteam jobname=STDIN queue=dteam
ctime=1142644255 qtime=1142644255 etime=1142644255
start=1142644256 exec_host=grid17.lal.in2p3.fr/2 Resour-
ceList.cput=02 :00 :00 ResourceList.neednodes=1 :ppn=1
ResourceList.nodect=1 ResourceList.nodes=1 :ppn=1 Resour-
ceList.walltime=02 :30 :00 session=13321 end=1142644639
ExitStatus=0 resourcesUsed.cput=00 :00 :12 resourcesU-
sed.mem=52192kb resourcesUsed.vmem=149516kb resource-
sUsed.walltime=00 :06 :23

03/18/2006 04 :09 :50 ;S ;317.grid10.lal.in2p3.fr ;user=bio150
group=biomed jobname=STDIN queue=biomed
ctime=1142627637 qtime=1142627637 etime=1142627637
start=1142651390 execHost=grid24.lal.in2p3.fr/0
ResourceList.cput=24 :00 :00 Resource-
List.neednodes=grid24.lal.in2p3.fr ResourceList.nodect=1 Re-
sourceList.nodes=1 :ppn=1 ResourceList.walltime=36 :00 :00

03/18/2006 05 :10 :41 ;Q ;335.grid10.lal.in2p3.fr ;queue=dteam
```

FIG. 6.12 – Trois évènements extraits du système de log de TORQUE Maui

l'université Paris Sud (ref). Les traces s'étendent du 24 Juillet au 12 Septembre 2006. Elles comptent un total de 10000 jobs répartis sur 50 jours d'utilisation du site. Le site comprenait à cette époque un total d'une centaine de machines pour la plupart mono coeur en activité constante avec des périodes de maintenance. Les tâches de supervision (*monitoring*) du site sont effectuées par l'intermédiaire d'une organisation virtuelle dédiée mais passant par la même chaîne de traitement que les jobs des utilisateurs finaux.

Dans ce système, les logs de l'ordonnanceur PBS, enregistrés par le système TORQUE [Sta06], sont formées d'un ensemble d'évènements. L'arrivée d'un job, le démarrage de son exécution ainsi que sa terminaison sont consignés avec un ensemble d'indicateurs correspondants. La figure 6.4.1 est un exemple des traces produites sur le site de grille étudié. Le système d'ordonnancement et de log fonctionne à un niveau d'abstraction de l'unité de calcul. Ainsi, une tâche est affectée à un processeur sur une machine. Un processeur ne traite que d'un job à un temps donné et la préemption n'est pas possible.

6.4.2 Acquisitions

Les traces s'étendent du 24 Juillet au 12 Septembre 2006. Elles comptent un total de 10000 jobs répartis sur 50 jours d'utilisation du site.

Les évènements extraits des logs sont composés d'une date, d'un type d'évènement, de la machine émettrice du message de log, en l'occurrence celle qui exécute l'ordonnanceur PBS. Selon le type de l'évènement, des champs de description du job sont ajoutés :

- Utilisateur
- VO
- nom du job
- nom de la queue
- la durée d'exécution
- la taille de mémoire utilisée
- la machine sur laquelle le job s'est exécuté.

Le format évènementiel du log a nécessité le développement d'un analyseur syntaxique spécifique pour l'extraction des données nécessaires aux simulations. C'est en se basant sur l'évenement de fin d'exécution d'un job qu'une représentation concise des jobs est construite sur une période de temps donnée. La date d'arrivée du job est conservée, ainsi que sa date de début, de fin d'exécution et la VO associée.

Le nombre de machines utilisées lors d'une simulation est estimé en calculant le nombre de machines distinctes qui ont été mentionnées dans le log PBS. Ce calcul est essentiel, car un site de grille subit fréquemment des pannes ainsi que des changements de configuration matérielle.

On suppose également que le parc de machine constituant le cluster de calcul est homogène et donc que la durée d'exécution d'un job enregistrée dans l'évènement de fin de ce dernier et un indicateur fiable de la charge de travail qu'il représente. Cette hypothèse d'homogénéité serait gravement érronée à l'échelle de la grille, comme on considère ici des séries temporelles internes à un site, elle est acceptable. Les sites de grilles sont institutionnels donc raisonnablement cohérent.

6.4.3 Segmentation

Cette section traite de la question de la segmentation de la trace récoltée dans PBS. Le but est de créer des benchmarks de politique d'ordonnancement représentatifs de l'activité du site. Pour des questions de temps de traitement, il est impossible d'effectuer des simulations d'ordonnancement au delà d'un nombre de job donné. Un segment de log se

définit comme un ensemble de jobs dont les dates d'arrivée en queue se suivent. La question est d'extraire un segment afin d'effectuer des simulations réalistes du comportement de la grille et représentative vis à vis des performances de l'ordonnancement.

Ce choix d'un extrait de log significatif au regard des expériences a été effectué selon plusieurs critères. Les indicateurs suivant ont été considérés dans le choix :

- **Stabilité matérielle du site**. L'objectif est d'identifier des périodes où la configuration matérielle du site reste inchangée. La stabilité de la configuration est indispensable à la validité des analyses qui compareront les performances de l'ordonnancement par apprentissage et celles de l'ordonnanceur natif du site. En effet, les performances en terme de délai d'attente et de qualité de service en général sont, évidemment, fortement dépendante du nombre de machines disponibles sur le site au moment de l'expérimentation. Ce facteur est indispensable à la validité des expériences dont le résultat sera basé sur une comparaison entre les résultats obtenus par le biais du simulateur et les résultats calculés sur les logs d'EGEE. Les performances en terme de délai d'attente et de qualité de service en général sont, évidemment, fortement dépendante du nombre de machines disponibles sur le site au moment de l'expérimentation. Il n'est pas simple a valider, en effet, le site, comme la plupart des sites de grille est très dynamique, en expansion continue. D'autre part, les activités de maintenance du site peuvent entraîner un arrêt momentané d'une partie du cluster.
- **Distribution des VOs**. On souhaite utiliser une période d'activité de la grille EGEE où la distribution des VOs soit représentative. Dans notre cas, une grande partie des jobs soumis est issue de l'organisation virtuelle Atlas. Cependant une part non négligeable de l'activité du site doit être dédiée à plusieurs autres VOs.
- **Distribution des durées d'exécution**. L'objectif, comme pour la distribution des VOs est de choisir un intervalle de temps où la distribution des durées d'exécution reflète l'activité de la grille. Ainsi, il a fallu éviter les périodes de lancement de tests de la VO d'administration d'EGEE où des pics de soumission de job extrêmement courts masquent l'activité classique de la grille. Ces jobs constituent une exception à la règle, en cela qu'ils sont exécutés concurremment aux jobs des utilisateurs. les périodes qui les contiennent ne peuvent donc malheureusement pas être incluses dans un benchmark. Il faut d'ailleurs remarquer que des jobs de ce type sont lancés périodiquement (toutes les 3 heures) ; mais en fonctionnement normal, ils sont suffisamment peu intrusifs pour que la perturbation qu'ils apportent puisse être négligée.
- **Nombre significatif de jobs**. Le nombre 10000 a été défini comme suffisant pour représenter une période d'activité de la grille qui correspond en moyenne à 4 semaines d'activité du site. Ce nombre permet également de conserver des durées de simulation faisables.

– **Facteur d'utilisation du segment**. Pour estimer l'utilisation du site, un indicateur synthétique a été défini, le *coefficient d'utilisation*. Comme détaillé, ci après, la mesure d'utilisation a permis d'identifier des périodes d'activité où la pression sur les ressources est moyenne ou forte.

Le facteur d'utilisation du segment est le rapport entre la moyenne des durées d'exécution et la moyenne des délais d'inter-arrivée pour les jobs du segment.En effet, à la différence du cas de la génération synthétique de jobs, présenté dans la section précédente, où les distributions sont connues à l'avance et permettent donc le calcul du coefficient d'utilisation $\rho = \lambda/\mu P$, nous estimons une valeur qui renseigne sur les éventuels pics de charge dans l'échantillon est estimée. Ceci n'implique aucune hypothèse concernant la forme des distributions d'inter-arrivée et de service dans ces segments.

L'emploi de ces critères de choix pour segmenter la trace est en grande partie automatisable ; un outil logiciel capable d'identifier et d'extraire des segments de jobs d'une taille donnée, répondant partiellement aux critères énoncés, a été développé. Malgré cela, le processus n'a pu être entièrement automatisé, en effet, la grande variabilité des processus à l'oeuvre dans la grille rend les indicateurs synthétiques insuffisants. Une analyse manuelle a été nécessaire afin de considérer la distribution des VOs et des durées d'exécution parfois très différentes selon les segments. Enfin, une discussion avec l'administration système du site concerné a confirmé qu'aucune modification matérielle ou logicielle majeure n'a eu lieu dans l'intervalle de temps représenté par ce segment. La section suivante décrit la trace qui sera utilisée pour la partie réelle des expériences du chapitre suivant. Dans la suite, le segment choisie par la méthode décrite sera nommé *l'échantillon*

6.4.4 Statistiques descriptives de l'échantillon

La figure 6.13 présente la distribution des temps de service pour l'échantillon. On constate une variété importante de comportements en terme de durée d'exécution des demandes soumises à la grille. Des demandes de calcul d'une trentaine d'heures côtoient des demandes de tâches de quelques minutes uniquement. On note également que presque 50% des jobs soumis ont une durée d'exécution inférieurs à 15 minutes (900 secondes), ce qui démontre une présence importante de jobs de type interactif.

La figure 6.14 présente la distribution d'inter-arrivée sur le segment choisi pour l'expérimentation. On remarque ici que 80% des délais d'inter-arrivée des jobs ne dépasse pas 10 minutes. On remarque également que des périodes de forte demande figurent dans l'échantillon.

Un second critère de choix est la disparité des VOs des jobs présents dans l'échantillon. Bien qu'une majorité des tâches revienne à l'organisation virtuelle Atlas, utilisateur majeur

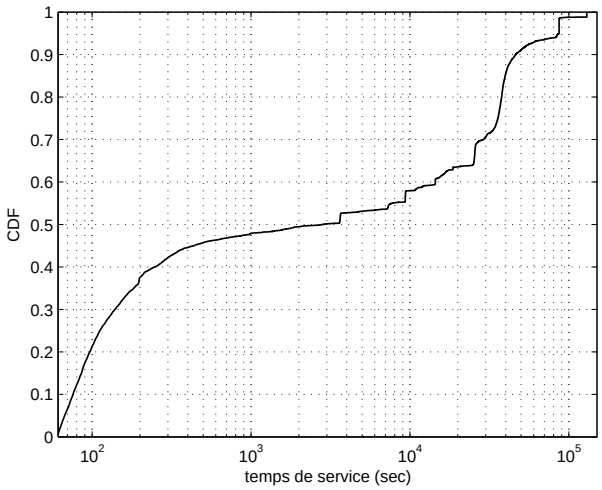

FIG. 6.13 – Distribution des temps de service

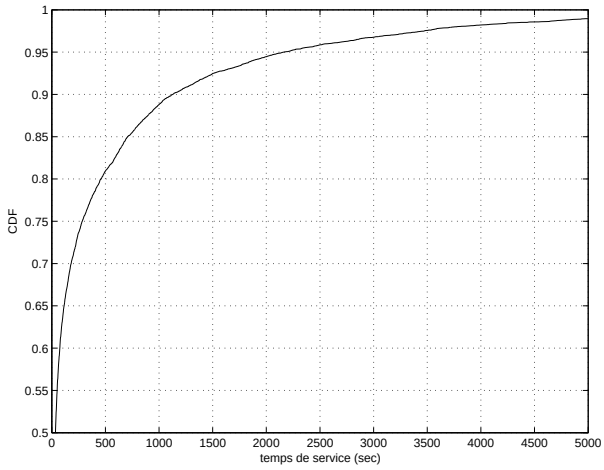

FIG. 6.14 – Distribution des délais d'inter-arrivée

de la grille EGEE, on constate la présence de tâches de calcul liés à la recherche médicale, de l'organisation virtuelle *Biomed*.

La figure 6.15 présente la distribution des temps de service de l'échantillon par VO. On constate ici une grande différence de profil d'exécution en fonction de l'organisation virtuelle. Alors que seulement 10% des jobs lancés depuis la VO Atlas ne dépasse pas le quart d'heure d'exécution, c'est plus de 50% des jobs des VOs *Biomed, LHCB, DZERO* agrégées et *ILC* qui s'inscrivent dans cet intervalle. L'utilisation d'un système décisionnel basé sur la caractérisation des jobs semblent donc, encore une fois, appropriée pour fournir une qualité de service différenciée.

Le tableau 6.4.4 présente les moments des distributions des temps d'exécution pour

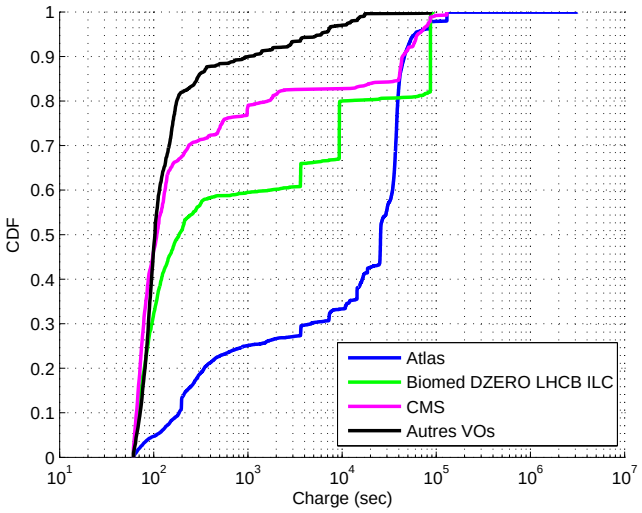

FIG. 6.15 – Distribution des temps de service d'une trace EGEE

Type job	Moyenne	Variance
batch	37024	78756
interactif	160	134

TAB. 6.5 – Statistiques des temps d'exécutions des jobs du segment choisi (en secondes)

les jobs identifiés comme batchs ou interactifs dans l'échantillon choisi. On remarque ainsi dans les deux profils de jobs une variance importante, notamment pour les jobs batchs. On observe également une différence d'échelle de deux ordres de grandeur de ces valeurs entre les deux types de jobs considérés.

6.4.5 Distribution des organisations virtuelles

Des distributions de VOs correspondant aux activités enregistrées au sein d'EGEE ont été utilisées. Dans le cas des listes jobs réelles, la VO de chaque jobs a été conservée. Dans le cas des traces synthétiques, la distribution observée dans les traces réelles a été reprise. Cette distribution est donc propre au centre de calcul d'où les traces d'exécution sont tirées. Il appartiendra à l'administrateur système du site supervisé de définir les objectifs d'équité en fonction du site.

Dans le cas du site du Laboratoire de l'Accélérateur Linéaire, une part importante de la demande de service provient d'une organisation virtuelle, Atlas, groupement d'expérimentation de la physique des particules.

L'apprentissage d'un tel système peut être simplifié en regroupant les organisations

Organisation	Proportion
Atlas	53%
Biomed	2%
CMS	17%
DZERO	8%
ILC	1%
LHCB	3%
autres VOs	16%

TAB. 6.6 – Distribution des VOS dans le site LAL de la grille EGEE

virtuelles ayant les taux de présence les plus faibles. L'objectif est d'obtenir des distributions de poids homogènes. En effet, la fonction de gain d'équité présentée dans le chapitre 4 est calculée en fonction de l'écart le plus important observé dans la proportion des différentes organisations virtuelles prises en compte. Dans le cas où une organisation virtuelle présente une fréquence d'apparition trop restreinte, le gain d'équité risque d'être anormalement faible du fait de la non-présence de cette VOs dans la distributions des jobs au cours d'une partie importante de l'expérience. Ainsi, en regroupant les organisations virtuelles ayant les fréquences d'apparitions les plus faibles dans la liste des jobs, on rétabli des mesures de équité plus réaliste et on facilite l'apprentissage d'une politique de contrôle efficace.

Organisation	Proportion
Atlas	53%
Biomed DZERO LHCB ILC	14%
CMS	17%
Autres VOs	16%

TAB. 6.7 – Distribution uniformisée des VOS dans le site LAL de la grille EGEE

La table 6.7 présente les poids d'équité où ont été regroupées les organisations virtuelles présentant les taux d'apparition les plus faibles de la table 6.6.

Résumé

Ce chapitre a présenté les composants logiciels de mise en oeuvre et d'expérimentation des algorithmes d'allocation de ressources présentés au chapitre 5. Cette plateforme est composée des algorithmes de simulation de la dynamique du site ainsi que des algorithmes d'apprentissage de politique. L'architecture logicielle de l'ensemble a été conçue de telle sorte que les algorithmes d'ordonnancement se comportent de manière indépendante de

l'environnement réel ou de simulation dans lesquels ils pourraient être utilisés. Ensuite, les traces d'activités synthétiques et réelles utilisées pour la validation ont été décrites. Dans le cadre des traces synthétiques, la paramétrisation des différents modèles de charge $M/M/P$ a été justifiée. L'objet était de fournir des traces réalistes et suffisamment chargées pour permettre d'évaluer la qualité des politiques de contrôle issues de la méthodologie d'apprentissage par renforcement. Nous avons enfin présenté le système de log du gestionnaire de site d'EGEE d'où sont issues les traces d'activités réelles. L'accent a alors été porté sur les problèmes du choix d'un segment d'activité significatif du comportement de la grille. Le chapitre suivant présentera les expériences exploitant ces développements.

Chapitre 7

Expériences

7.1 Introduction

Ce chapitre propose un ensemble de validations expérimentales des modèle d'apprentissage par renforcement pour l'allocation de ressources que nous avons proposées au chapitre 6. Les performances sont étudiées sur les différents types de charges synthétiques et sur la trace extraite de l'activité réelle d'un site de la grille EGEE, grâce à la plate-forme d'expérimentation présentée au chapitre 6.

La première partie de ce chapitre présente deux catégories d'expériences portant sur les traces synthétiques. Toutes deux portent uniquement sur le problème de l'ordonnancement, donc dans le cadre d'un environnement non élastique. L'objectif de cette première série d'expériences est de valider bon le comportement des algorithmes dans un contexte stationnaire. Une configuration d'ordonnancement faisable en termes de d'équité est d'abord expérimentée, où l'objectif d"équité demandé correspond à la distribution des organisations virtuelles des jobs de la trace. Ensuite, les résultats des mêmes expériences sont présentés avec un objectif d'équité infaisable au regard des distributions des organisations virtuelles des jobs. L'objectif de cette seconde série d'expériences sera d'analyser la robustesse des algorithmes d'apprentissage par renforcement dans ce cas particulier, fréquemment observé.

Dans une seconde partie, les résultats obtenus dans le cas d'une trace d'exécution réelle d'EGEE sont détaillés. Après avoir analysé les performances observées dans le cas d'un nombre fixe de machines, une dernière section sera consacrée au contrôleur de dimensionnement dynamique du cluster. L'analyse présentera l'apport des algorithmes proposés vis à vis de l'ordonnancement natif. Cette partie montre qu'un second contrôleur, adjoint au processus d'ordonnancement, sera en mesure d'assurer l'élasticité du site en améliorant sa performance en utilisation.

7.2 Protocole expérimental

Cette section présente plusieurs considérations valables pour toutes les expérimentations effectuées. Ces choix opérationnels d'expérimentation, tels que les indicateurs de performance, l'hypothèse faite sur la connaissance des durées d'exécution des jobs et les variables de pondération des fonctions d'objectifs des systèmes apprenants sont explicités.

7.2.1 Mesures de performance

Les fonctions d'objectif présentées dans les sections 5.2.3 et 5.3.2 permettent de mesurer la qualité de l'apprentissage par renforcment, comme elle ont un sens physique, ce sont aussi

des indicateurs de performances. Il faut cependant les compléter par d'autre afin de nous montrer s'il ont a bien choisir nos fonctions d'objectif. La capacité à suivre une politique de contrôle définie, par l'apprentissage par renforcement, est évaluée par l'analyse de la performance de réactivité ainsi que la comparaison de cet indicateur avec l'algorithme de référence. Si cet indicateur est optimisé de manière satisfaisante, il reste encore à montrer que cette mesure capture correctement les espérances des utilisateurs en terme de qualité de service.

La satisfaction de l'utilisateur s'exprime principalement par la durée d'attente en queue qui est reportée dans l'intégralité des simulations effectuées. Une attention particulière sera portée à la valeur de cette durée dans le cas des jobs interactifs. Concernant l'équité, la différence entre la mesure effectuée par l'ordonnanceur de référence (FIFO pour la charge synthétique, et l'ordonnanceur natif, PBS, pour la charge EGEE, qui constitue l'état de l'art dans le domaine) et l'équité de notre ordonnanceur sera utilisée comme indicateur (cf. section 5.2.3).

La performance d'utilisation est reportée selon l'équation définie dans la section 5.3.2. Elle sera employée dans la dernière partie des expérimentations utilisant des traces réelles, consacrées à l'évaluation du système d'allocation dynamique de ressources.

En outre, nous rapporterons systématique, pour chacun des indicateurs utilisés, les moments de distributions mais aussi la médiane. En effet, on constatera souvent que la variance est tres grande et donc que seul la médiane a un sens.

7.2.2 Gestion des durées d'exécution

Plusieurs variables de description de l'environnement et de l'action des modèles de décision de Markov considérés nécessitent, afin de pouvoir être calculées, une connaissance des durées d'exécution des jobs en queue (cf. section 5.2.1). De modèles de réprésentation des durées d'exécution seront utilisées dans les expériences présentées. Le modèle *oracle* prend pour hypothèse que les durées d'exécution sont connues à priori. Dans le modèle *estimé*, les durées d'exécution ne sont plus supposées connues. Elles sont estimée, pour chacune des classes (batch et interactif), à partir de la médiane des durées d'exécution calculées sur une fenêtre.

Bien qu'idéale, le modèle oracle n'est pas totalement irréaliste car, dans l'infrastructure EGEE, l'utilisateur a la possibilité de fournir une estimation de la durée d'exécution du job lors de la soumission de celui-ci, par l'intermédiaire de son fichier *JDL* (cf. section 4.3). On considère cette fiable, car si l'exécution dépasse le temps estimé, EGEE (PBS) arrêtera prématurément l'exécution du job [Ba08] et si l'estimation du temps d'exécution

est excessive, il sera placé dans une file où ses prédécesseurs sont des jobs longs (cf. section 4.1). L'utilisateur donc aucun intérêt à fournir une information volontairement erronée. L'hypothèse est faite qu'un utilisateur a connaissance de l'ordre de grandeur de la durée d'exécution des jobs qu'il soumet, ce qui est généralement le cas. Cette hypothèse est déjà irréaliste dans un environnement complètement contrôlé, dans notre contexte cette hypothèse est irréalisable car elle suppose une connaissance exacte des durées d'exécution. L'intérêt de la méthode est de fournir une approximation de la borne supérieure des performances.

Dans le modèle *estimé*, les durées d'exécution ne sont plus supposées connues. Elles sont estimées, pour chacune des classes (batch et interactif), à partir de la médiane des durées d'exécution calculées sur une fenêtre. Cette hypothèse est réaliste : les données historiques sont accessibles en ligne par le biais de l'intergiciel de la grille. A ce propos, on peut se poser la question de la précision nécessaire au bon fonctionnement des systèmes de décision utilisés. En fait, nous verrons que meme l'estimation très grossiere que nous faisons est suffisante.

7.2.3 Paramètres expérimentaux

Pondération des fonctions d'objectif

Deux variables d'ajustement sont respectivement utilisées dans les fonctions de gain des modèles de décision (cf. section 5.4). Dans un premier temps, λ_s régule la part de la réactivité dans le MDP d'ordonnancement vis à vis de l'objectif d'équité. D'autre part, λ_e régule la part du dimensionnement dans le MDP d'ordonnancement vis à vis de l'objectif de la réactivité. Pour toutes les expériences synthétiques, on choisit $\lambda_s = \lambda_e = 0.5$.

Paramètres de l'algorithme d'apprentissage par renforcement

Le paramètre ϵ gérant le compromis exploration-exploitation de l'algorithme d'apprentissage par renforcement *SARSA* est fixé à 0.05, le paramète d'horizon γ est réglé à 0.8 et le facteur d'apprentissage ν est réglé à 0.2. Les valeurs de ces paramètres ont été déterminés expérimentalement et sont cohérents avec les principaux travaux de l'état de l'art du domaine [Tes02, Ven05, TDC+08]. La procédure de régression de la fonction Q est relancée à une période de $n = 500$ décisions dans le cas du contrôleur d'ordonnancement. Concernant le contrôleur de dimensionnement, cette procédure a lieu tout les $m = 10$ décisions, sachant qu'une décision de dimensionnement a lieu après l'ordonnancement d'une centaine de jobs.

On rappelle que le réservoir de l'Echo State Network est composé de 100 neurones avec une fonction d'activation sigmoidale. Les poids sont aléatoirement fixés avec 10% de connectivité entre les neurones du réservoir et 15% de connectivité entre les neurones du réservoir et les neurones de sortie. La taille du réservoir a, encore une fois, été déterminée expérimentalement.

7.3 Cas synthétiques : présentation

7.3.1 Notations

Pour les résultats proposés dans cette partie, les algorithmes de type apprentissage par renforcement utilisant un Echo State Network seront nommé *RL-ESN*. Ceux utilisant un réseau de neurones multicouche seront désignés sous de le nom de *RL-NN*. Le suffixe *-ORA* sera adjoint lorsque le système d'apprentissage par renforcement utilisera les durées d'exécution réelles des jobs et *-EST* lorsqu'une estimation en sera faite.

7.3.2 Organisation

Ces premières expériences étudient le cas oú le nombre de machine est fixé au cours de chaque simulation avec un modèle *oracle* des durées d'exécution des jobs. L'objectif est de démontrer que l'apprentissage par renforcement est un bon candidat pour assurer un niveau de réactivité suffisant pour les jobs batchs et interactifs issus de flots de demandes divers. Cette série d'expériences étudie également les performances en terme d'équité. Les performances des algorithmes d'ordonnancement basés sur un apprentissage par renforcement sont comparées à celles d'un algorithme de référence, l'ordonnancement FIFO. Les listes de jobs utilisées sont générées selon les paramètres présentés dans la section 6.3. Les résultats pour les jobs batchs et interactifs sont analysés séparément afin d'évaluer le gain pour chacun de ces deux types ainsi que des résultats totaux regroupant l'ensemble des jobs.

L'expérimentation s'organise en deux parties distinctes. Dans un premier temps, on considèrera un *ordonnancement faisable* en terme d'équité, c'est à dire que les exigences de l'administration système du site en terme d'équité seront équivalentes à la distribution des VOs des jobs employés pour les simulations. Dans une seconde partie, les poids d'objectif d'équité seront modifiés pour se situer dans un cas d'*ordonnancement infaisable*. L'objet sera d'étudier la robustesse de l'apprentissage par renforcement dans un tel contexte.

Dans cette expérience, la configuration d'équité se compose de 4 VOs, avec les poids respectifs (0.53, 0.14, 0.17, 0.16). Comme expliqué précédemment, la distribution des VOs

des jobs utilisée suit également cette distribution. Les résultats en terme de réactivité sont présentés dans une première partie puis les performances en terme de durée d'attente pour les cinq charges synthétiques proposées suivis des performances d'équité.

7.4 Cas synthétique : Ordonnancement faisable, étude de la réactivité

7.4.1 Charge PE-1

Les statistiques de réactivité pour l'expérience utilisant la charge PE-1 sont résumées dans la table 7.1. Dans ce premier cas, où le taux de jobs interactifs est de 10%, les deux algorithmes d'apprentissage par renforcement apportent une amélioration pour ce type de job tout en conservant une performance équivalente pour les jobs batchs.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy	Std	Médiane	Moy	Std	Médiane
FIFO	0.79	0.23	0.87	0.45	0.39	0.27	0.76	0.26	0.86
RL-NN-ORA	0.80	0.23	0.90	0.48	0.39	0.33	0.77	0.27	0.89
RL-ESN-ORA	0.83	0.25	0.96	0.56	0.38	0.54	0.80	0.27	0.96

Tab. 7.1 – Réactivité, trace synthétique PE-1.

Concernant les performances calculées sur l'ensemble des jobs, l'amélioration est de 5% en faveur des algorithmes d'apprentissage. Ce résultat s'explique essentiellement par la faible proportion de jobs interactifs. En effet, la fonction d'objectif favorise les jobs courts alors que les jobs très longs sont quasiment équivalents en terme de réactivité. On note toutefois une amélioration de 12% de la médiane obtenue par apprentissage par renforcement vis à vis de FIFO.

La figure 7.1 présente les distributions cumulatives inverses (i.e. $P(W > x)$ comme une fonction de x) des gains de réactivité obtenus par les jobs batchs et interactifs. Dans ce cas, la maximisation de la valeur obtenue sera le comportement attendu (*Larger is Better*). Dans ce cas, la médiane pour les jobs interactifs ne dépassent pas une réactivité de 0.3 par un ordonnancement FIFO alors que cette même proportion de jobs bénéficie d'une réactivité au moins égale à 0.6 dans le cas d'un ordonnancement RL-ESN. L'amélioration, bien que faible en moyenne s'effectue donc principalement dans le cas des jobs interactifs. Dans ce premier cas, l'apport du modèle RL-NN semble négligeable en comparaison de l'apport du modèle RL-ESN. Ce premier résultat semble conforté l'hypothèse de la nécessité d'utilisation d'un modèle à mémoire causée par une représentation incomplète

de l'environnement étudié.

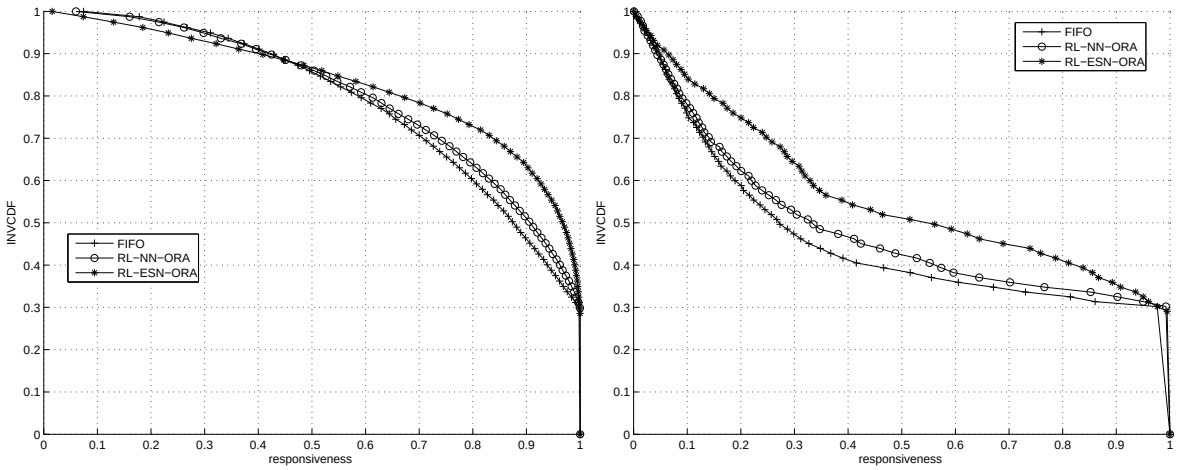

FIG. 7.1 – Distribution cumulative inverse de la réactivité des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-1.

7.4.2 Charge PE-2

Les statistiques de réactivité pour l'expérience utilisant la charge PE-2 sont résumées dans la table 7.2. Dans ce second cas, le taux d'interactifs reste de 10% mais le facteur d'utilisation ρ passe de 0.95 à 0.99. Les performances sont sensiblement équivalentes à celles énoncées dans la première distribution. La réactivité des jobs interactifs est améliorée de 5%. La moyenne de réactivité pour les jobs batchs bénéficie également d'une amélioration de 5%. En comparaison avec la première expérience, l'augmentation de la charge du système détériore les performances des jobs interactifs pour l'ensemble des algorithmes. L'algorithme d'apprentissage par renforcement utilisant un ESN bénéficie d'une amélioration de 17% en termes de médiane vis à vis de FIFO en ce qui concernant les performances sur l'ensemble de la trace. Les algorithmes RL-NN et FIFO conservent des performances équivalentes. Ces résultats globaux justifient la nécessité de détailler ces résultats par type de job.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy	Std	Médiane	Moy	Std	Médiane
FIFO	0.77	0.22	0.84	0.38	0.37	0.19	0.73	0.27	0.81
RL-NN-ORA	0.79	0.23	0.87	0.41	0.37	0.25	0.75	0.27	0.85
RL-ESN-ORA	0.81	0.27	0.96	0.45	0.35	0.30	0.77	0.30	0.95

TAB. 7.2 – Réactivité, trace synthétique PE-2.

La figure 7.2 présente les distributions cumulatives inverses des valeurs de réactivité pour les jobs batchs et interactifs de la charge PE-2. L'augmentation de charge altère les résultats obtenus. En effet, en augmentant la valeur de charge, on augmente le nombre moyen de jobs présents en queue à tout moment, augmentant ainsi le délai d'attente moyen en queue et diminuant du même coup l'espérance de réactivité pour tout les jobs de la trace et notamment des jobs interactifs, peu nombreux. Dans le cas des jobs batchs, l'intersection des courbes se situe à un gain de réactivité de 0.5 environ avec 85% des jobs ayant une valeur de réactivité au moins égale à celle ci. Dans le cas d'un ordonnancement RL-ESN 60% des jobs batchs ont une réactivité supérieure à 0.9 alors que 40% de ces jobs bénéficient de cette performance dans le cas de FIFO et RL-NN. Par ailleurs, la moitié des jobs batchs ordonnancés par FIFO bénéficie d'une réactivité supérieure à 0.85 alors que cette même proportion bénéficie d'une réactivité supérieure à 0.95 dans le cas RL-ESN. A l'opposé de ces résultats en faveur de l'algorithme RL-ESN, les 10% de jobs batchs obtenant les plus mauvaises performances par cet algorithme connaissent un gain situé dans l'intervalle $[0, 0.3]$. Ce même sous-ensemble de job obtient un gain situé dans l'intervalle $[0, 0.4]$ dans le cas des ordonnancement RL-NN et FIFO. Ce comportement ne se remarque pas dans le cas des jobs interactifs où les gains de l'ordonnancement RL-ESN vis à vis de FIFO sont présents sur toute la trace utilisée.

Ce résultat montre que l'algorithme RL-ESN semble effectuer une rétention des jobs batchs en queue au profit de jobs ayant une espérance de gain plus élevée, les jobs interactifs notamment. Ce comportement est une explication de la manière dont l'algorithme obtient les gains observés et fait référence au principe de rétention énoncé dans la section 5.2.4. Dans les expériences suivantes, la proportion croissante des jobs interactifs dans une charge de même intensité devrait accentuer cette tendance.

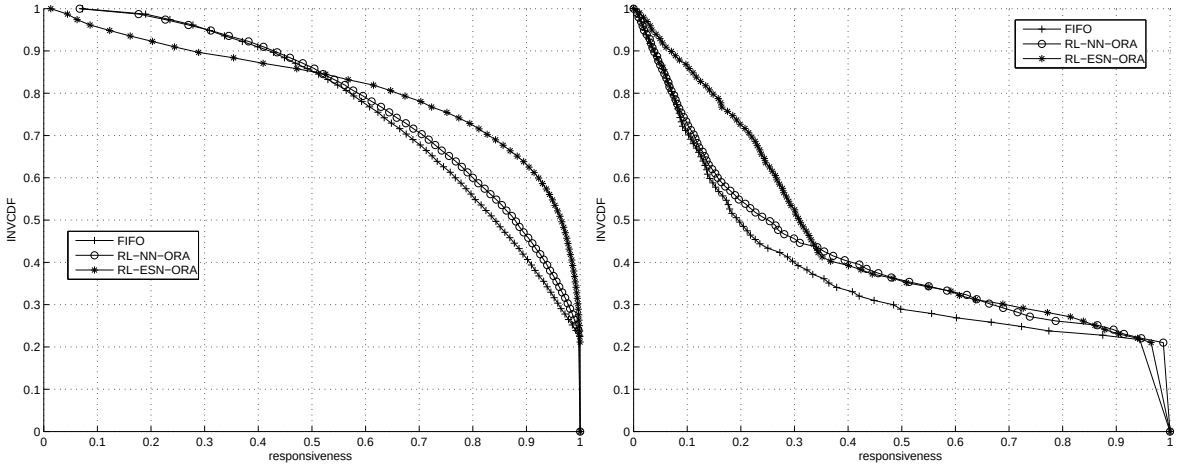

FIG. 7.2 – Distribution cumulative inverse de la réactivité des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-2.

7.4.3 Charge PE-3

Les statistiques de réactivité pour l'expérience utilisant la charge PE-3 sont résumées dans la table 7.3. Dans ce troisième cas, la charge est maintenue à 0.99 avec la moitié des jobs appartenant à la catégorie des interactifs. Les performances pour les jobs batchs sont améliorées de 18% pour l'apprentissage par renforcement via l'algorithme RL-ESN en termes de médiane. L'amélioration est significative, 95%, dans le cas des jobs interactifs.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy	Std	Médiane	Moy	Std	Médiane
FIFO	0.79	0.17	0.80	0.50	0.32	0.43	0.64	0.30	0.68
RL-NN-ORA	0.80	0.18	0.85	0.53	0.33	0.47	0.66	0.30	0.72
RL-ESN-ORA	0.86	0.19	0.96	0.69	0.34	0.88	0.78	0.29	0.94

Tab. 7.3 – Réactivité, trace synthétique PE-3.

Concernant les performances totales calculées sur la totalité de la trace, la réactivité en moyenne est améliorée de 21% dans le cas de l'algorithme d'apprentissage employant un ESN et la médiane est améliorée de 38% par rapport à FIFO. Pour cette charge, les performances globales sont également améliorées, à la différence des deux expériences précédentes. Une explication à cette amélioration globale peut être la présence plus importante de jobs interactifs privilégiés par la fonction d'objectifs de l'ordonnanceur par apprentissage par renforcement. Rappelons que cette proportion de jobs interactifs se rapproche de celle observée dans les traces réelles de la grille EGEE.

Les distributions cumulatives inverses des valeurs de réactivité présentées dans la figure 7.3 illustrent l'amélioration significative obtenue pour les jobs interactifs par l'algorithme RL-ESN. Dans le cas de l'ordonnancement FIFO, 50% de ces jobs ont une réactivité supérieur à 0.4 alors que ces mêmes 50% ont une réactivité supérieure ou égale à 0.9 dans le cas RL-ESN. Un gain présent mais plus modéré figure dans le cas de l'algorithme RL-NN. Dans le cas d'un ordonnancement RL-ESN, 90% des jobs batchs ont une réactivité supérieure à 0.2, alors que ces mêmes 90% de jobs batchs ont une valeur de réactivité deux fois supérieure dans le cas d'un ordonnancement FIFO. Néanmoins, dans le cas FIFO, le moitié de ces jobs batchs ont une réactivité supérieure ou égale à 0.8 alors que dans le cas RL-ESN, cette même proportion de jobs bénéficie d'une réactivité de 0.95. Une partie des jobs batchs subissent donc un temps d'attente en queue plus important dans le cas RL-ESN pénalisant leur gain en réactivité. Encore une fois, c'est par un prolongement de l'attente en queue de ce sous-ensemble des jobs batchs que l'ordonnanceur parvient à obtenir les gains observés dans la classe des jobs interactifs. Cependant, cette rétention n'affecte que partiellement les résultats obtenus pour les jobs batchs qui bénéficient, malgré

tout, globalement de meilleures performances que FIFO.

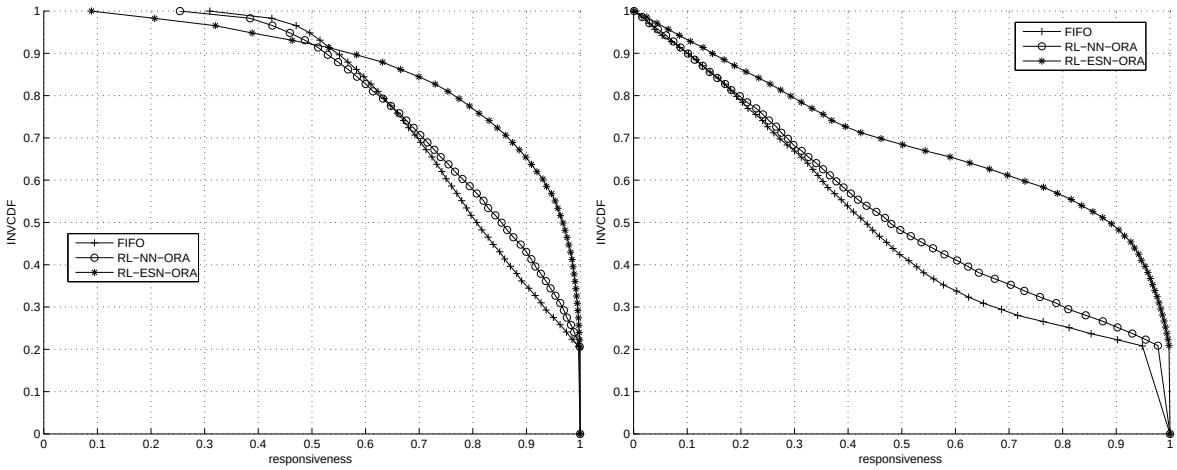

FIG. 7.3 – Distribution cumulative inverse de la réactivité des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-3.

7.4.4 Charge PE-4

Les statistiques de réactivité pour l'expérience utilisant la distibution PE-4 sont résumées dans la table 7.4. Dans ce dernier cas de configuration Poisson/exponentielle, le coefficient d'utilisation ρ est maintenu à 0.99 avec 90% des jobs appartenant à la catégorie des interactifs. Le gain en réactivité reste marqué dans le cas des jobs interactifs et batchs, l'apport est essentiellement prononcé dans le cas des jobs interactifs où la réactivité est améliorée de 29%. Dans le cas des jobs batchs, les performances des trois algorithmes sont quasiment équivalentes malgré une amélioration de 6% dans le cas de l'ordonnancement par apprentissage par renforcement utilisant un ESN. Cependant, l'apport vis à vis de FIFO est moins marqué que dans le cas PE-3, du fait certainement de la trop faible quantité de jobs batchs sur laquelle faire levier afin de favoriser les jobs interactifs, comme expliqué dans le paragraphe précédent. Concernant les performances totales, la réactivité en moyenne est améliorée de 18% dans le cas de l'algorithme d'apprentissage employant un ESN et la médiane est améliorée de 25% par rapport à FIFO. Pour cette charge, les performances globales sont améliorées de manière sensiblement équivalente, quoique légèrement inférieure à la charge PE-3.

Les distributions présentées dans la figure 7.4 confirment les remarques précédentes. Pour l'algorithme RL-ESN, 2% des jobs batchs ont une valeur de réactivité comprise entre 0.2 et 0.6 alors que, pour l'ensemble des jobs batchs, cette valeur est toujours supérieure à 0.6 dans les cas des ordonnancements FIFO et RL-NN. En contre-partie, 90% des jobs batchs ont une réactivité supérieure à 0.95 alors que 60% des jobs batchs atteignent ce

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.90	0.09	0.93	0.66	0.30	0.72	0.69	0.29	0.76
RL-NN-ORA	0.91	0.10	0.95	0.68	0.30	0.77	0.71	0.29	0.81
RL-ESN-ORA	0.96	0.10	0.99	0.81	0.25	0.93	0.82	0.24	0.95

Tab. 7.4 – Réactivité, trace synthétique PE-4.

résultat dans le cas du l'ordonnancement FIFO. Ce phénomène semble équivalent à celui décrit pour la charge PE-3, dans l'algorithme RL-ESN, une partie des jobs batchs subit une perte de réactivité vis à vis des algorithme RL-NN et FIFO, au profit de tous les autres jobs qui obtiennent une valeur de réactivité supérieure.

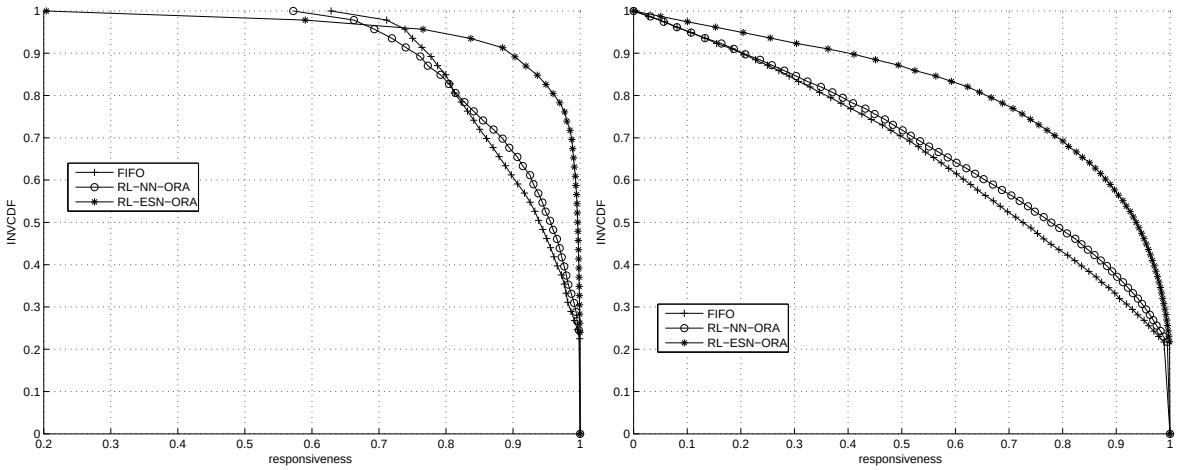

Fig. 7.4 – Distribution cumulative inverse de la réactivité des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-4.

7.4.5 Charge MMPP-1

Les statistiques de réactivité pour l'expérience utilisant la distibution MMPP-1 sont résumées dans la table 7.5. Une amélioration de 28% du traitement des jobs interactifs de l'algorithme RL-ESN est à observer. La réactivité des jobs batchs est multipliée par deux. Dans ce cas, le problème d'ordonnancement posé est plus difficile, les performances de l'algorithme obtenues par l'utilisation d'un réseau de neurones multicouche sont mauvaises, au même titre que FIFO, en revanche avec l'ESN, on obtient une amélioration considérable. Ce phénomène peut s'expliquer par les difficultés associées à l'ordonnancement de flot de demandes contenant des pics d'activités, modélisés grâce au MMPP. Lors de ces pics de charge, le nombre de jobs présents en queue est plus important, élargissant ainsi l'espace

de choix de job à ordonnancer pour les algorithmes d'ordonnancement RL. Une difficulté comparable se pose dans le cas de l'ordonnancement FIFO : lorsqu'un flot de jobs batchs suivi d'un flot de jobs interactifs sont placés en queue à une date donnée. Dans ce cas, les gains de réactivité des jobs interactifs seront plus faibles que si la même série de jobs se présente en queue dans le même ordre, mais à des dates distinctes du fait de l'attente prolongée en queue lié au traitement des jobs batchs.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.45	0.25	0.42	0.08	0.18	0.04	0.41	0.26	0.38
RL-NN-ORA	0.49	0.27	0.46	0.12	0.20	0.04	0.45	0.29	0.41
RL-ESN-ORA	0.70	0.33	0.87	0.36	0.29	0.27	0.66	0.34	0.84

Tab. 7.5 – Réactivité, charge synthétique MMPP-1.

L'apport de l'algorithme RL-ESN se maintient également à l'échelle de la distribution complète des jobs. Dans ce cas, l'algorithme RL-NN constitue un apport deux fois moins important que RL-ESN.

La figure 7.5 présente les distributions cumulatives de réactivité des jobs. Dans le cas de l'ordonnancement RL-ESN des jobs batchs, 15% de ceux-ci obtiennent une valeur réactivité inférieure aux valeurs de réactivité des jobs ordonnancés via FIFO. En revanche, 75% des jobs batchs ordonnancés par RL-ESN connaissent un gain supérieur aux deux autres algorithmes étudiés. Cette observation se rapporte encore au principe de rétention énoncé, une minorité des jobs les plus longs sont retenus en queue afin de favoriser les jobs interactifs constituant une plus grande espérance de gain. La distribution montre que l'amélioration est vrai pour tous les jobs interactifs.

7.4.6 Charge MMPP-2

La table 7.6 résume les statistiques de réactivité pour la charge MMPP-2. L'intérêt de cette expérience réside dans le traitement défavorable des jobs interactifs observé dans les performances obtenues par l'algorithme FIFO. Bien que les jobs batchs connaissent des valeurs de réactivité très satisfaisantes, une perte de 22% en moyenne de cette valeur est observée pour les jobs interactifs avec cet ordonnancement. L'effet de pic est moins prononcé dans cette seconde distribution du fait de la présence importante des jobs interactifs, 50%, alors que cette proportion était de 10% dans le cas MMPP-1. L'algorithme RL-ESN améliore de 11% les performances des jobs interactifs sans dégrader les gains des jobs batchs.

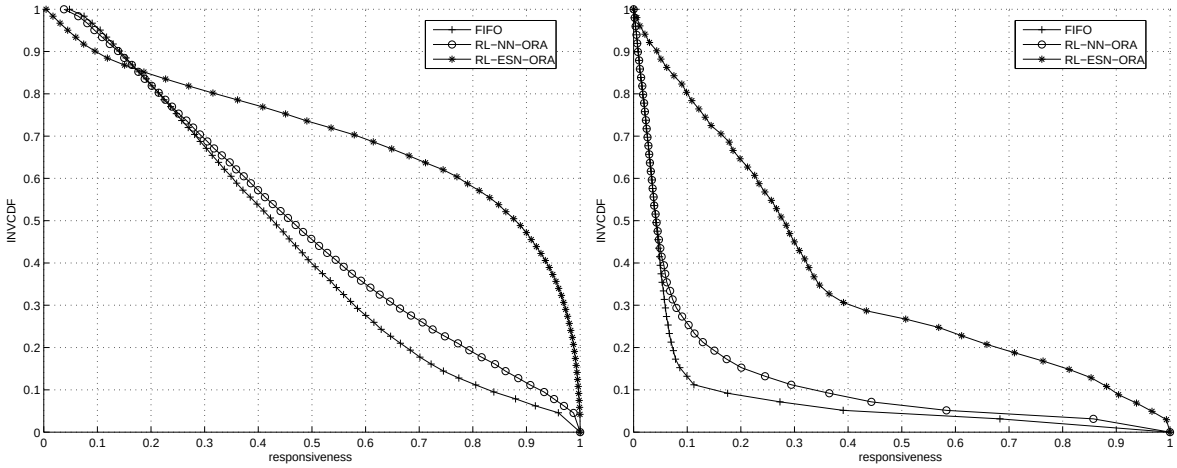

FIG. 7.5 – Distribution cumulative inverse de la réactivité des jobs batchs (gauche) et interactifs (droite), trace synthétique MMPP-1.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.94	0.07	0.98	0.77	0.26	0.88	0.85	0.21	0.96
RL-NN-ORA	0.94	0.08	0.98	0.78	0.26	0.92	0.86	0.21	0.97
RL-ESN-ORA	0.95	0.09	0.99	0.86	0.23	0.99	0.91	0.18	0.99

TAB. 7.6 – Réactivité, charge synthétique MMPP-2.

Les résultats obtenus pour les jobs interactifs diminuent les performances globales de l'algorithme FIFO. Par ailleurs, pour cette charge comme pour celles présentées précédemment, les ordres de grandeur des valeurs de variance sont équivalents.

Les distributions cumulatives inverses de réactivité pour la charge MMPP-2 sont présentées en figure 7.6. L'effet de rétention n'est pas observé ici, les jobs batchs et interactifs connaissent un gain de réactivité supérieur dans le cas RL-ESN. Le gain apporté par l'algorithme RL-NN est marginal en comparaison à celui observé via l'algorithme RL-ESN.

Plusieurs conclusions sont à tirer de ces premiers résultats. Tout d'abord, les modèles par apprentissage par renforcement améliorent les performances dans le cas des modèles de configuration Poisson/exponentielle et MMPP. Les améliorations de performances globales sont les plus marquées dans le cas des charges PE-3 et PE-4, caractérisées par une plus forte présence de jobs interactifs. Cet apport n'est plus aussi important lorsque la distribution des temps d'inter-arrivées est transitoire, tel que dans le cas MMPP-2.

Concernant les performances comparées des algorithmes d'apprentissage, le modèle ESN offre, en tout point, des performances supérieures aux résultats obtenus avec un

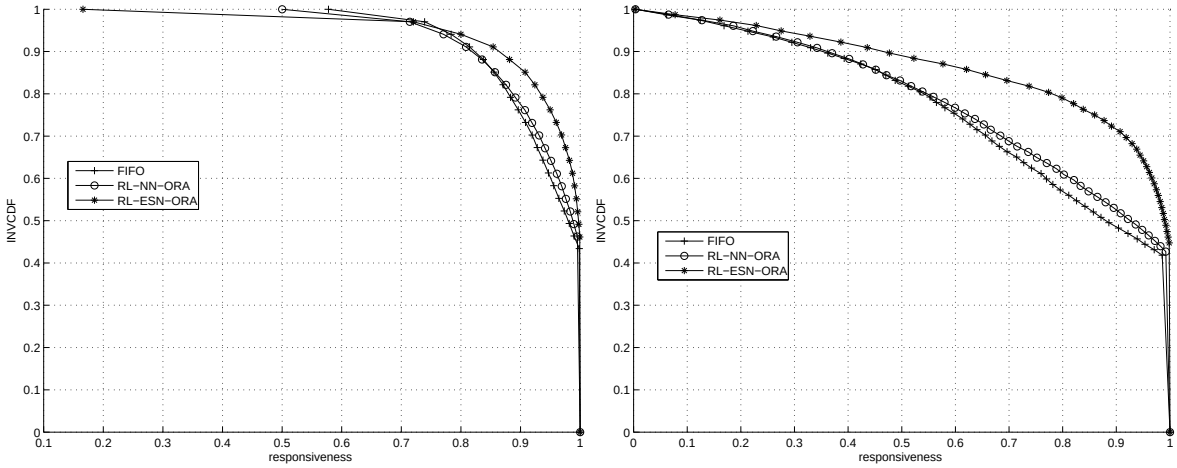

FIG. 7.6 – Distribution cumulative inverse de la réactivité des jobs batchs (gauche) et interactifs (droite), trace synthétique MMPP-2.

réseau de neurones multi-couches standard. Il est important de mettre en perspective de ce résultat le coût moindre de la procédure d'apprentissage d'un ESN vis à vis de celle d'un réseau de neurones multi-couches. En effet, l'ESN ne nécessite qu'une régression linéaire sur ses poids de sortie à la différence d'une descente de gradient stochastique souvent coûteuse en temps de calcul pour les réseaux de neurones multi-couches.

Les résultats précédents semblent montrer qu'un système par apprentissage par renforcement, utilisant comme fonction d'objectif la réactivité, est capable d'améliorer la prise en charge des jobs interactifs au sein d'une architecture de site de grille. L'apport d'un modèle à mémoire dans ce contexte a également été mis en évidence par l'amélioration systématique des résultats vis à vis d'un réseau de neurone multicouche standard, ne comportant aucun mécanisme de mémoire.

7.5 Cas synthétique : Ordonnancement faisable, étude des délais d'attente

L'utilisation du délai d'attente en queue des jobs est un indicateur concret aux yeux de l'utilisateur final de la performance du système d'ordonnancement. Bien que l'utilisation de la réactivité soit un indicateur suffisant de la qualité des performances des ordonnanceurs par apprentissage par renforcement, l'étude des délais d'attente permet de constater l'apport de notre approche, aux yeux de l'utilisateur, dans le cas des jobs de courte durée d'exécution. Pour ces jobs particuliers, le temps passé en queue est une variable critique dans l'estimation de la qualité de l'ordonnancement adopté. Au cours de l'interprétation des résultats, l'importance de l'utilisation de la médiane est à mentionner, justifiée par

la présence de valeurs extrêmes et d'une grande variance de la mesure rendant difficile l'analyse des résultats par le biais de l'espérance seule.

7.5.1 Charge PE-1

Les statistiques des délais d'attente sont résumées dans cette section. Dans les cinq traces synthétiques considérées, les résultats sont consistants avec les observations effectuées pour les valeurs de réactivité. Les délais d'attente sont diminués dans le cas des ordonnanceurs par apprentissage par renforcement. Une amélioration plus importante est à observer dans le cas des jobs interactifs ainsi que dans le cas du modèle ESN vis à vis du modèle de réseau de neurone multicouche standard.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	2126	2566	1270	2112	2545	1341	2124	2564	1280
RL-NN-ORA	2133	3010	807	2053	2859	859	2125	2996	811
RL-ESN-ORA	2506	6015	235	1768	3798	276	2438	5850	238

Tab. 7.7 – Délais d'attente, charge synthétique PE-1. Tous les temps en secondes.

La figure 7.7 présente les distributions cumulatives des délais d'attente pour les jobs batchs et interactifs. Dans ce type de représentation, la maximisation de la valeur étudiée correspond au comportement souhaité (*Larger is Better*). Pour cette premiere configuration, si on considére un délai d'attente de 5 minutes, l'ordonnancement FIFO le permet pour 50% des jobs interactifs, l'ordonnancement NN le permet pour 60% et pour 80% dans le cas ESN. On note une fois de plus la supériorité de la méthode. Concernant les jobs batchs, les performances sont quasi équivalentes. Cependant, dans le cas RL-ESN, 5% des jobs attendent plus que les jobs ordonnancés par FIFO. Ce résultat est consistant avec le principe de rétention énoncé précedemment, c'est en retenant plus longtemps des jobs batchs en file d'attente que l'ordonnanceur parvient à améliorer les performances des jobs interactifs.

FIXME : Fixer à deux minutes les comparaisons de CDF systématiquement, avec un mot dans l'intro.

7.5.2 Charge PE-2

La figure 7.8 présente des distributions cumulatives des délais d'attente pour les jobs batchs et interactifs pour la distribution PE-2. Dans ce second cas, le phénomène observé

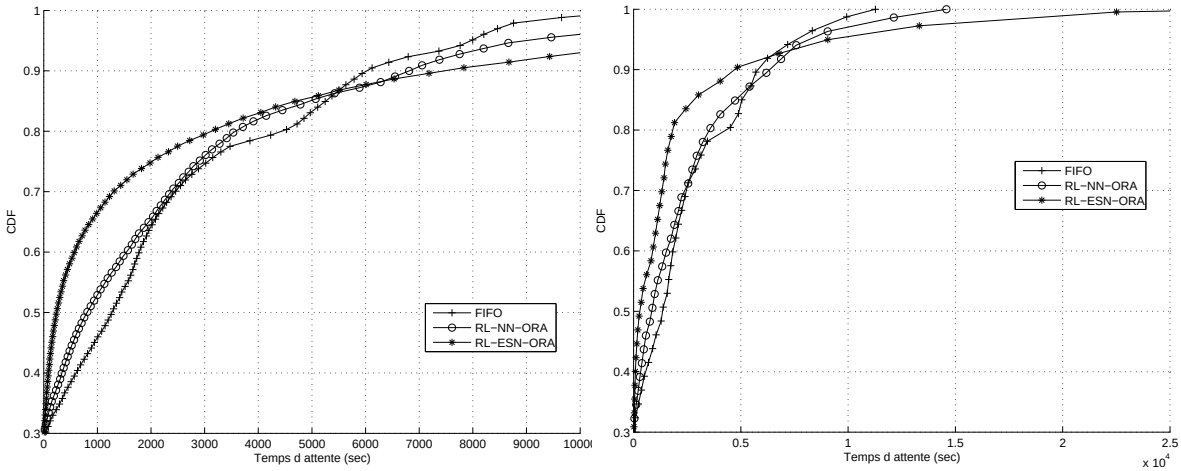

FIG. 7.7 – Distribution cumulative des délais d'attente des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-1.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	2241	2367	1410	2325	2428	1602	2249	2373	1425
RL-NN-ORA	2203	2756	1079	2365	2951	1063	2219	2777	1079
RL-ESN-ORA	3635	9401	281	1590	3281	899	3423	8983	313

TAB. 7.8 – Délais d'attente, charge synthétique PE2. Tous les temps en secondes.

dans la distribution PE-1 est visible. Alors que 50% des jobs interactifs n'attendant pas plus de 4 minutes, c'est 90% de ces mêmes jobs qui ne franchissent pas ce seuil d'attente dans le cas de l'ordonnancement RL-ESN. Encore une fois, l'apport, bien que notable, est très inférieur dans le cas de l'ordonnancement RL-NN. En contre partie, 15% des jobs batchs connaissent des délais d'attente plus longs dans le cas de l'ordonnancement RL-ESN. En comparaison, les résultats obtenus par FIFO et RL-NN pour les jobs batchs sont quasi-similaires.

7.5.3 Charge PE-3

La figure 7.9 présente les distributions cumulatives des délais d'attente des jobs batchs et interactifs pour la distribution PE-3. Les remarques générales effectuées pour les deux charges précédentes restent valables. Les résultats sont encore une fois satisfaisants, 75% des jobs interactifs attendent moins de 4 minutes en file d'attente pour l'ordonnancement RL-ESN, contre 30% dans le cas FIFO et 40% dans le cas RL-NN.

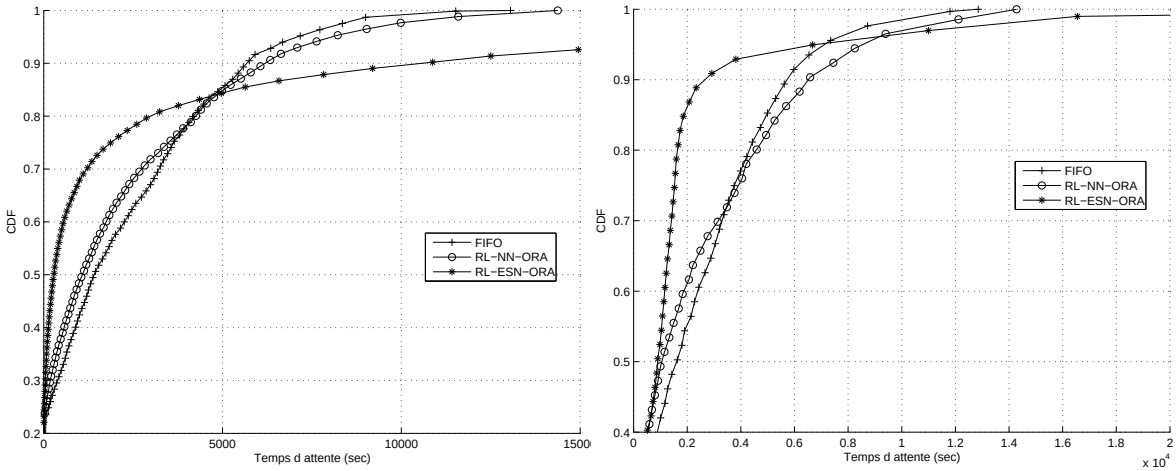

FIG. 7.8 – Distributions cumulatives des délais d'attente des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-2.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	569	500	522	586	502	542	578	501	535
RL-NN-ORA	568	608	356	577	603	394	573	606	378
RL-ESN-ORA	568	1393	60	467	1079	36	516	1243	46

TAB. 7.9 – Délais d'attente, charge synthétique PE-3. Tous les temps en secondes.

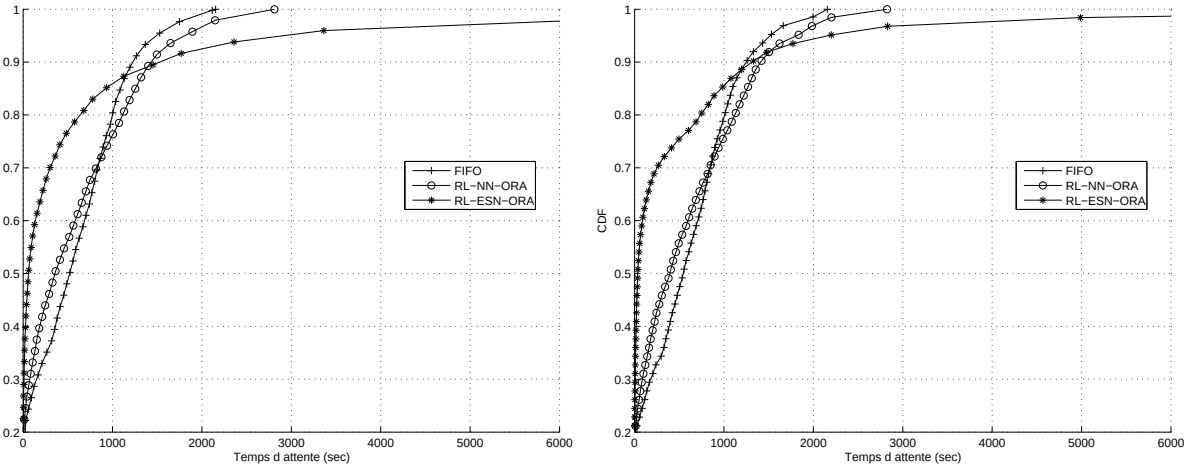

FIG. 7.9 – Distributions cumulatives des délais d'attente des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-3.

7.5.4 Charge PE-4

La figure 7.10 présente les distributions cumulatives des délais d'attente des jobs batchs et interactifs pour la distribution PE-4. Dans ce dernier cas, le gain est toujours significa-

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	134	140	74	148	158	82	147	156	81
RL-NN-ORA	133	165	58	147	184	60	145	182	60
RL-ESN-ORA	83	337	4	126	379	11	122	375	10

Tab. 7.10 – Délais d'attente, charge synthétique PE-4. Tous les temps en secondes.

tif bien que moins important que dans l'expérience utilisant la distribution PE-3. Ainsi, 60% des jobs interactifs ordonnancés par FIFO attendent moins de 4 minutes contre pour l'algorithme RL-ESN. En outre, seulement 2% des jobs batchs connaissent des délais d'attente plus longs dans le cas de l'ordonnancement RL-ESN, en comparaison avec FIFO. Ce phénomène s'explique par la proportion de jobs batchs, ramenée ici à 10% dans cette trace synthétique, limitant donc la possibilité de rétention de l'algorithme, vis à vis des charges PE-1, PE-2 et PE-3.

7.5.5 Charge MMPP-1

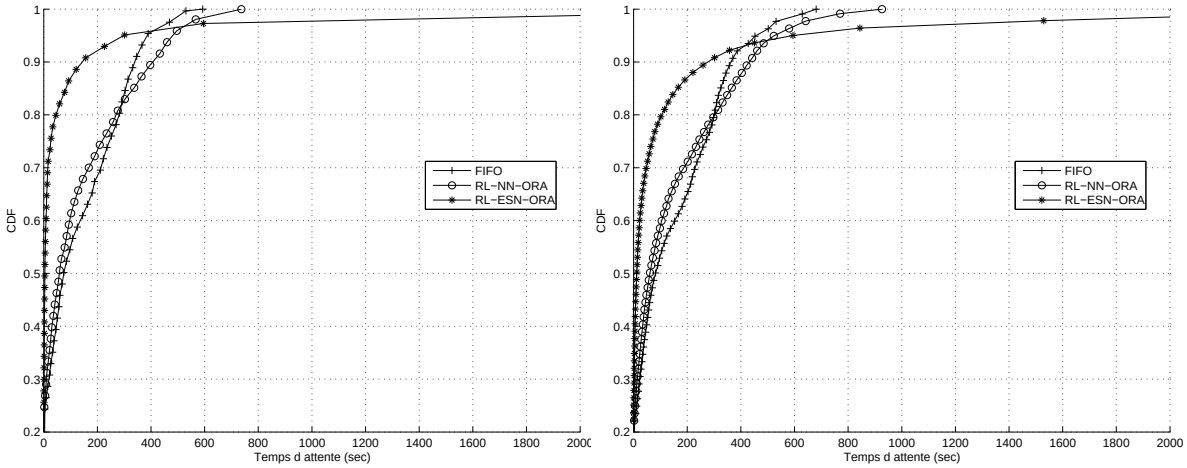

Fig. 7.10 – Distributions cumulatives des délais d'attente des jobs batchs (gauche) et interactifs (droite), trace synthétique PE-4.

La table 7.11 résume les statistiques de temps d'attente pour la charge MMPP-1. Les remarques faites dans le cas de la réactivité de cette charge sont confirmées. Dans le cas RL-ESN, la moyenne des durée d'attente des jobs batchs augmente de 30%, avec une augmentation de quasiment un ordre de grandeur de la variance de la valeur de ceux-ci. Ce résultat est à mettre en correspondance avec la division par dix de la médiane et de trois de la moyenne des délais d'attente des jobs interactifs. Enfin, malgré une augmentation de

30% de la variance dans le cas des jobs interactifs, l'amélioration constatée sur la médiane atteste du gain apporté par l'apprentissage par renforcement.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	10284	4875	11510	10510	4651	11586	10307	4852	11518
RL-NN-ORA	10321	7235	10035	10465	7178	10237	10336	7229	10055
RL-ESN-ORA	13621	33482	992	2989	7916	1145	12482	31913	1033

TAB. 7.11 – Délais d'attente, charge synthétique MMPP-1. Tous les temps en secondes.

Concernant les performances sur l'ensemble de la trace, l'écart des valeurs de médiane entre FIFO est RL-ESN est à noter, un ordre de grandeur, à la différence des valeurs moyennes qui restent, dans ce cas, comparables. Les performances obtenues par RL-NN sont comparables à celles observées via l'algorithme FIFO dans ce cas.

La figure 7.11 présente les distributions cumulatives des délais d'attente pour la charge MMPP-1. L'algorithme RL-ESN offre un délai d'attente inférieur aux deux autres algorithmes pour 80% des jobs batchs. Pour les jobs interactifs, c'est 95% de ceux ci qui connaissent un délai d'attente inférieur, toujours vis à vis de FIFO et RL-NN. A contrario, le principe de rétention est encore une fois vérifié car 20% des jobs batchs connaissent un délai d'attente supérieur aux délais d'attente connus avec les autre algorithmes, ceci au profit des jobs interactifs.

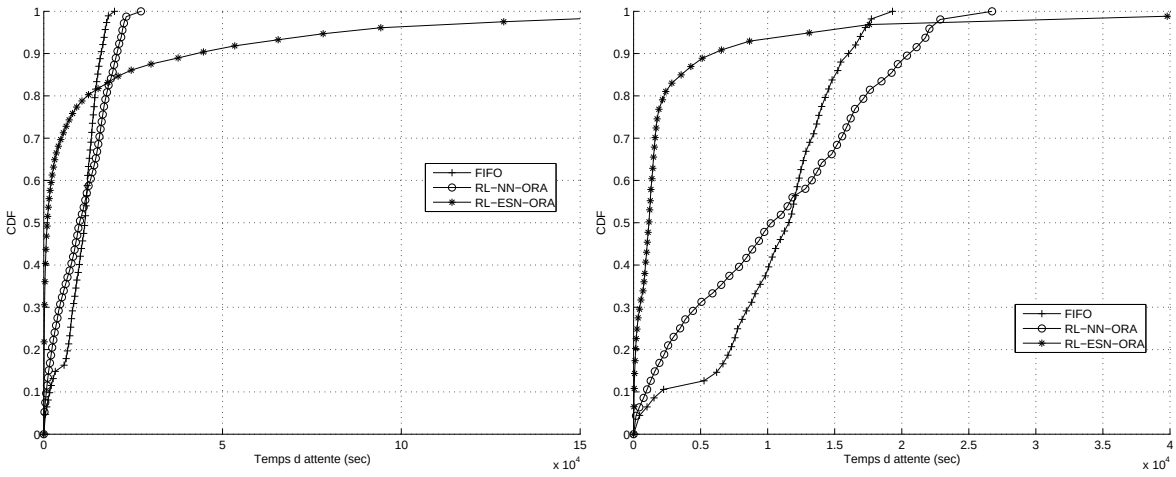

FIG. 7.11 – Distributions cumulatives des délais d'attente des jobs batchs (gauche) et interactifs (droite), trace synthétique MMPP-1.

7.5.6 Charge MMPP-2

La table 7.24 détaillent les statistiques de temps d'attente pour la charge MMPP-2. En correspondance avec les remarques de réactivité effectuées sur cette charge, ce sont les jobs interactifs qui bénéficie majoritairement de l'amélioration de leur durée d'attente.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	122	158	45	126	157	50	124	158	48
RL-NN-ORA	122	184	24	126	184	28	124	184	27
RL-ESN-ORA	123	398	5	91	311	4	107	357	4

TAB. 7.12 – Délais d'attente, charge synthétique MMPP-2. Tous les temps en secondes.

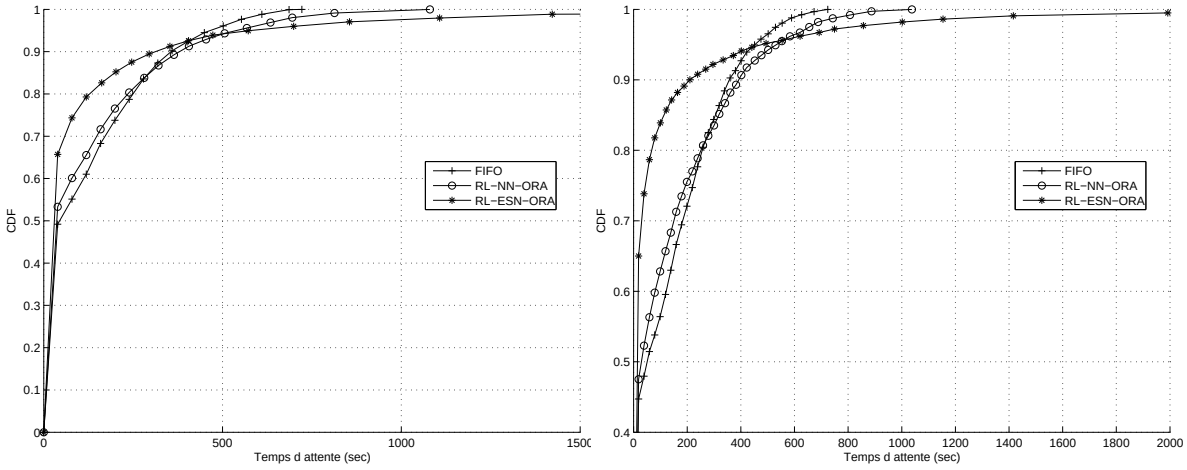

FIG. 7.12 – Distributions cumulatives des délais d'attente des jobs batchs (gauche) et interactifs (droite), trace synthétique MMPP-2.

La première constatation à effectuer, sur ce dernier résultat, porte sur la différence importante en terme de délai d'attente en comparaison avec les modèles de Poisson et exponentiels. La présence de pics d'activité pose un problème plus difficile et diminue considérablement les performances en terme de qualité de service pour les 3 algorithmes considérés, autant pour les jobs batchs que les jobs interactifs. Cependant, le temps d'attente moyen des jobs interactifs est réduit de plus de moitié pour les modèles d'apprentissage par renforcement utilisant un ESN. Néanmoins, les délais d'attente pour les jobs batchs subissent une hausse de 10%. Cette inflexion reste marginale en considération de la diminution d'un ordre de grandeur du délai d'attente des jobs interactifs.

7.6 Cas synthétique : Ordonnancement faisable, étude de l'équité

Dans cette partie, on s'interesse avant tout à la dynamique de la mesure d'équité tout au long des simulations car celle-ci est faisable en fin d'expérience. Les résultats présentés dans les figures 7.13, 7.14, 7.15, 7.16, 7.17 et 7.18 comparent les performances des algorithmes d'apprentissage par renforcement RL-NN et RL-ESN aux performances obtenues avec l'algorithme FIFO. L'objectif d'équité est à considérer comme une contrainte à respecter plutôt qu'un objectif à maximiser, en comparaison avec l'analyse faite sur les performances de réactivité.

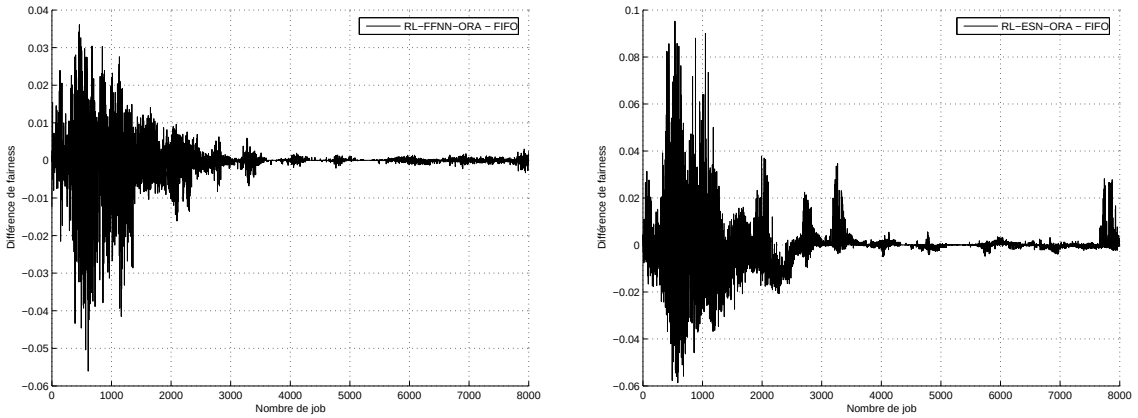

FIG. 7.13 – Différence d'équité pour la distribution PE-1

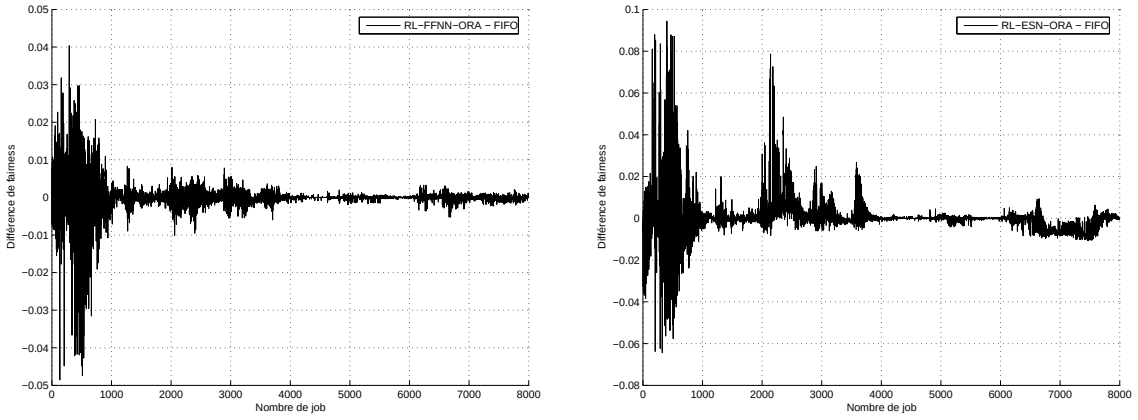

FIG. 7.14 – Différence d'équité pour la distribution PE-2

Sur les six couples de figures présentés, l'axe horizontal représente la série des jobs ordonnés par date de fin d'exécution, et l'axe vertical est la différence de mesure d'équité avec FIFO suivant la formule de gain présentée dans la section 5.2.3. Les 1000 premiers et derniers jobs ne sont pas utilisés pour l'analyse statistique car les phases de remplissage et de vidage du système produisent des échantillons aberrants. Ainsi, avec un ordonnan-

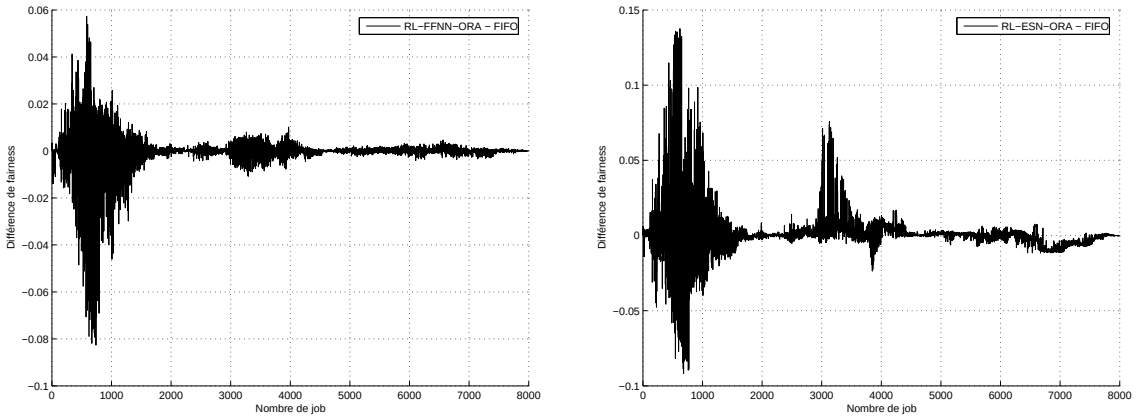

FIG. 7.15 – Différence d'équité pour la distribution PE-3

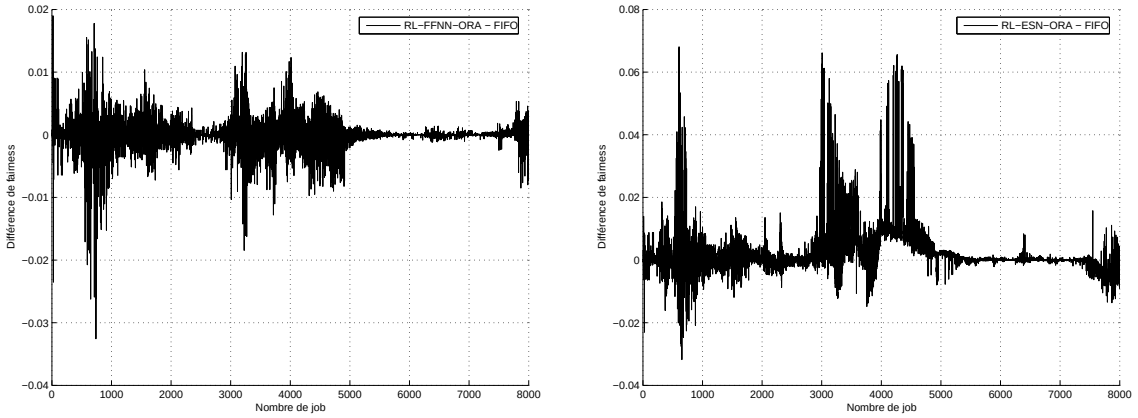

FIG. 7.16 – Différence d'équité pour la distribution PE-4

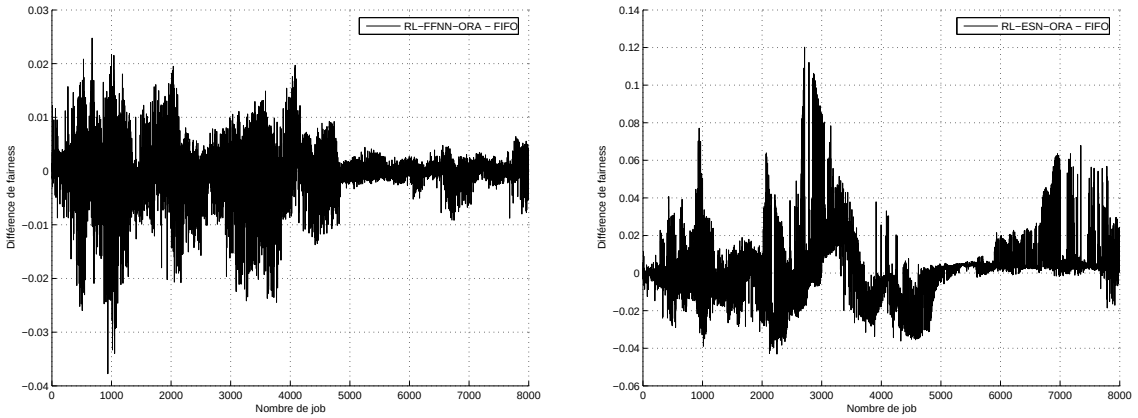

FIG. 7.17 – Différence d'équité pour la distribution MMPP-1

cement faisable, sur le long terme, l'échantillon de jobs est comparable aux performances du équité de FIFO, dans les cinq cas considérés. En considérant les ordres de grandeur des différences avec l'équité de référence, ces résultats montrent que les algorithmes d'appren-

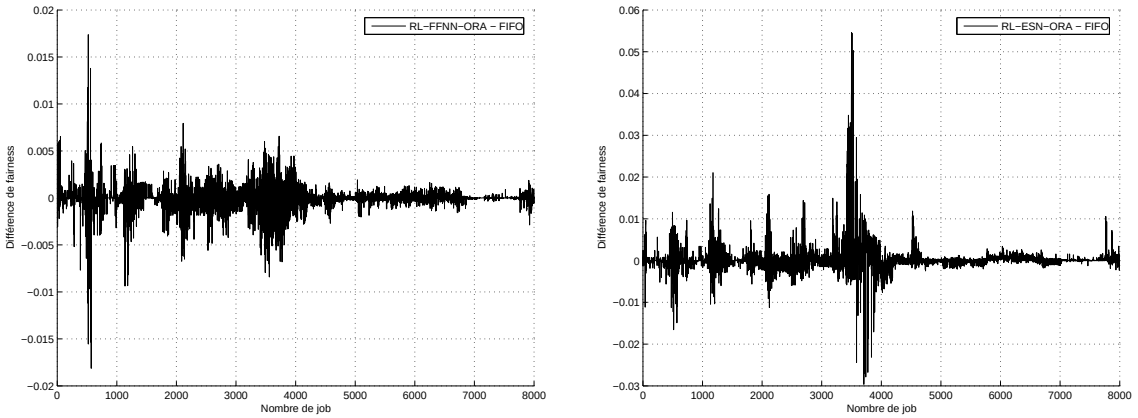

FIG. 7.18 – Différence d'équité pour la distribution MMPP-2

tissage par renforcement sont en mesure de suivre une politique d'équité tout en optimisant un objectif de réactivité. En effet, dans tous les cas, la différence d'équité est très faible, de l'ordre du centième, avec des excursions de l'ordre du dixième. Il est à noter que les algorithmes d'apprentissage semblent converger plus rapidement dans le cas des distributions Poisson/exponentielle. On notera enfin que la sous optimalité des résultats d'équité sont le prix à payer des performances obtenues en termes de réactivité, bien que ces résultats restent satisfaisants.

7.7 Cas synthétique : Ordonnancement infaisable

Les objectifs de haut niveau définis par les différents acteurs de la grille peuvent être irréalisables. Ce phénomène est fréquent dans le cas des objectifs d'équité stipulés par les administrateurs systèmes des sites de grille. Les poids définissant les objectifs d'équité décrivent l'activité des utilisateurs telle que l'espèrent ces administrateurs, ce qui peut être significativement différent des proportions réelles de jobs arrivant sur le site. Cette section assure que l'apprentissage par renforcement est capable d'intégrer l'équité comme fonction d'objectif dans le cas où la requête d'équité ne correspond pas à la distribution effective des organisations virtuelles dans la trace utilisée. En d'autres termes, on étudie la robutesse de l'algorithme.

Dans ce but, on considère le cas où la cible d'équité est $(0.3, 0.3, 0.2, 0.2)$, alors que la proportion réelle reste $(0.53, 0.14, 0.17, 0.16)$. Cet ordonnancement est infaisable car la première organisation virtuelle ne fournit pas assez de charge alors que la troisième et la quatrième demandent plus de temps de service qu'attribué dans les poids stipulés comme objectifs d'équité. Les traces d'activités synthétiques utilisées dans cette série d'expériences sont identiques à celles des expériences précédentes mais les objectifs d'équité ont été mo-

difiés. Elles conserveront donc des dénominations identiques aux expériences précédentes.

7.7.1 Réactivité

Les statistiques de réactivité pour l'ensemble des expériences effectuées sont résumées dans les tables 7.13, 7.14, 7.15, 7.16, 7.17 et 7.18 pour les traces d'activités synthétiques considérées.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.79	0.23	0.87	0.45	0.39	0.27
RL-NN-ORA	0.80	0.23	0.90	0.48	0.39	0.32
RL-ESN-ORA	0.84	0.24	0.96	0.60	0.38	0.74

Tab. 7.13 – Réactivité, trace synthétique PE-1, ordonnancement infaisable.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.77	0.22	0.84	0.38	0.37	0.19
RL-NN-ORA	0.78	0.22	0.87	0.41	0.37	0.25
RL-ESN-ORA	0.84	0.23	0.96	0.60	0.35	0.71

Tab. 7.14 – Réactivité, trace synthétique PE-2, ordonnancement infaisable.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.79	0.17	0.80	0.50	0.32	0.43
RL-NN-ORA	0.80	0.18	0.85	0.53	0.33	0.48
RL-ESN-ORA	0.86	0.21	0.97	0.61	0.35	0.66

Tab. 7.15 – Réactivité, trace synthétique PE-3, ordonnancement infaisable.

Pour tous les cas étudiés, les conclusions générales restent valables en matière de performance et de compromis entre la gestion des jobs batchs et interactifs. Les valeurs de réactivité sont quasiment équivalentes, les algorithmes d'apprentissage par renforcement semblent robustes aux variations éventuelles d'équité.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.90	0.09	0.93	0.66	0.30	0.72
RL-NN-ORA	0.91	0.09	0.95	0.68	0.30	0.77
RL-ESN-ORA	0.91	0.15	0.99	0.81	0.27	0.95

Tab. 7.16 – Réactivité, trace synthétique PE-4, ordonnancement infaisable.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.45	0.25	0.42	0.08	0.18	0.04
RL-NN-ORA	0.49	0.27	0.46	0.12	0.20	0.04
RL-ESN-ORA	0.64	0.36	0.80	0.31	0.28	0.24

Tab. 7.17 – Réactivité, charge synthétique MMPP-1, ordonnancement infaisable.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	0.94	0.07	0.98	0.77	0.26	0.88
RL-NN-ORA	0.94	0.08	0.98	0.78	0.26	0.92
RL-ESN-ORA	0.96	0.10	0.99	0.76	0.30	0.95

Tab. 7.18 – Réactivité, charge synthétique MMPP-2, ordonnancement infaisable.

7.7.2 Délais d'attente

Les statistiques des délais d'attente sont résumés dans les tables 7.19, 7.20, 7.21, 7.22, 7.23 et 7.24. Les résultats sont consistants avec les conclusions effectuées sur l'indicateur de réactivité.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	2126	2566	1270	2112	2545	1341
RL-NN-ORA	2111	2966	843	2087	3031	811
RL-ESN-ORA	2323	5278	218	1554	3478	151

Tab. 7.19 – Délais d'attente, charge synthétique PE-1, ordonnancement infaisable. Tous les temps en secondes.

Pour cette seconde configuration d'équité, les performances en terme de délais d'at-

Algorithme	Batch			Inter		
	Moy.	Std	Mediane	Moy.	Std	Mediane
FIFO	2241	2367	1410	2325	2428	1602
RL-NN-ORA	2201	2729	1095	2280	2823	1095
RL-ESN-ORA	2359	6045	232	1221	3145	155

Tab. 7.20 – Délais d'attente, charge synthétique PE-2, ordonnancement infaisable. Tous les temps en secondes.

Algorithme	Batch			Inter		
FIFO	569	500	522	586	502	542
RL-NN-ORA	574	617	370	581	615	393
RL-ESN-ORA	614	1492	56	656	1275	133

Tab. 7.21 – Délais d'attente, charge synthétique PE-3, ordonnancement infaisable. Tous les temps en secondes.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	134	140	74	148	158	82
RL-NN-ORA	131	162	59	148	186	61
RL-ESN-ORA	188	502	11	149	486	10

Tab. 7.22 – Délais d'attente, charge synthétique PE-4, ordonnancement infaisable. Tous les temps en secondes.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	10284	4875	11510	10510	4651	11586
RL-NN-ORA	10321	7235	10035	10465	7178	10237
RL-ESN-ORA	18259	36625	1892	3057	5996	1260

Tab. 7.23 – Délais d'attente, charge synthétique MMPP-1, ordonnancement infaisable. Tous les temps en secondes.

tente ne sont pas altérées et suivent la même tendance que celles observées dans le cas d'une équité faisable. Une configuration d'objectif avec une pondération majoritaire de l'objectif d'équité aurait été nécessaire afin d'éventuellement observer des modifications importantes dans les performances de réactivité et de temps d'attente. Cependant, ce type

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
FIFO	122	158	45	126	157	50
RL-NN-ORA	122	184	25	129	193	31
RL-ESN-ORA	98	346	5	253	550	13

Tab. 7.24 – Délais d'attente, charge synthétique MMPP-2, ordonnancement infaisable. Tous les temps en secondes.

de formulation n'est pas réaliste en situation réelle d'ordonnancement.

7.7.3 Équité

Les figures 7.19, 7.20, 7.21, 7.22, 7.23 et 7.24 présentent les séries temporelles des différences d'équité entre l'ordonnancement FIFO et les ordonnancements RL-NN et RL-ESN dans ce contexte d'objectif d'équité infaisable. La présentation des résultats suit la même convention que dans le cas d'une équité faisable.

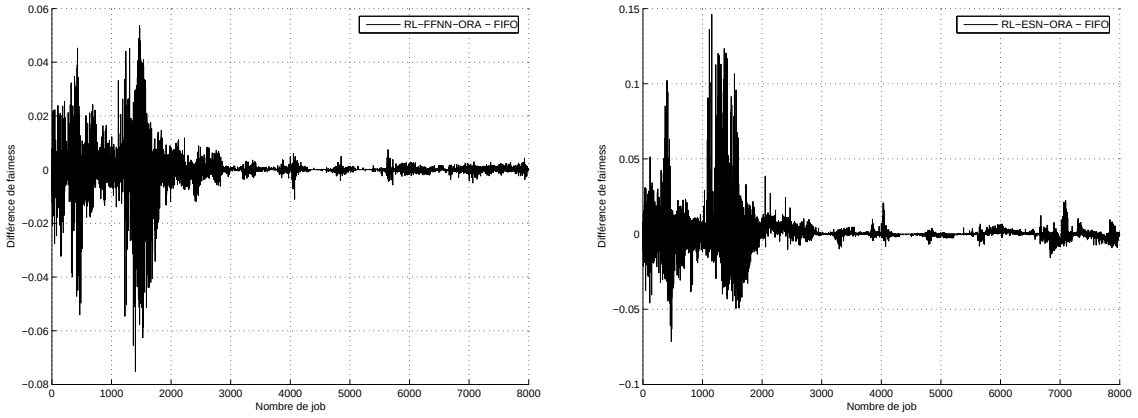

Fig. 7.19 – Différence d'équité pour la distribution PE-1

Dans tous les cas considérés, la différence avec l'ordonnancement natif reste marginale. Toutefois, une distance à l'ordonnancement natif supérieure de l'ordre de 30% en moyenne par rapport aux performances est obtenues dans le cas d'un ordonnancement faisable. Par ailleurs, les différences semblent toujours se situer dans des intervalles de temps localisés. Ce résultat permet de conclure encore une fois à une forme de robustesse de l'apprentissage par renforcement pour le problème de l'ordonnancement, indépendamment du modèle d'apprentissage utilisé, Echo State Network ou réseau de neurones multicouche. Bien que les valeurs de gain associées à l'objectif d'équité soient altérées, aucun impact significatif n'est constaté sur les performances de réactivité et délais d'attente.

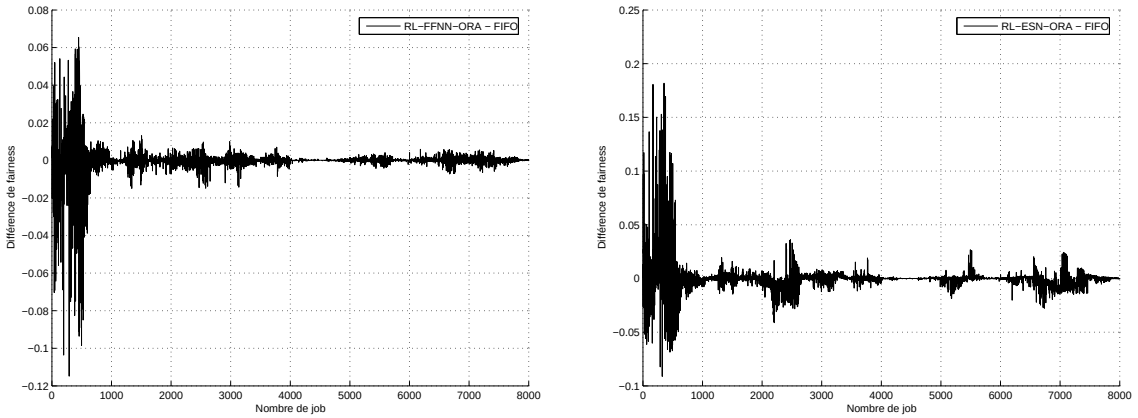

FIG. 7.20 – Différence d'équité pour la distribution PE-2

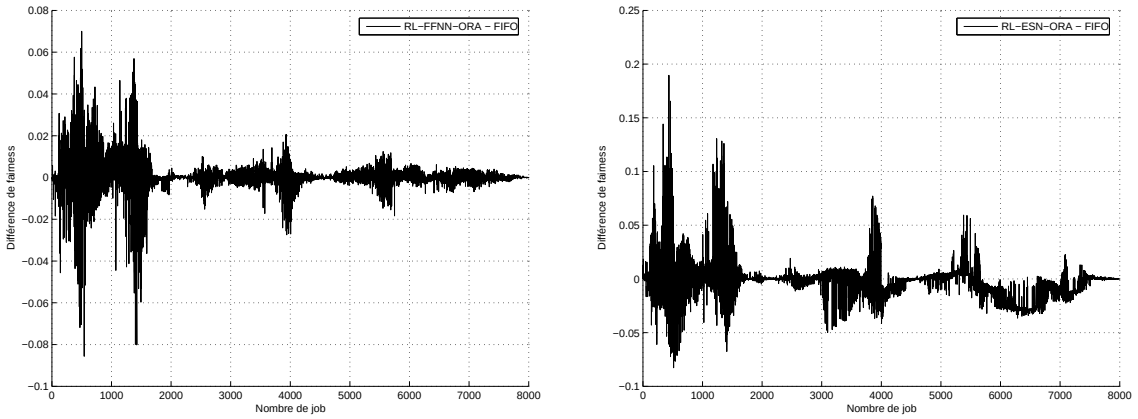

FIG. 7.21 – Différence d'équité pour la distribution PE-3

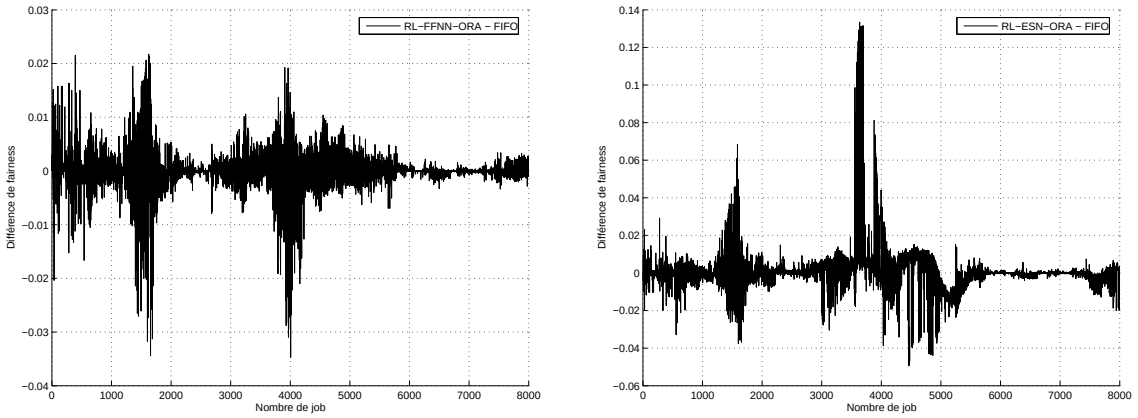

FIG. 7.22 – Différence d'équité pour la distribution PE-4

7.8 Cas synthétique : Discussion

Cette première série d'expériences met en lumière plusieurs aspects de la méthode d'ordonnancement proposée. Tout d'abord, les performances en terme de réactivité sont

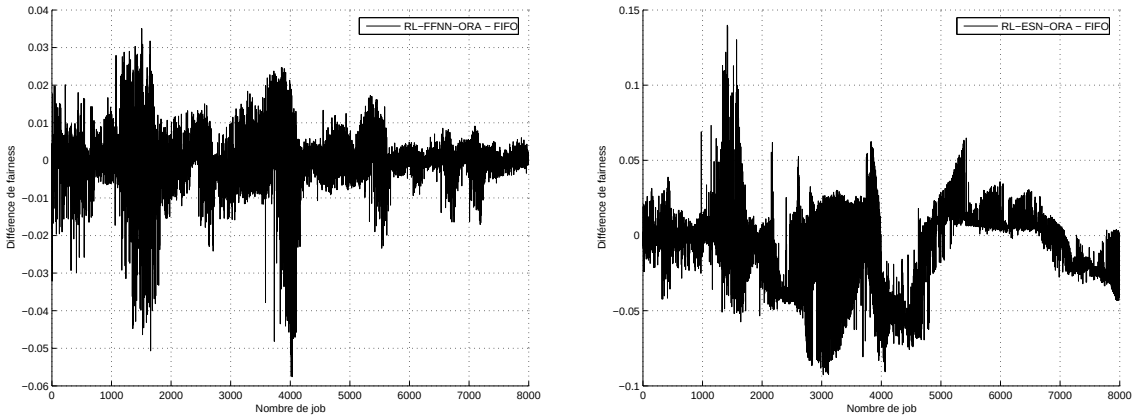

FIG. 7.23 – Différence d'équité pour la distribution MMPP-1

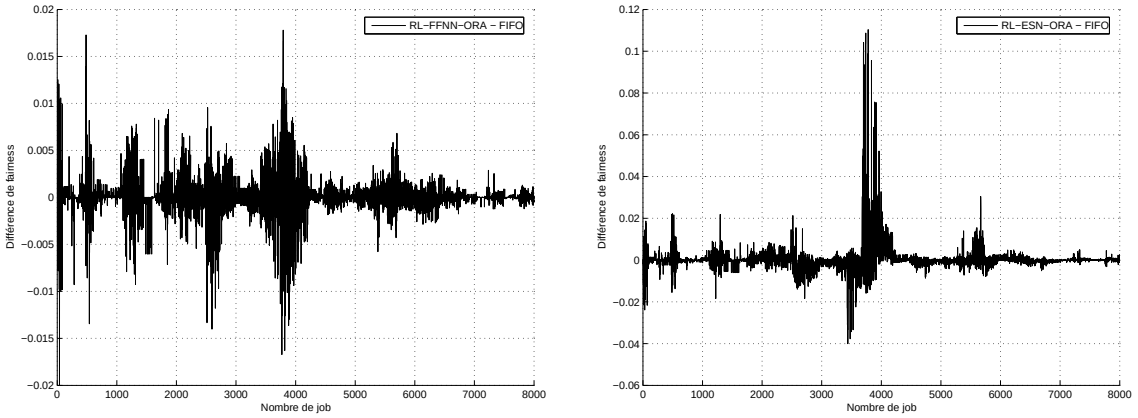

FIG. 7.24 – Différence d'équité pour la distribution MMPP-2

améliorées pour l'ensemble des jobs mais plus particulièrement dans le cas des jobs interactifs, comme souhaité. Suivant le principe de rétention (cf. section 5.2.4), le gain observé sur les jobs interactifs semble se baser sur une augmentation plus importante de la variance des délais d'attente des jobs batchs, au profit des délais d'attente des jobs interactifs. Malgré ce phénomène, les jobs batchs bénéficient d'une amélioration significative de la médiane de leur délai d'attente.

La réactivité semble constituer une fonction d'objectif efficace pour l'amélioration des performances de gestion des jobs à faible durée d'exécution dans les sites de grille. L'analyse des délais d'attente montre que les ordres de grandeur de ces gains sont notables pour les utilisateurs finaux du système administré. Ensuite, les résultats d'équité montrent, dans un premier temps, que l'utilisation d'une somme pondérée d'objectifs est une solution réaliste et efficace dans le cadre d'un apprentissage par renforcement multi objectif. Dans un second temps, les performances obtenues dans le cas d'une spécification d'un objectif d'équité infaisable assure de la robutesse d'un tel système dans ces conditions particulières fréquemment rencontrées dans les sites de grilles de calcul.

Concernant les modèles d'apprentissage utilisés, les résultats expérimentaux montrent clairement le gain apporté par un modèle ESN vis à vis d'un réseau de neurones multi-couche classique. Deux raisons peuvent expliquer ce résultat : 1) la simplicité algorithmique de l'apprentissage liée à la régression linéaire nécessaire à la paramétrisation de ce modèle 2) la capacité de mémoire du modèle. En effet, l'ensemble des variables utilisées par le MDP, bien que suffisant pour que l'apprentissage d'un modèle déterministe comme un réseau de neurones offre une politique de contrôle performante, ne peut être qualifié d'exhaustif au vue de la complexité de l'environnement et de l'action considérés. Si on admet que ces définitions de l'environnement et des actions sont incomplètes, nous sommes fondamentalement en présence d'un modèle partiellement observable (POMDP), où un modèle à mémoire est alors une solution efficace [SGL06] afin d'obtenir une politique de contrôle quasi-optimale, sachant que le modèle est défini dans un espace continu.

7.9 Cas réel : présentation

Une trace de 10000 jobs extraite du système de log de la plateforme d'ordonnancement locale d'EGEE, TORQUE-MAUI, décrite dans le chapitre précédent, a été utilisée pour cette seconde série d'expériences. Les conditions expérimentales suivantes ont été étudiées :

- **RIG-ORA** : La configuration des ressources est rigide, le nombre de machines P est fixé à 81, identique au nombre de machines identifiées dans la trace EGEE via leur identifiant y figurant. Dans ce contexte, le contrôleur d'ordonnancement de job est expérimenté seul. La durée d'exécution des jobs est supposée connue dès l'arrivée en file d'attente (e.g. modèles oracle). Dans ce cas de figure, $\lambda_s = 0.5$.
- **RIG-EST** : La configuration des ressources est rigide, comme dans le cas précédent, mais les durées d'exécution des jobs sont estimées par la médiane de leur catégorie respective, batch ou interactif.
- **ELA-ORA** et **ELA-EST** : L'allocation des ressources est désormais élastique. Le système d'allocation dynamique des ressources, dans le cas oracle et estimé, est analysé. Pour des raisons de simplicité d'interprétation des résultats, $\lambda_s = 1.0$, afin que le contrôle d'ordonnancement optimise uniquement la réactivité. Dans cette configuration, le poids de l'utilité réactivité λ_e varie de 0.5 (poids égal pour le gain de réactivité et d'utilisation pour le contrôleur de dimensionnement) à 1, où l'utilisation n'est alors pas considérée. Par exemple, l'expérience *ELA-ORA-0.5* correspond à $\lambda_e = 0.5$. Rappelons que la fonction de gain du processus de dimensionnement est

$$R_e = \lambda_e W + (1 - \lambda_e)U \tag{7.1}$$

avec W, la mesure de réactivité et U, la mesure d'utilisation. Ainsi, lorsque $\lambda_e = 1.0$,

le contrôleur ne maximisera que la réactivité de l'infrastructure. On devra alors remarquer un gain sur cet indicateur, en dépit du gain d'utilisation, car le système de décision aura tendance à choisir de maintenir en activité le nombre maximum de machines autorisé. A contrario, si $\lambda_e = 0.0$, le contrôleur aura alors tendance à sous dimensionner l'infrastructure de calcul, en ne maintenant actif qu'un nombre moins important de machines, afin de maximiser leur utilisation. Le cas médian, $\lambda_e = 0.5$, sera également étudié.

Après avoir étudié l'utilisation de l'apprentissage par renforcement dans le cas d'un cluster de calcul rigide, deux configurations élastiques seront étudiées. Dans le premier cas, l'objectif du contrôleur de dimensionnement sera de tenter d'améliorer les performances observées dans le cas rigide en autorisant ce dernier à allouer un nombre légèrement supérieur (de l'ordre de 10%) de machines vis à vis du cas rigide tout en autorisant ce dernier à reduire du même ordre de grandeur la taille de cluster de calcul suivant le compromis réactivité/utilisation. Dans le second cas, le contrôleur de dimensionnement sera uniquement en mesure de restreindre le nombre de machines en activité. On observera alors dans quelle mesure le contrôle sera capable d'améliorer la mesure d'utilisation du cluster tout en conservant des performances de réactivité exploitable. Pour toutes les configurations étudiées, les performances des méthodologies proposées sont comparées à celles de l'ordonnanceur natif présent sur le site d'EGEE, nommé (*EGEE*) dans la suite de ce chapitre. Ces performances natives, en terme de réactivité, temps d'attente, équité et utilisation sont calculées grâce aux informations présentes dans la trace d'activité.

7.10 Cas réel : ressources rigides

Le protocole d'analyse des résultats adopté dans cette partie est identique à celui utilisé pour les expériences synthétiques avec équité faisable. On s'attend essentiellement, dans ce contexte, à une amélioration de la qualité de service fournie aux jobs à faible durée d'exécution, avec un impact minime pour la réactivité associée aux jobs batchs.

7.10.1 Réactivité

La table 7.25 résume les performances de réactivité obtenues pour les différentes approches décrites dans la sous section précédente.

Dans le cas de l'ordonnanceur natif d'EGEE, les jobs interactifs connaissent un défaut de réactivité de 30% vis à vis des jobs batchs, comme remarqué dans les chapitres précédents. Concernant les performances des algorithmes d'apprentissage, la performance de réactivité

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
EGEE	0.82	0.26	0.99	0.62	0.41	0.96	0.73	0.36	0.98
RL-NN-ORA	0.93	0.18	1	0.94	0.22	1	0.93	0.20	1
RL-NN-EST	0.92	0.18	1	0.94	0.22	1	0.93	0.20	1
RL-ESN-ORA	0.93	0.19	1	0.94	0.20	1	0.93	0.19	1
RL-ESN-EST	0.93	0.17	1	0.95	0.19	1	0.94	0.18	1

TAB. 7.25 – Réactivité, charge EGEE

pour les jobs interactifs est améliorée de 53%, la classe des jobs batchs connait une amélioration plus faible de 13%.

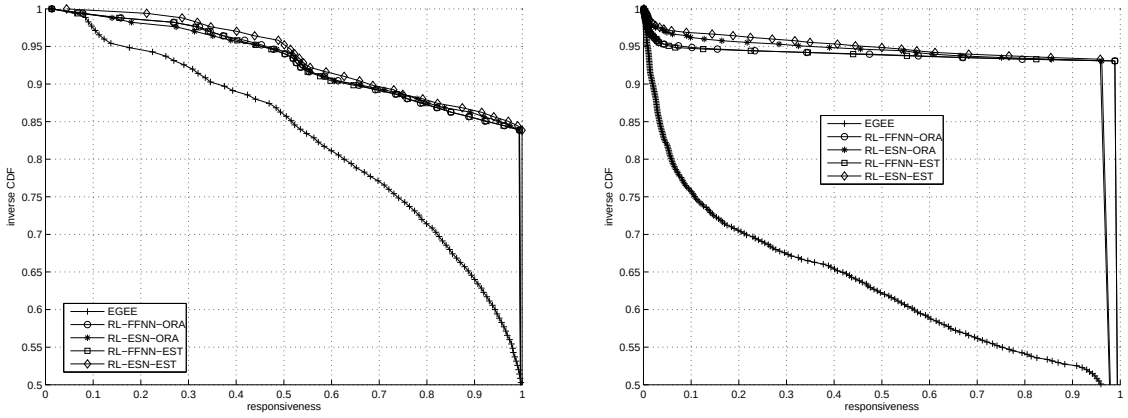

FIG. 7.25 – Distribution cumulative inverse de la réactivité pour les jobs batchs (gauche) et interactifs (droite), charge EGEE

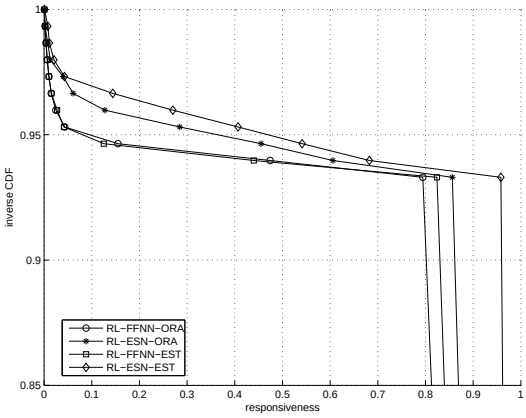

FIG. 7.26 – Distribution cumulative inverse de la réactivité des jobs interactifs, charge EGEE

La figure 7.25 présente les distributions cumulatives inverses des valeurs de réactivité

pour les jobs batchs puis interactifs de la trace utilisée. Concernant les jobs batchs, tous les algorithmes sont globalement satisfaisants, y compris l'ordonnancement natif dont la médiane de la réactivité se situe à 0.95. Toutefois, les algorithmes d'apprentissage par renforcement, offrant cette même valeur de réactivité de 0.95 à 85% des jobs considérés, confirment l'apport de cette famille d'algorithmes. Concernant les jobs interactifs, la médiane de la réactivité se situe à 0.8 dans le cas de l'ordonnancement natif, 95% de ces mêmes jobs interactifs connaissent cette valeur de réactivité dans le cas des algorithmes de RL. Le passage du contexte oracle vers une méthode d'estimation des durées d'exécution, pour la définition des variables d'environnement et d'action du MDP utilisé, ne semble pas dégrader les performances des algorithmes d'apprentissage par renforcement. L'estimation calculée semble donc être une information suffisante pour permettre un apprentissage d'une politique de contrôle efficace. Pour des raisons de lisibilité, la figure 7.26 présente les résultats de réactivité pour les jobs interactifs de la trace sans les performances de l'ordonnancement natif d'EGEE.

7.10.2 Délais d'attente

La table 7.26 résume les performances en terme de délais d'attente des jobs batchs et interactifs. Les algorithmes d'apprentissage par renforcement connaissent des apports consistants avec les observations effectuées sur les valeurs de réactivité dans la section précédente. Dans le cas des jobs batchs, le gain d'un facteur 4 en terme de délais d'attente moyen est observé, le gain d'un facteur d'environ 2 à 6 est à noter dans le cas des jobs interactifs. Le modèle d'apprentissage utilisant un Echo State Network et une estimation des durées d'exécution semble offrir les meilleurs résultats.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
EGEE	13041	26726	79	2756	8844	4
RL-NN-ORA	3777	12168	0	1366	7989	0
RL-NN-EST	3803	12232	0	1382	8076	0
RL-ESN-ORA	3962	13014	0	1085	8491	0
RL-ESN-EST	3492	12175	0	495	3589	0

Tab. 7.26 – Délais d'attente, charge EGEE. Temps en secondes

La figure 7.27 présente les distributions cumulatives des délais d'attente des jobs batchs et interactifs des différents modèles étudiés. L'algorithme RL-ESN semble offrir de meilleurs résultats dans le cas des jobs interactifs, bien qu'équivalents aux autres algorithmes d'apprentissage dans le cas des jobs batchs. L'utilisation d'un modèle estimé

des temps d'exécution semble offrir des performances marginalement meilleures que l'utilisation des durées d'exécution exactes de chaque job. Ce résultat peut s'expliquer par la réduction de bruit apportée par l'emploi d'une estimation des durées d'exécution pouvant améliorer la capacité de généralisation du modèle appris. Cette amélioration n'est toutefois pas observée dans le cas de l'algorithme RL-NN.

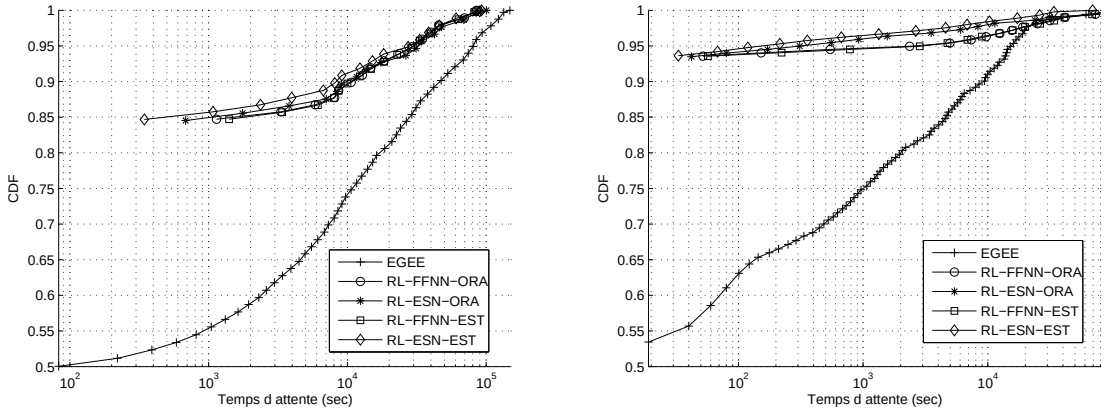

FIG. 7.27 – Distribution cumulative des délais d'attente pour les jobs batch (gauche) et interactifs (droite), charge EGEE

Malgré une médiane à 4 secondes, les résultats sont globalement mauvais dans le cas de référence. Ainsi, 35% des jobs interactifs attendent plus de deux minutes, 25% plus d'un quart d'heure et 10% plus de 3 heures. La forme beaucoup plus constante des apprentissages par renforcement a pour conséquence que de l'ordre de 5% des jobs subissent les même pénalités. Dans les deux cas considérés, 90% des jobs n'attendent pas plus d'un quart d'heure via un ordonnancement par apprentissage par renforcement, indépendamment du modèle d'apprentissage et de temps d'exécution utilisé. Seulement 30% des jobs dans le cas de l'ordonnancement natif, attendent au plus ce délai. Cette observation atteste de l'apport réel des algorithmes d'apprentissage par renforcement pour les utilisateurs finaux de l'infrastructure de grille dans ce cas réaliste.

La figure 7.28, de gauche, présente les distributions cumulatives des délais d'attente des jobs pour les différents modèles étudiés. Conformément aux remarques faites au sujet de la réactivité, ce sont les modèles de type ESN qui fournissent les apports les plus importants. Le modèle (RL-ESN-EST) offre un délai de d'attente, pour les durées les plus importantes constatées, au minimum 20% inférieur aux autres modèles considérés.

La figure de droite présente les délais d'attente les plus importants pour les jobs interactifs des ordonnancements par apprentissage par renforcement. Il est cependant important d'analyser plus en détail la queue de la distribution : elles correspondent aux attentes vraiment excessives qui sont, du point de vue des utilisateurs, assimilables à une défaillance.

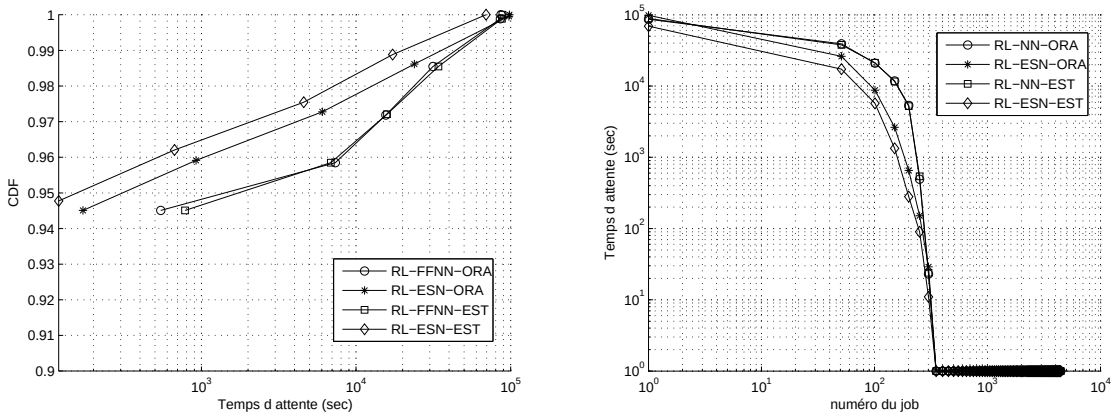

FIG. 7.28 – Distribution cumulative (gauche) et valeurs en fonction du rang (droite) des délais d'attente des jobs interactifs, charge EGEE

Il n'y a cependant pas de consensus sur la valeur du seuil correspondant, car les attentes (ou la résignation) des diverses communautés ne sont pas identiques. La figure 7.28, droite présente un diagramme rang-valeur (*rank plot*) des délais d'attente, qui permet de visualiser en détail les délais d'attente vraiment élevés : les jobs (en abscisse) sont ordonnés par délais d'attente décroissants.

La première constatation importante est la bi modalité de la distribution. Dans tous les cas, le dernier centile est supérieur à 2h, autrement dit 100 jobs attendent plus que cette durée, alors le 98 ème centile est très proche de 0. Ceci est cohérent avec le seuil observé sur la réactivité dans la section précédente. On note également la persistance du meilleur comportement de l'Echo State Network par rapport au réseau de neurones multicouche dans la queue de distribution. Mais, même dans ce cas, le fait que 1% des jobs soient aussi lourdement pénalisés reste problématique.

7.10.3 Équité

Les performances d'équité des algorithmes d'ordonnancement étudiés sont comparées à celles obtenues par l'ordonnanceur natif. En effet, l'ordonnancement PBS est en mesure de fournir de bonnes performances d'équité, l'objectif est de démontrer que les algorithmes d'ordonnancement RL sont en mesure d'optimiser la réactivité tout en respectant au mieux une contrainte d'équité donnée dans le cadre d'une infrastructure de calcul rigide.

Les différences de performance d'équité des différents algorithmes considérés sont présentées dans les figures 7.29 et 7.30. Celles-ci sont marginales si l'on s'en réfère aux échelles des valeurs observées, de l'ordre de 10^{-3}. Ce résultat est observé dans les cas oracle et estimé. Les algorithmes d'apprentissage par renforcement semblent donc capable

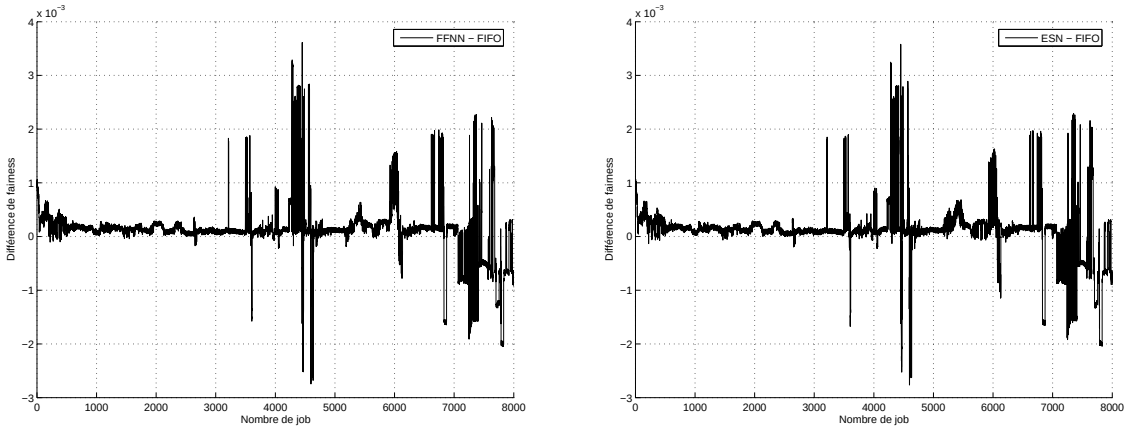

FIG. 7.29 – Différence d'équité de l'ordonnancement EGEE, modèle oracle

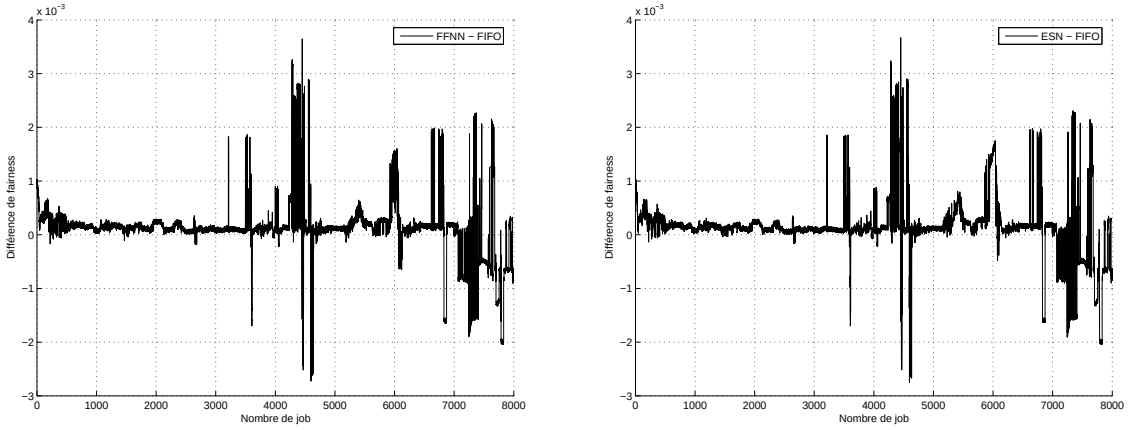

FIG. 7.30 – Différence d'équité de l'ordonnancement EGEE, modèle estimé

de fournir des performances d'équité au moins équivalentes à celles de l'ordonnanceur natif du site EGEE étudié.

7.10.4 Discussion

Cette première série d'expériences utilisant une trace réelle atteste de la faisabilité de la méthodologie proposée dans ce cadre réaliste. Le système est en mesure d'offrir de meilleures performances de réactivité et de délais d'attente que celles proposées par l'ordonnanceur natif du site considéré et équivalente dans le cas de l'objectif d'équité. Ces résultats montrent que la prioritisation des jobs interactifs est possible par le biais d'un système d'apprentissage par renforcement piloté par une mesure de réactivité comme fonction d'objectif. En outre, pour les deux modèles d'apprentissage étudiés, l'utilisation d'une estimation des durées d'exécution des jobs est réaliste. La méthodolologie reste donc valide au cas où ces informations seraient indisponibles. Concernant l'objectif d'équité, ce dernier

est correctement satisfait si l'on considère la différence faible avec l'ordonnancement natif du cluster étudié qui est déjà considéré satisfaisant.

7.11 Cas réel : ressources élastiques élargies

Dans cette première configuration élastique, le contrôleur de dimensionnement bénéficie d'un intervalle de 70 à 90 machines à mettre à disposition. On rappelle qu'on qualifie cette configuration d'*élargie* car le contrôleur est en mesure de sur-dimensionner le nombre de machines mise à disposition, en comparaison au cas rigide. Comme on l'a vu dans l'étude des délais d'attente, le système développé n'est pas complètement satisfaisant. Une fraction très limitée, mais non négligeable (de l'ordre de 1%) des jobs subit des délais d'attente intolérables, s'apparentant à une défaillance du gestionnaire des jobs. La question est alors de savoir s'il s'agit d'un problème intrinsèque à la charge de travail, ou d'une limitation de la stratégie d'apprentissage par renforcement. Par problème intrinsèque, nous voulons dire que l'infrastructure sous-jacente (le cluster) serait sous-dimensionnée par rapport aux excursions ponctuelles de la demande. Une solution possible est alors de proposer une adaptation de l'infrastructure à la demande, par l'ajout du MDP de dimensionnement. La suite de cette section est consacrée à cette étude.

Dans les deux cas considérés, *élargi* et *restreint*, l'intervalle du nombre de machines mobilisables a été choisi afin d'étudier l'efficacité de la méthode de dimensionnement au regard des 10000 jobs constituant la trace utilisée. En effet, en élargissant de manière trop importante l'espace du nombre de machines, une trace plus importante aurait été nécessaire afin de permettre au contrôleur d'explorer l'ensemble des dimensionnements possibles et d'estimer correctement la fonction d'espérance de gain associée.

La première série d'expérience (cas *élargi*) a donc pour but de montrer qu'il est possible d'améliorer la réactivité de l'infrastructure en autorisant le système à augmenter épisodiquement ses ressources, tout en respectant un compromis d'utilisation. On différencie les expériences utilisant un modèle oracle (ORA) et estimé (EST) pour les deux contrôleurs étudiés, d'ordonnancement et de dimensionnement. Les deux contrôleurs utilisent un modèle de type Echo State Network, pour l'apprentissage de la fonction d'espérance de gain Q, qui a montré son intérêt dans l'ensemble des séries d'expériences précédentes. La pondération, λ_e, de la fonction d'objectif pour le MDP de dimensionnement a été successivement placée à 0.0, 0.5 et 1.0. Au cours des deux séries d'expériences élastiques, les pondérations $\lambda_e = 0.0$ et $\lambda_e = 1.0$ ne doivent pas être considérées comme réalistes. Leur rôle est de fournir une borne de performance attendu par les contrôleurs de dimensionnement.

7.11.1 Réactivité

Les performances en termes de réactivité sont détaillées dans la table 7.27. Les résultats en réactivité des différentes méthodes d'apprentissage par renforcement sont globalement comparables et offrent un gain de 12% en moyenne dans le cas des jobs batchs et de 50% dans le cas des jobs interactifs, vis à vis de l'ordonnancement natif.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
EGEE	0.82	0.26	0.99	0.62	0.41	0.96	0.73	0.36	0.98
RL-ESN-ORA-00	0.93	0.16	1	0.94	0.21	1	0.94	0.19	1
RL-ESN-EST-00	0.90	0.21	1	0.89	0.27	1	0.90	0.24	1
RL-ESN-ORA-05	0.94	0.15	1	0.95	0.19	1	0.95	0.17	1
RL-ESN-EST-05	0.95	0.15	1	0.95	0.19	1	0.95	0.17	1
RL-ESN-ORA-10	0.94	0.15	1	0.92	0.24	1	0.93	0.20	1
RL-ESN-EST-10	0.94	0.16	1	0.92	0.24	1	0.93	0.19	1

Tab. 7.27 – Réactivité, modèle élastique élargi, charge EGEE.

La figure 7.31 présente les distributions cumulatives inverses des valeurs de réactivité. Elle atteste des gains présentés dans la table précédente. Considérons une réactivité minimale de 0.8, pour les jobs batchs, 70% des jobs bénéficient de cette valeur, contre 85% à 90% dans le cas de l'apprentissage par renforcement. Concernant les jobs interactifs, 55% de ceux-ci bénéficient d'une telle valeur de réactivité dans le cas de l'ordonnancement natif, contre approximativement 90% dans le cas des méthodes de renforcement. Dans les deux cas, la paramétrisation de $\lambda = 0.5$ offre de bons résultats, ce qui est encourageant car, dans cette configuration, le contrôleur de dimensionnement effectue un compromis entre réactivité et utilisation.

La figure 7.32 compare les distributions cumulatives inverses des valeurs de réactivité avec celles obtenues dans le cas rigide utilisant un ESN, décrites dans la section précédente. Pour une réactivité de 0.8, dans le cas des jobs batchs, 88% des jobs bénéficie au minimum de cette valeur dans les cas rigides (RL-ESN-ORA et RL-ESN-EST) contre 90% dans les (RL-ESN-ORA-05) et (RL-ESN-EST-05). Par ailleurs, (RL-ESN-ORA-10) offre également cette amélioration. Concernant les jobs interactifs, 94% des jobs bénéficient du gain de réactivité de 0.8 dans les deux configurations rigides, 95% des jobs en bénéficie dans les cas (RL-ESN-ORA-05) et (RL-ESN-EST-05).

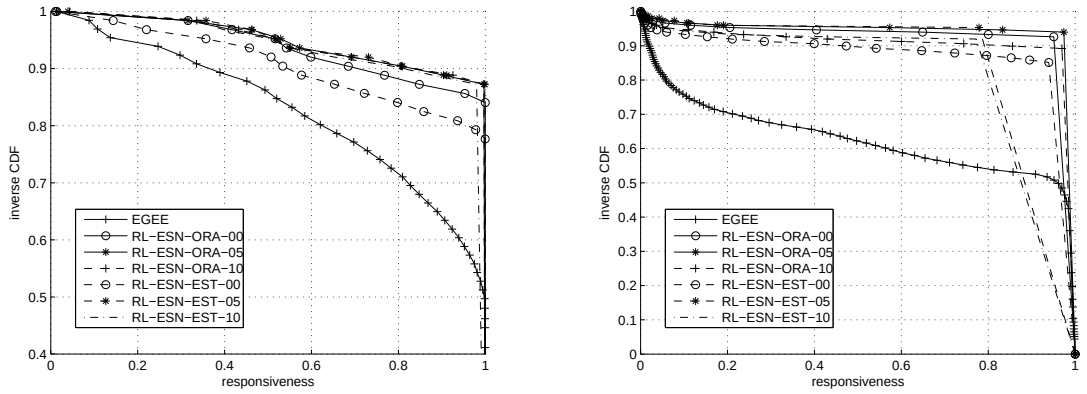

FIG. 7.31 – Distribution cumulative inverse de la réactivité pour les jobs batchs (gauche) et interactifs (droite), modèle élastique élargi, charge EGEE

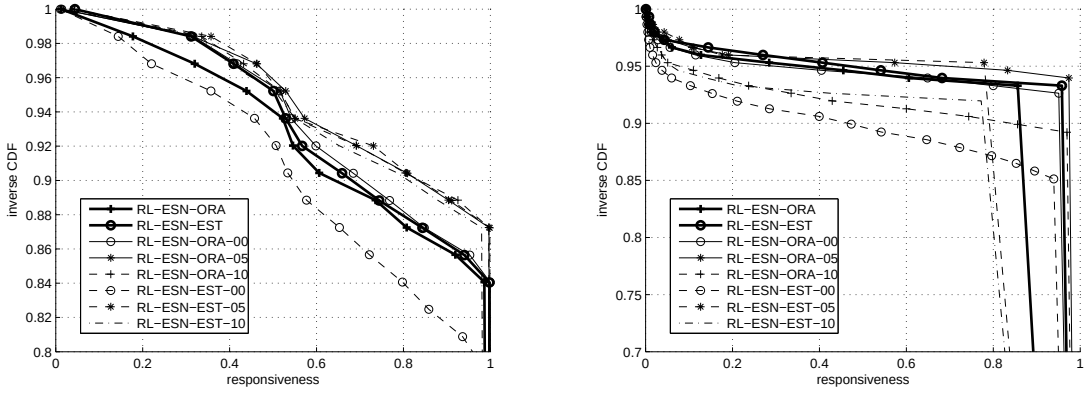

FIG. 7.32 – Distribution cumulative inverse de la réactivité pour les jobs batchs (gauche) et interactifs (droite), modèle élastique élargi, charge EGEE

7.11.2 Délais d'attente

Les performances en termes de délais d'attente sont détaillées dans la table 7.28. Bien que l'ensemble des méthodes proposées apportent un gain de l'ordre de 200 à 300% dans le cas des jobs batchs et interactifs, la paramétrisation de $\lambda = 0.5$ et $\lambda = 1.0$ semblent offrir les gains les plus significatifs, autant par l'emploi d'un modèle oracle ou estimé des durées d'exécution des jobs. En effet, en moyenne, les configurations où $lambda_e = 0.0$, n'optimisant donc que l'utilisation en dépit de la réactivité et à fortiori des délais d'attentes, présentent des résultats dégradés, ce qui était un comportement attendu.

La figure 7.33 présente les distributions cumulatives inverses des délais d'attente des méthodes considérées. Pour un délai d'attente de 2 minutes des jobs interactifs, 65% des jobs en bénéficient dans le cas natif, contre 85% à 95% dans le cas des méthodes par

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
EGEE	13041	26726	79.5	2756	8844	4	8158	20943	6
RL-ESN-ORA-00	3190	10988	0	1013	7776	0	2163	9669	0
RL-ESN-EST-00	5373	15055	0	2012	11092	0	3788	13438	0
RL-ESN-ORA-05	2663	10320	0	872	7274	0	1819	9057	0
RL-ESN-EST-05	2779	10571	0	507	4407	0	1707	8336	0
RL-ESN-ORA-10	2996	12427	0	754	4828	0	1939	9688	0
RL-ESN-EST-10	3270	13001	0	953	6776	0	2178	10597	0

TAB. 7.28 – Délais d'attente, modèle élastique élargi, charge EGEE.

apprentissage.

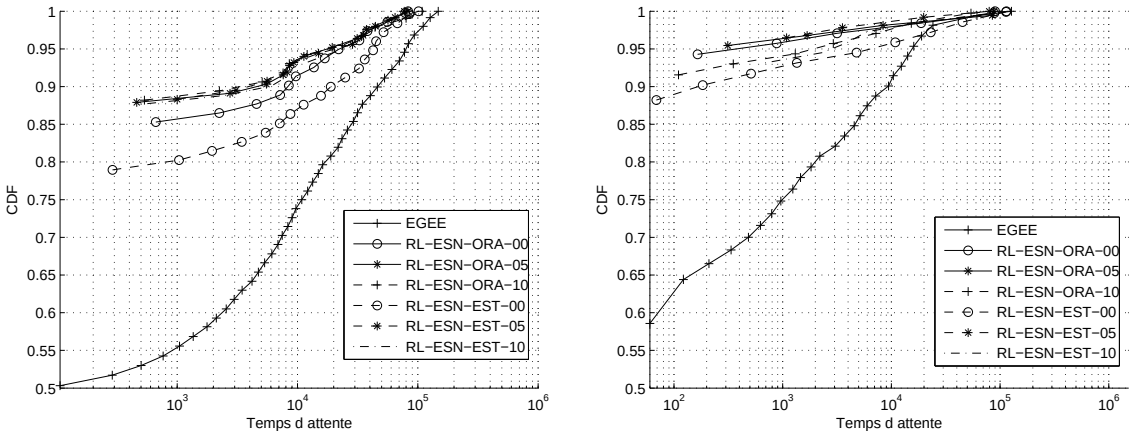

FIG. 7.33 – Distribution cumulative des délais d'attente pour les jobs batch (gauche) et interactifs (droite), modèle élastique, charge EGEE

La figure 7.34 présente les distributions cumulatives des délais d'attente comparées aux configurations rigides, de la section précédente, utilisant un modèle de type Echo State Network. Les améliorations constatées sur le critère de réactivité sont confirmées dans le cas des jobs batchs. Les configurations utilisant une pondération $\lambda_e = 0.5$ offrent les améliorations les plus constantes, quelque soit le modèle de durée d'exécution utilisé, *oracle* ou *estimé*. Les performances observées dans le cas des jobs interactifs sont marginalement meilleures pour les modèles améliorant ces performances des jobs batchs. Toutefois, les résultats pour $lambda_e = 1.0$ montrent que la paramétrisage du système n'est pas triviale et nécessitera une phase d'expérimentation pour le paramétrisage correct du système lors du passage en production.

La figure 7.35 présente les durées d'attente les plus importantes, triées par ordre

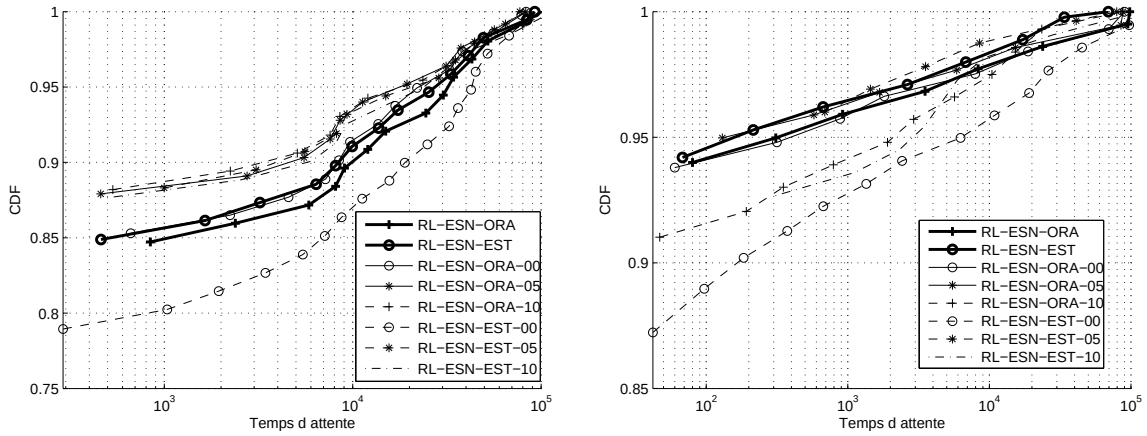

FIG. 7.34 – Distribution cumulative des délais d'attente pour les jobs batch (gauche) et interactifs (droite), modèle élastique, charge EGEE

décroissant, et les compare à un cas rigide de référence, (RL-ESN-EST), utilisé précédemment. Une très légère amélioration est constatée dans le cas (RL-ESN-EST-10). Ce résultat était attendu car le contrôleur de dimensionnement n'optimise que l'objectif de réactivité. La configuration (RL-ESN-EST-05) montre une altération du résultat s'explicant par le compromis nécessaire, dans ce cas, entre réactivité et utilisation.

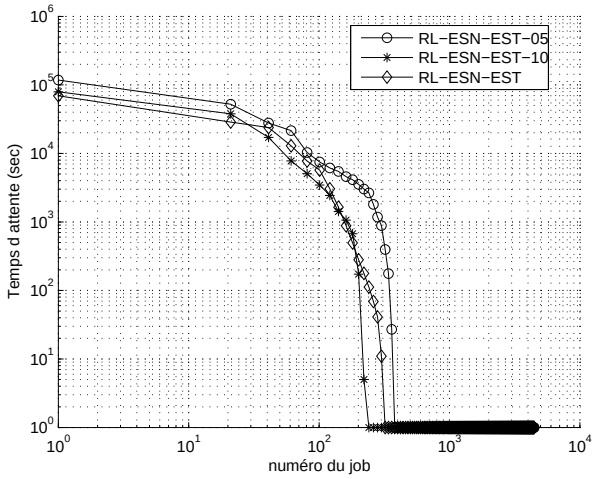

FIG. 7.35 – Valeurs en fonction du rang des délais d'attente des jobs interactifs, modèle élastique élargi, charge EGEE

Dans la sous-section suivante, nous nous assurons que les gains affichés dans ces deux premières parties n'affectent pas sévèrement les performances en utilisation de l'infrastructure.

7.11.3 Utilisation

La figure 7.36 présente les distributions cumulatives inverses des valeurs d'utilisation mesurées à chaque terminaison de job de chaque simulation. Pour des questions de lisibilité, les modèles oracle et estimés sont représentés sur deux figures séparées. Pour toutes les configurations étudiées, les différences restent marginales, aucune dégradatation notable de l'utilisation n'est à déplorer dans ce premier cas d'élasticité.

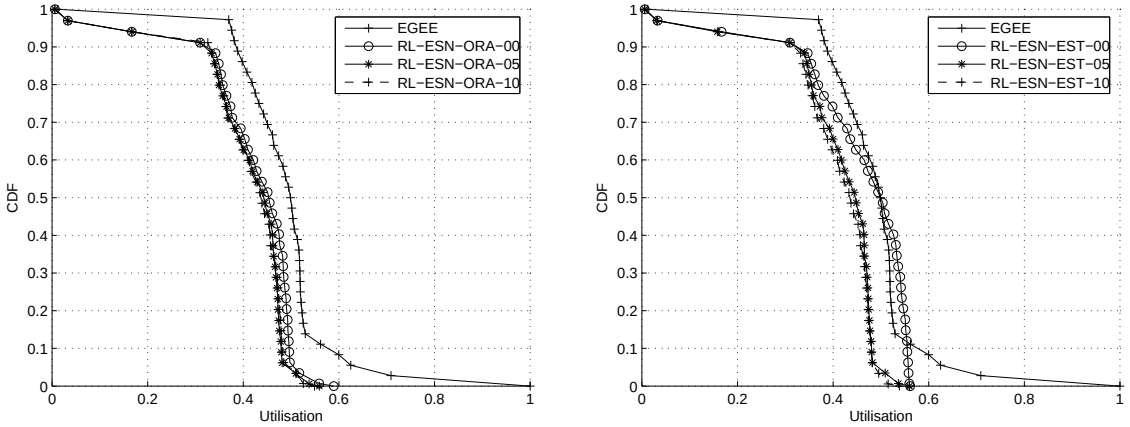

FIG. 7.36 – Distribution cumulative inverse de l'utilisation du cluster mesuré en fin d'exécution de chaque job, modèle oracle (gauche) et modèle estimé (droite)

7.12 Cas réel : ressources élastiques restreintes

Dans cette seconde configuration, le contrôleur de dimensionnement bénéficiera d'une marge de manoeuvre de 20 noeuds de calcul permettant de moduler la taille du parc actif entre 40 et 80 machines en fonction de l'intensité du flux de demandes. On rappelle que 80 est le nombre de machines utilisées dans le cas de référence. Cette dernière série d'expérience permettra de montrer si l'approche proposée est en mesure d'assurer une réactivité acceptable tout en améliorant l'utilisation du cluster.

7.12.1 Réactivité

La table 7.29 résume les performances de réactivité détaillées des différents algorithmes expérimentés. L'identification du meilleur modèle d'apprentissage n'est pas aussi immédiate que dans le cas d'un MDP unique ou d'un modèle élastique élargi. Ce sont les modèles (RL-ESN-ORA-10) et (RL-ESN-EST-10) qui, en termes de moyenne et de variance, semblent offrir les meilleurs gains à la fois pour la classe des jobs interactifs, avec un

gain de l'ordre de 10%, et batch, avec gain constaté de 45%. Ce résultat était, encore une fois, prévisible car le contrôleur se contente, dans cette configuration, de n'optimiser que la mesure de réactivité. Les modèles (RL-ESN-EST-05) et (RL-ESN-ORA-05) semblent permettre d'atteindre des résultats encourageants.

Algorithme	Batch			Inter			Total		
	Moy.	Std	Médiane	Moy.	Std	Médiane	Moy.	Std	Médiane
EGEE	0.82	0.26	0.99	0.62	0.41	0.96	0.73	0.36	0.98
RL-ESN-ORA-00	0.88	0.22	1.0	0.89	0.28	1.0	0.89	0.25	1
RL-ESN-EST-00	0.90	0.21	1	0.90	0.26	1	0.90	0.24	1
RL-ESN-ORA-05	0.90	0.21	1	0.91	0.25	1	0.91	0.23	1
RL-ESN-EST-05	0.88	0.23	1.0	0.87	0.31	1.0	0.87	0.27	1
RL-ESN-ORA-10	0.91	0.20	1.0	0.90	0.27	1.0	0.91	0.23	1
RL-ESN-EST-10	0.92	0.19	1.0	0.91	0.26	1.0	0.92	0.22	1

Tab. 7.29 – Réactivité, modèle élastique restreint, charge EGEE.

Sur la totalité de la trace, les valeurs de variances, faibles aux vues des valeurs de moyenne, rendent consistants ces résultats avec l'analyse précédente.

La figure 7.37 présente les distributions cumulatives inverses des valeurs de réactivité pour les jobs batchs et interactifs des expériences considérées. Les remarques concernant le mécanisme de rétention, très apparent dans le cadre des expériences synthétiques, n'est pas observé ici. Ce mécanisme semble devenir nécessaire au delà d'un niveau de charge important. Ce phénomène avait volontairement été mis en situation dans le cas des expériences synthétiques, afin de mettre clairement en évidence l'apport des algorithmes proposés. Malgré tout, l'amélioration des performances est nettement plus prononcée dans le cadre des jobs interactifs. En effet, 90% des jobs interactifs connaissent un gain de réactivité supérieur à 0.8, alors que 55% des jobs connaissent ce gain dans le cas de l'ordonnancement EGEE. Dans le cas des jobs batchs, EGEE assure à 70% des jobs cette valeur de réactivité de 0.8 alors que 90% des jobs ont bénéficié de cette valeur dans le cas des algorithmes RL précédemment cités. La recherche d'une bonne configuration dans ce cadre multiobjectif n'est toutefois pas triviale, les expérimentations ont mis en évidence une grande variabilité des résultats pouvant, en fonction du choix des paramètres expérimentaux, mener à des résultats inférieurs au cas de référence. Les mesures de variances des résultats de la section atteste également de ce fait.

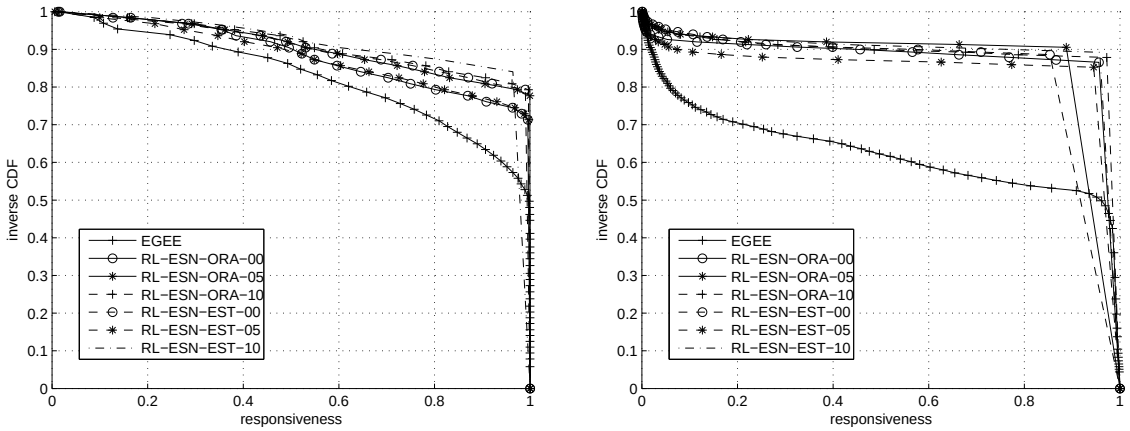

FIG. 7.37 – Distribution cumulative inverse de la réactivité pour les jobs batchs (gauche) et interactifs (droite), modèle élastique restreint, charge EGEE

7.12.2 Délais d'attente

Les durées d'attente avant exécution pour les jobs batchs et interactifs sont résumées dans la table 7.30. Les résultats sont consistants avec les résultats de réactivité présentés dans la section précédente.

Algorithme	Batch			Inter		
	Moy.	Std	Médiane	Moy.	Std	Médiane
EGEE	13041	26726	79	2756	8844	4
RL-ESN-ORA-00	6788	17025	0	2878	13268	0
RL-ESN-EST-00	5293	14931	0	1109	6543	0
RL-ESN-ORA-05	4863	13310	0	1108	6332	0
RL-ESN-EST-05	7307	19681	0	4476	20762	0
RL-ESN-ORA-10	4702	13669	0	1598	9596	0
RL-ESN-EST-10	3911	12505	0	964	4643	0

TAB. 7.30 – Délai d'attente, charge EGEE. Temps en secondes.

En accord avec les analyses faites de la réactivité, les modèles (RL-ESN-ORA-10) et (RL-ESN-EST-10) offrent les meilleures performances sur ce second indicateur. Par ailleurs, les résultats obtenus ici sont inférieurs à ceux observés dans le cas rigide, illustrés dans la table 7.26. Cette observation semble mettre en évidence la difficulté de paramétrer des systèmes de décision indépendants, même lorsque ceux-ci, par le biais d'actions différentes, optimisent la même fonction de gain. Ce phénomène peut être considéré comme un exemple de concurrence, où deux agents, par leurs actions, agissent sur l'état courant d'un environ-

nement commun où chacun évolut. Dans ces circonstances, l'apprentissage d'une politique de contrôle efficace peut être complexifié pour chaque agent. En accord avec les analyses de la réactivité, les modèles (RL-ESN-ORA-05) et RL-ESN-EST-05) semblent permettre d'offrir un gain vis à vis de l'architecture d'ordonnancement native d'EGEE, en divisant par plus de 2 le délai d'attente pour les jobs interactifs et batchs, tout en prenant en considération le gain en utilisation. Dans ce cas précis, la moyenne des délais d'attente pour les jobs interactifs est ramenée approximativement à 15 minutes en moyenne, en comparaison aux 45 minutes constatées dans le cas EGEE. Le résultat est donc encourageant, mais nécessite encore une analyse des performances de l'utilisation associées.

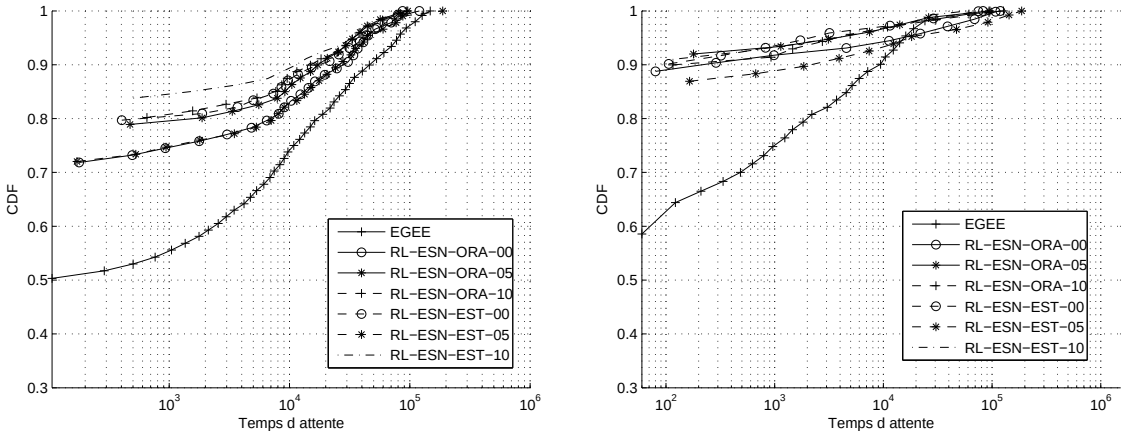

FIG. 7.38 – Distribution cumulative des délais d'attente pour les jobs batch (gauche) et interactifs (droite), modèle élastique, charge EGEE

Les distributions cumulatives des délais d'attente des jobs batchs et interactifs sont présentées dans la figure 7.38. Ici encore, les configurations ne tenant compte que de la performance en réactivité offrent les meilleurs résultats pour les jobs batchs et interactifs. Pour les 6 configurations d'apprentissage par renforcement étudiées, les temps d'attente des jobs interactifs sont toujours plus courts que pour les jobs batchs. Alors que 50% des jobs batchs connaissent un délai d'attente de moins de 2 minutes, c'est 70% de ces mêmes jobs qui n'attendent pas plus de ce délai dans le cas (RL-ESN-ORA-00) et plus de 80% dans les cas (RL-ESN-ORA-10) et (RL-ESN-EST-10). Dans le cas des jobs interactifs, alors que l'ordonnancement natif assure ce délai d'attente de 2 minutes à 60% de ceux-ci, c'est 90% des jobs de ce type qui connaissent une durée d'attente ne dépassant pas ce délai lorsque le paramètre $\lambda_e = 1.0$. La configuration (RL-ESN-EST-05) semble également parvenir à offrir de bonne performance en terme de délai d'attente tout en prenant en compte l'objectif d'utilisation.

La figure 7.39 montre les temps d'attentes les plus importants constatés dans la trace et les comparent au cas rigide. De manière prévisible, les durées d'attente les plus impor-

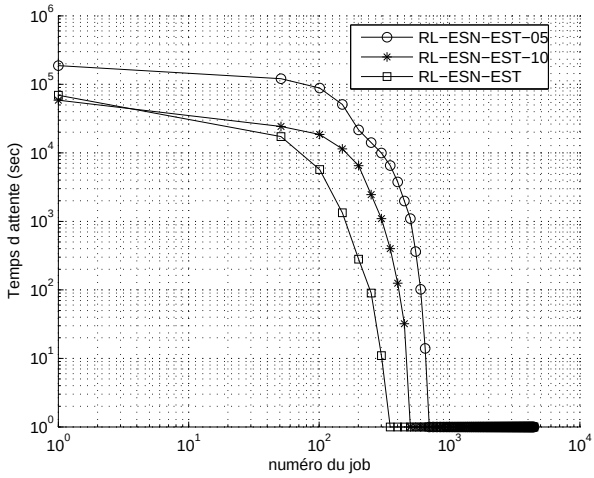

FIG. 7.39 – Valeurs en fonction du rang des délais d'attente des jobs interactifs, modèle élastique restreint, charge EGEE

tants ne sont pas améliorées car, dans le meilleur des cas, le système choisira d'utiliser en permanence le même nombre de machines que le cas rigide, (RL-ESN-EST-10). Ce cas ne se produira en pratique pas car le système de dimensionnement les solutions à sa disposition est réduira le nombre de machine lors de sa phase d'exploration.

7.12.3 Utilisation

On s'intéresse désormais aux performances en terme d'utilisation des différents modèles considérés. On mesurera cet indicateur à chaque fin d'éxecution de la trace d'activité utilisé pour chaque simulation.

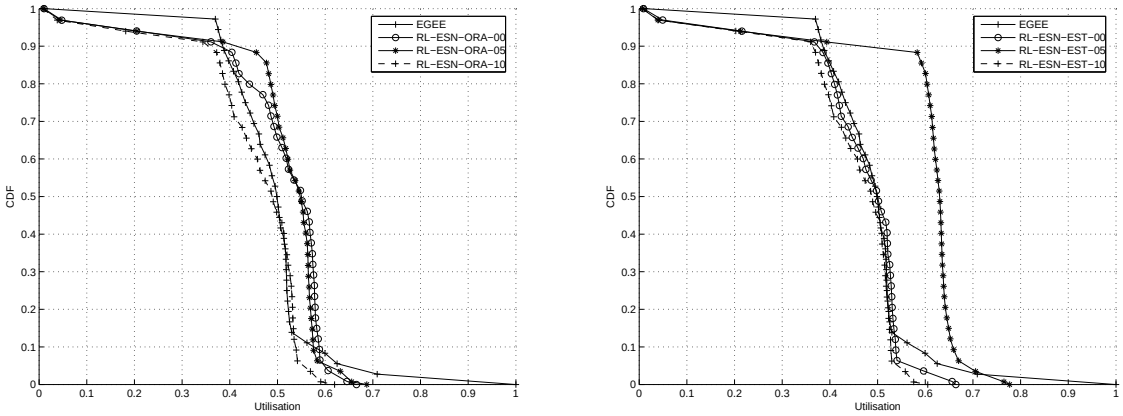

FIG. 7.40 – Distribution cumulative inverse de l'utilisation du cluster mesuré en fin d'exécution de chaque job, modèle oracle (gauche) et modèle estimé (droite)

La figure 7.40 présente la distribution cumulative inverse de l'utilisation du cluster, calculée à la fin de l'exécution de chaque job sur la trace d'exécution considérée. Les contrôleurs incorporant l'utilisation comme fonction d'objectif, avec $\lambda_e \in \{0.0, 0.5\}$, offrent les meilleurs gains pour cet indicateur. A contrario, les gains d'utilisation les plus faibles sont obtenus lorsque $\lambda = 1.0$. Ces résultats étaient envisagés car les gains d'utilisation les plus importants sont obtenus lorsque le contrôleur de dimensionnement prend en compte, dans sa fonction d'objectif associée, le gain en utilisation. C'est à dire, lorsque $\lambda_e \in \{0.0, 0.5\}$. Ces performances sont particulièrement satisfaisantes dans le cas du modèle (RL-ESN-EST-05). En effet, ce modèle a montré, dans les sections précédentes, qu'il était en mesure d'améliorer la réactivité de l'infrastructure d'ordonnancement en comparaison avec l'ordonnancement natif. Ce modèle semble donc simultanément capable d'améliorer l'utilisation de cette infrastructure.

7.13 Cas réel : Ressources élastiques, Discussion

Les trois dernières séries de résultats présentées, utilisant une trace d'activité extraite de la grille EGEE, ont montré que l'emploi de l'apprentissage par renforcement semble constituer une solution réaliste pour les deux problèmes de contrôle considérés, l'ordonnancement et le dimensionnement. Les deux dernières séries d'expériences exposées ont montré que l'élasticité permet d'améliorer la réactivité de l'infrastructure dans le cas d'une configuration de ressources élastique où le controleur peut, au besoin, élargir le nombre de machines du cluster, en comparaison à la taille du cluster de référence. Dans le cas de l'élasticité restreinte, l'utilisation de l'infrastructure a pu être améliorée sans sacrifier sévèrement la propriété de réactivité recherchée. Un ordonnancement peut donc être constitué de deux superviseurs disjoints, chacun en charge d'un aspect de la politique d'ordonnancement du site administré. En effet, l'ordonnancement dans les grilles de calcul est fréquemment décrit comme un système monolithique [LLM88] et non comme un ensemble de contrôleurs indépendants pouvant être administrés séparément.

Résumé

Dans ce dernier chapitre, la présentation des expériences effectuées s'est organisée autour des traces synthétiques puis réelles décrites au cours du chapitre précédent. Les expériences synthétiques ont permis de démontrer que des configurations de type Poisson exponentielle et à queue lourde, de type MMPP exponentielle, peuvent être efficacement traitées par les algorithmes d'apprentissage par renforcement. Cette première série d'expériences synthétiques a également permis de valider l'utilisation de la réactivité

comme fonction d'objectif d'un système d'apprentissage par renforcement en charge d'effectuer un ordonnancement préférentiel des jobs interactifs, sans pour autant pénaliser de manière significative les jobs à durée d'exécution longue. La manifestation du principe de rétention énoncé dans le chapitre 6 a été observé à plusieurs reprises, un sous-ensemble des jobs à longue durée d'exécution subit un délai d'attente plus important que dans le cas de l'ordonnanceur de référence afin de favoriser les jobs à faible durée d'exécution. Ce phénomène constitue un comportement souhaitable. La spécification d'un objectif d'équité infaisable a permis d'évaluer la robustesse des algorithmes d'apprentissage par renforcement dans ce contexte particulier, fréquemment observé en conditions réelles d'ordonnancement. Dans la seconde partie de ce chapitre, une trace réelle extraite des logs d'un site de la grille EGEE a servi de support aux analyses effectuées. L'apport des algorithmes d'apprentissage par renforcement a été prouvé, pour le problème de l'ordonnancement, dans le cas d'une connaissance parfaite (*oracle*) et estimée des durées d'exécution des jobs afin d'adhérer aux conditions réelles d'ordonnancement sur la grille étudiée. Dans une seconde sous partie, un système de dimensionnement dynamique du cluster a été adjoint et les apports ont été présentés dans le cadre de deux configurations de contrôle distinctes, restreinte et étendue. En outre, cette dernière série d'expériences a suggéré que le problème de l'ordonnancement peut être considéré comme un ensemble de sous problèmes indépendants, ici l'affectation des jobs aux machines du cluster et le dimensionnement de ce même cluster. Dans ce cas d'association de décideurs disjoints, la paramétrisation des algorithmes est cruciale car elle mène, dans certains des cas exposés, à des contre performances vis à vis de l'algorithme natif.

Chapitre 8

Conclusions et Perspectives

8.1 Apports

Nous avons souhaité étudier, dans cette thèse, l'utilisation de l'apprentissage artificiel pour le problème de l'ordonnancement réactif dans les grilles de calcul hautes performances. Après avoir rappelé et justifié le besoin de réactivité dans ces infrastructures, nous avons mis en évidence les difficultés des systèmes d'ordonnancement batch couramment utilisés à fournir une qualité de service satisfaisante. Nous avons mis en évidence que l'utilisation de fonctions d'objectifs explicites, telles que proposées dans l'approche générale de l'autonomic computing et plus pratiquement dans le paradigme de l'apprentissage par renforcement, permet de répondre efficacement à ce problème.

De là, nous nous sommes proposés de formaliser cette question de l'ordonnancement réactif comme un problème d'apprentissage par renforcement multi objectif en domaine continu. Cette formalisation nous a amené à effectuer un certain nombre de choix quant à la description des processus de décision de Markov à adopter. Ce sont les difficultés liées à ces choix de modélisation qui nous ont conduit à envisager l'utilisation d'un modèle d'apprentissage à mémoire, permettant de répondre aux difficultés liées au caractère non markovien des problèmes de décision traités. En effet, le manque d'informations disponibles concernant l'environnement et l'action du processus de décision considéré rendait pertinent l'utilisation d'un modèle à mémoire. Les expérimentations effectuées ont montré que dans un ensemble de cas synthétiques et réels, un modèle à mémoire, l'Echo State Network, offre de meilleurs résultats que les réseaux de neurones à rétro propagation du gradient classique, constituant aujourd'hui l'état de l'art dans ce domaine.

Enfin, nous nous sommes proposé de scinder le processus de décision en deux problèmes indépendants : le placement des jobs et le dimensionnement dynamique de l'infrastructure de calcul supervisée. Il nous a alors été possible d'améliorer, à la fois, la réactivité et l'utilisation du site de calcul. L'utilisation de traces réelles issues d'un site de calcul de la grille EGEE nous a permis de prouver l'apport de cette approche vis à vis des performances actuellement offertes par l'ordonnanceur le plus couramment utilisé dans les sites de la grille EGEE, PBS. Concernant la modélisation du problème de décision considéré, ces résultats expérimentaux nous ont également permis de montrer que la réactivité, associée à l'équité puis à l'utilisation, constituaient des fonctions de gain pertinentes dans ce contexte.

8.2 Limites et Perspectives

Cette thèse a proposé une formalisation d'un problème d'ordonnancement de tâches comme un processus de décision de Markov. Cette approche nécessite la définition d'un espace d'état et d'action composé d'un ensemble de variables. Aujourd'hui, la pertinence des

choix de modélisation effectués ne se valide que par l'analyse des performances atteintes par les systèmes apprenant, une fois celui-ci paramétré. Une première question serait de quantifier la pertinence d'une variable pour un problème de contrôle donné où la fonction de gain est définie. On se retrouve ici face à un problème particulier d'analyse en composante principale où le but serait d'identifier des corrélations éventuelles entre les variables d'états et d'actions du problème de contrôle traité avec la fonction de gain considérée. Un avantage d'un tel procédé serait de limiter l'impact lié à l'utilisation de variables descriptives éventuellement insignifiantes pour le problème de contrôle traité rendant la recherche d'une politique de contrôle efficace plus complexe.

Une seconde perspective de travail concerne l'algorithme d'apprentissage proposé dans cette thèse, cette observation est valable pour les deux processus de décision considérés. Dans cette procédure, on choisit d'effectuer un réapprentissage à un nombre d'évènements constant d'une simulation. Il serait intéressant d'envisager un mécanisme de réapprentissage conditionné par les performances obtenues par le système apprenant. En cas de changement brutal de profil de charge, comme observé dans le cas des traces synthétiques de type MMPP, proposées dans cette thèse, une hypothèse, au sens apprentissage statistique du terme, peut s'avérer produire une politique de contrôle sous-optimale lorsqu'un régime fortement chargé succède rapidement à un régime de charge moyenne ou faible. Sachant que cette sous optimalité est détectable par l'analyse en ligne des valeurs de gains obtenues, il est envisageable de proposer un critère qui adapterait les fréquences de réapprentissage des modèles de contrôle en fonction des valeurs de gain collectés au cours d'une simulation. Cette méthode pourrait également, dans le cas d'un algorithme de type SARSA, permettre d'économiser le coût de réapprentissages pouvant être optionel lorsque la politique de contrôle courante satisfait déjà les objectifs définis.

Une dernière question soulevée dans ce travail concerne l'utilisation d'une somme pondérée de fonctions d'objectifs comme modélisation d'un apprentissage par renforcement multiobjectif. Les problèmes abordés dans cette thèse présentent un nombre de restreint de fonctions d'objectifs, nous pouvons nous poser la question du passage à l'échelle de cette approche dans le cas d'un système décisionnel devant optimiser un nombre plus important de fonctions de gain. Nous pouvons ainsi ramener cette question à la question de l'optimisation en grande dimension où les approches à base de distances sont inefficaces. Suivant la même idée, un nombre trop important de fonctions d'objectifs rendrait inopérante une formalisation de problème de contrôle basée sur une somme pondérée des fonctions d'objectifs.

Bibliographie

[AAC+08] J. Allison, J. Apostolakis, G. Cosmo, P. Nieminen, and M. G. Pia. Geant4 status and results, 2008.

[AMS07] András Antos, Rémi Munos, and Csaba Szepesvári. Fitted Q-iteration in continuous action-space MDPs. In *NIPS*. MIT Press, 2007.

[Arh77] J. Arhnung. A.K. erlang. *Teleteknik No 4*, 4 :201–210, 1977.

[Ba08] Stephen Burke and al. *GLite 3.1 user guide*, 2008.

[Bar94] Andrew R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14(1) :115–133, 1994.

[BBNT08] Pete Beckman, Ivan Beschastnikh, Suman Nadella, and Nick Trebon. Building an infrastructure for urgent computing. In Lucio Grandinetti, editor, *High Performance Computing (HPC) and Grids in Action*, volume 16 of *Advances in Parallel Computing*, pages 75–95. IOS Press, 2008.

[BCD00] Rajkumar Buyya, Steve Chapin, and David Dinucci. Architectural models for resource management in the grid. In *In 1st IEEE/ACM Int'l Workshop on Grid Computing (Grid '00*, pages 86–2. Springer-Verlag, 2000.

[BCPV96] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate Progress : A Notion of Fairness in Resource Allocation. *Algorithmica*, 15(6), 1996.

[BE04] Yuriy Brun and Michael D. Ernst. Finding latent code errors via machine learning over program executions. In *Proceedings of the 26th International Conference on Software Engineering*, pages 480–490, 2004.

[Bed00] Olivier Bedelet. *Reconstruction et mesure de volume à partir d'images tomodensimétriques*. PhD thesis, LIMSI-CNRS, Orsay, Université Paris Sud, 2000.

[BEDB07] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška. Continuous-state reinforcement learning with fuzzy approximation. In K. Tuyls, S. de Jong, M. Ponsen, and K. Verbeeck, editors, *Proceedings of the 7th Annual Symposium on Adaptive and Learning Agents and Multi-Agent Systems (ALAMAS 2007)*, pages 21–35, 2007.

[Bel57] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[Bel61] Richard E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.

[BGH06] George K. Baah, Alexander Gray, and Mary Jean Harrold. On-line anomaly detection of deployed software : a statistical machine learning approach. In *Proceedings of the 3rd International Workshop on Software Quality Assurance*, pages 70–77, 2006.

[Bha92] Vasanthakumar N. Bhat. Switched poisson process/g/1 queue with service interruptions. *Comput. Oper. Res.*, 19(9) :751–756, 1992.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BLC05] Léon Bottou and Yann Le Cun. On-line learning for very large data sets : Research articles. *Appl. Stoch. Model. Bus. Ind.*, 21(2) :137–151, 2005.

[BLHS97] Barbro Back, Teija Laitinen, Jukkapekka Hekanaho, and Kaisa Sere. The effect of sample size on different failure prediction methods, 1997.

[BMM03] V. Breton, M. Medina, and J. Montagnat. Datagrid, prototype of a biomedical grid. In *Methods of Information in Medicine*, pages 200–3, 2003.

[BRH04] James F. Bowring, James M. Rehg, and Mary Jean Harrold. Active learning for automatic classification of software behavior. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis*, pages 195–205, 2004.

[BRMO03] Mark Brodie, Irina Rish, Sheng Ma, and Natalia Odintsova. Active probing strategies for problem diagnosis in distributed systems. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1337–1338, 2003.

[Bus08] Keith A. Bush. *An Echo State Model of non-Markovian Reinforcement Learning*. PhD thesis, Colorado State University, Fort Collins, 2008.

[CAN08] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *ICML '08 : Proceedings of the 25th international conference on Machine learning*, pages 144–151. ACM, 2008.

[Car93] Richard A. Caruana. Multitask learning : A knowledge-based source of inductive bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993.

[Car97] Rich Caruana. Multitask learning. In *Machine Learning*, pages 41–75, 1997.

[Cat03] Charlie Catlett. Standards for grid computing : Global grid forum. *Journal of Grid Computing*, 1(1), 2003.

[CB01] Walfredo Cirne and Francine Berman. A model for moldable supercomputer jobs. In *IPDPS*, page 59. IEEE Computer Society, 2001.

[CBL06] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[CCD06] C.Blanchet, C. Combet, and G. Deleage. Integrating bioinformatics resources on the egee grid platform. In *CCGRID '06 : Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, page 48, Washington, DC, USA, 2006. IEEE Computer Society.

[CDO07] Georges Da Costa, Marios D. Dikaiakos, and Salvatore Orlando. Nine months in the life of EGEE : a look from the south. In *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS'07, Istanbul, 24/10/2007-26/10/2007*, pages 281–287. IEEE, octobre 2007.

[CFA+07] John Cavazos, Grigori Fursin, Felix V. Agakov, Edwin V. Bonilla, Michael F. P. O'Boyle, and Olivier Temam. Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the 5^{th} International Symposium on Code Generation and Optimization*, pages 185–197, 2007.

[CM02] A. Cornuejols and L. Miclet. *Apprentissage Artificiel : Concepts et Algorithmes*. Eyrolles, 2002.

[Cou02] Remi Coulom. Feedforward neural networks in reinforcement learning applied to high-dimensional motor control. In *In 13th International Conference on Algorithmic Learning Theory*, pages 402–413. Springer, 2002.

[CSAW05] Hoi Chan, Alla Segal, Bill Arnold, and Ian Whalley. How can we trust a policy system to make the best decision ? *International Workshop on Policies for Distributed Systems and Networks*, 1 :123–126, 2005.

[CWG09] E. Clévédé, D. Weissenbach, and B. Gotab. Distributed jobs on EGEE Grid infrastructure for an Earth science application : moment tensor computation at the centroid of an earthquake. *Earth Science Informatics*, 2(1-2) :97–106, 2009.

[CZL+04] Mike Y. Chen, Alice X. Zheng, Jim Lloyd, Michael I. Jordan, and Eric A. Brewer. Failure diagnosis using decision trees. In *Proceedings of the 1^{st} International Conference on Autonomic Computing*, pages 36–43, 2004.

[DCA+03] Ronald P. Doyle, Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin M. Vahdat. Model-based resource provisioning in a web service utility. In *USITS'03 : Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*. USENIX Association, 2003.

[DCCC06] Jim Dowling, Raymond Cunningham, Eoin Curran, and Vinny Cahill. Building autonomic systems using collaborative reinforcement learning. *The Knowledge Engineering Review*, 21(3) :231–238, September 2006.

[Der70] C. Derman. *Finite State Markov Decision Processes*, volume 67 of *Mathematics in Science and Engineering*. Academic Press, Inc., 1970.

[DG06] Kalyanmoy Deb and Himanshu Gupta. Introducing robustness in multi-objective optimization. *Evolutionary Computation*, 14(4) :463–494, 2006.

[DH98] Gross Donald and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley Press, 1998.

[Die00] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13 :227–303, 2000.

[dMBA08] André da Motta, Salles Barreto, and Charles W. Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, 172(4–5) :454–482, 2008.

[Doy00] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12 :219–245, 2000.

[ea05] F. Gagliardi et. al. Building an infrastructure for scientific grid computing : status and goals of the egee project. *Philosophical Transactions of the Royal Society A*, 1833, 2005.

[ER09] Yariv Ephraim and William J. J. Roberts. An EM algorithm for markov modulated markov processes. *IEEE Transactions on Signal Processing*, 57(2) :463–470, 2009.

[FFR+04] Ian T. Foster, Markus Fidler, Alain Roy, Volker Sander, and Linda Winkler. End-to-end quality of service for high-end applications. *Computer Communications*, 27(14) :1375–1388, 2004.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15, 2001.

[FMH93] Wolfgang Fischer and Kathleen S. Meier-Hellstern. The markov-modulated poisson process (MMPP) cookbook. *Perform. Eval*, 18(2) :149–171, 1993.

[FMR07] Hanhua Feng, Vishal Misra, and Dan Rubenstein. PBS : a unified priority-based scheduler. In Leana Golubchik, Mostafa H. Ammar, and Mor Harchol-Balter, editors, *International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS*, pages 203–214. ACM, 2007.

[Fre04] Alex Freitas. A critical review of multi-objective optimization in data mining : a position paper. *SIGKDD Explorations*, 6(2) :77–86, 2004.

[FZRL09] Ian T. Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. *CoRR*, 2009.

[GM08] Konstantinos Georgakopoulos and Konstantinos Margaritis. *Integrating Condor Desktop Clusters with Grid*, pages 37–42. Springer US Publisher, 2008.

[Gor00] Geoffrey J. Gordon. Reinforcement learning with function approximation converges to a region. In *NIPS*, pages 1040–1046. MIT Press, 2000.

[GRLMT07] Cécile Germain-Renaud, Charles Loomis, Jakub T. Moscicki, and Romain Texier. Scheduling for responsive grids. *Journal of Grid Computing*, 2007.

[GTO05a] C. Germain, R. Texier, and A. Osorio. Exploration of Medical Images on the Grid. *Methods of Information in Medecine*, 2005. To appear.

[GTO05b] C. Germain, R. Texier, and A. Osorio. Interactive volume reconstruction and measurement on the grid. *Methods of Information in Medecine*, 44(2) :227–232, 2005.

[HKO+05] Murali Haran, Alan F. Karr, Alessandro Orso, Adam A. Porter, and Ashish P. Sanil. Applying classification techniques to remotely-collected program execution data. In *Proceedings of the 10^{th} European Software Engineering Conference held jointly with 13^{th} ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 146–155, 2005.

[Hor01] P. Horn. Autonomic computing manifesto, 2001. http ://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[III95] Leemon C. Baird III. Residual algorithms : Reinforcement learning with function approximation. In *ICML*, pages 30–37, 1995.

[Jac02] David B. Jackson. *Maui scheduler : a multifunction cluster scheduler*, pages 351–368. MIT Press, Cambridge, MA, USA, 2002.

[Jac04] David B. Jackson. *Grid scheduling with Maui/Silver*, pages 161–170. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[Jae01] H. Jaeger. *Short term memory in echo state networks.* GMD Forschungszentrum Informationstechnik, Sankt Augustin, 2001.

[Jae02] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *NIPS*, pages 593–600. MIT Press, 2002.

[JLT85] E. Douglas Jensen, C. Douglas Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, pages 112–122, 1985.

[JSC01] David B. Jackson, Quinn Snell, and Mark J. Clement. Core algorithms of the Maui scheduler. In *JSSPP '01 : Revised Papers from the 7th International*

Workshop on Job Scheduling Strategies for Parallel Processing, pages 87–102, London, UK, 2001. Springer-Verlag.

[JSOS99] Klaus Jansen, Roberto Solis-Oba, and Maxim Sviridenko. Makespan minimization in job shops : A polynomial time approximation scheme. In *STOC*, pages 394–399, 1999.

[JSOS03] Jansen, Solis-Oba, and Sviridenko. Makespan minimization in job shops : A linear time approximation scheme. *SIJDM : SIAM Journal on Discrete Mathematics*, 16, 2003.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, 2003.

[KD07] Jeffrey O. Kephart and Rajarshi Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1) :40–48, 2007.

[KKV05] Srikanth Kandula, Dina Katabi, and Jean-Philippe Vasseur. Shrink : a tool for failure diagnosis in ip networks. In *Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 173–178, 2005.

[KLC98] Leslie Pack Kaebling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2) :99–134, 1998.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning : a survey. *Journal of Artificial Intelligence Research*, 4 :237–285, 1996.

[KM96] V. Kreinovich and G. C. Mouzouris. Fuzzy rule based modeling as A universal approximation tool, 1996.

[KW04] Emre Kiciman and Yi-Min Wang. Discovering correctness constraints for self-management of system configuration. In *Proceedings of the 1st International Conference on Autonomic Computing*, pages 28–35, 2004.

[LBLO07] L.Amar, A. Barak, E. Levy, and M. Okun. An on-line algorithm for fair-share node allocations in a cluster. In *CCGRID*, pages 83–91, 2007.

[LBRT97] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing, 1997.

[LeC85] Y. LeCun. Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85*, pages 599–604, Paris, France, 1985.

[Leu04] Joseph Y-T. Leung. Some basic scheduling algorithms. In Joseph Y-T. Leung, editor, *Handbook of Scheduling — Algorithms, Models, and Performance Analysis*, Computer and Information Science Series. Chapman & Hall/CRC, 2004.

[LL90] M. J. Litzkow and M. Livny. Experience with the Condor distributed batch system. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, pages 97–101, 1990.

[LLM88] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - A hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE Computer Society, 1988.

[Llo09] I.M. Llorente. Researching cloud resource management and use : The stratuslab initiative. *The Cloud Computing Journal*, 2009.

[LM07] Hui Li and Michael Muskulus. Analysis and modeling of job arrivals in a production grid. *SIGMETRICS Performance Evaluation Review*, 34(4) :59–70, 2007.

[LMW06] Hui Li, Michael Muskulus, and Lex Wolters. Modeling job arrivals in a data-intensive grid. In *Proc.12 th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 210–231. Springer, 2006.

[LP06] George J. Lee and Lindsey Poole. Diagnosis of TCP overlay connection failures using bayesian networks. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 305–310, New York, USA, 2006. ACM Press.

[LRFH04a] Michael L. Littman, Nishkam Ravi, Eitan Fenson, and Rich Howard. An instance-based state representation for network repair. In *Proceedings of the 19^{th} AAAI Conference on Artificial Intelligence*, pages 287–292, 2004.

[LRFH04b] Michael L. Littman, Nishkam Ravi, Eitan Fenson, and Rich Howard. Reinforcement learning for autonomic network repair. In *Proceedings of the 1^{st} International Conference on Autonomic Computing*, pages 284–285, 2004.

[MBB98] Ryszard S. Michalski, Ivan Bratko, and Avan Bratko, editors. *Machine Learning and Data Mining ; Methods and Applications*. John Wiley & Sons, Inc., New York, USA, 1998.

[MBPS09] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *ICML*, volume 382. ACM, 2009.

[MCP+07] Buehner M., Anderson C., Young P., Bush K., , and Hittle D.C. Improving performance using robust recurrent reinforcement learning control. In *European Control Conference*, pages 1676–1681, 2007.

[MES08] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *IEEE Trans. Image Processing*, 17(1) :53–69, 2008.

[MHLM09] José Antonio Martín H., Javier Lope, and Darío Maravall. The kNN-TD reinforcement learning algorithm. In *IWINAC '09 : Proceedings of the 3rd*

International Work-Conference on The Interplay Between Natural and Artificial Computation, pages 305–314. Springer-Verlag, 2009.

[Mir06] I. Mirman. Going Parallel the New Way. *Desktop Computing*, 10(11), June 2006.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[MNM02] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states : A new framework for neural computation based on perturbations. *Neural Computation*, 14(11) :2531–2560, 2002.

[MP08] Emi Mathews and Axel Poigné. An echo state network based pedestrian counting system using wireless sensor networks. In *WISES*, pages 1–14. IEEE, 2008.

[MR97] Tara M. Madhyastha and Daniel A. Reed. Input/output access pattern classification using hidden markov models. In *Proceedings of the 5th Workshop on I/O in Parallel and Distributed Systems*, pages 57–67, 1997.

[MR07] Francisco S. Melo and M. Isabel Ribeiro. Q-learning with linear function approximation. In *20th Annual Conference on Learning Theory, COLT 2007.*, volume 4539 of *Lecture Notes in Artificial Intelligence*, pages 308–322. Springer, 2007.

[MS04] Artur Merke and Ralf Schoknecht. Convergence of synchronous reinforcement learning with linear function approximation. In *ICML*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[MST94] Donald Michie, David J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[MTG+04] Michael Mesnier, Eno Thereska, Gregory R. Ganger, Daniel Ellard, and Margo I. Seltzer. File classification in self-* storage systems. In *Proceedings of the 1st International Conference on Autonomic Computing*, pages 44–51, 2004.

[MWS+07] Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Alice X. Zheng, and Gregory R. Ganger. Modeling the relative fitness of storage. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pages 37–48, 2007.

[NB07] Mohammad Reza Nami and Koen Bertels. A survey of autonomic computing systems. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, page 26, Washington, DC, USA, 2007. IEEE Computer Society.

[OR02] James Oly and Daniel A. Reed. Markov model prediction of I/O requests for scientific applications. In *Proceedings of the 6th Annual ACM International Conference on Supercomputing*, pages 147–155, 2002.

[Pa04] J. Pruyne and al. Ws-agreement specification draft, june 2004.

[Pey09] G. Peyre. Sparse modeling of textures. *Journal of Mathematical Imaging and Vision*, 34(1), 2009.

[PGRKL08a] Julien Perez, Cécile Germain-Renaud, Balazs Kégl, and Charles Loomis. Grid differentiated services : A reinforcement learning approach. In *International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 287–294, Washington, DC, USA, 2008. IEEE Computer Society.

[PGRKL08b] Julien Perez, C'ecile Germain-Renaud, Bal'azs K'egl, and Charles Loomis. Utility-based reinforcement learning for reactive grids. *ICAC*, pages 205–206, 2008.

[PGRKL09] Julien Perez, Cécile Germain-Renaud, Balázs Kégl, and Charles Loomis. Responsive elastic computing. In *6th International Conference on Autonomic Computing*, pages 55–64, New York, NY, USA, 2009. ACM.

[PGRKL10] Julien Perez, Cécile Germain-Renaud, Balazs Kégl, and Charles Loomis. Multi-objective reinforcement learning for responsive grids. *Journal of Grid Computing (to appear)*, 2010.

[PH08] Sang-Min Park and Marty Humphrey. Feedback-controlled resource sharing for predictable escience. In *Super Computing*. IEEE/ACM, 2008.

[PKS01] Georgios Paliouras, Vangelis Karkaletsis, and Constantine D. Spyropoulos, editors. *Machine Learning and Its Applications, Advanced Lectures*, volume 2049 of *Lecture Notes in Computer Science*. Springer, 2001.

[PL95] J. Pruyne and M. Livny. Providing resource management services to parallel applications. In *Internation Parallel Processing Symposium*, 1995.

[PLT+08] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Internation Conference of Machine Learning*, volume 307 of *ACM International Conference Proceeding Series*, pages 752–759. ACM, 2008.

[RHW86] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323 :533–536, October 1986.

[RN94] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[Rob00] Lawrence G. Roberts. Beyond Moore's law : Internet growth trends. *Computer*, 33(1) :117–119, 2000.

[RR03] Manos Renieris and Steven P. Reiss. Fault localization with nearest neighbor queries. In *Proceedings of the 8th IEEE International Conference on Automated Software Engineering*, pages 30–39, 2003.

[RTI02] RTI. The economic impacts of inadequate infrastructure for software testing. *Technical report, RTI*, 2002.

[RW06] Carl Edward Rasmusen and Chris Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[SB87] R. S. Sutton and A. G. Barto. A temporal-difference model of classical conditioning. Technical report, GTE Laboratories & University of Massachusetts, 1987.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, Cambridge, MA, 1998.

[SC92] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6) :44, June 1992.

[Sch90] Jürgen Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In Richard Lippmann, John E. Moody, and David S. Touretzky, editors, *NIPS*, pages 500–506, 1990.

[SCJG00] Quinn Snell, Mark Clement, David Jackson, and Chad Gregory. The performance impact of advance reservation meta-scheduling. *Lecture Notes in Computer Science*, 1911 :137–149, 2000.

[SGL06] Istvan Szita, Viktor Gyenes, and András Lörincz. Reinforcement learning with echo state networks. In *16th International Conference on Artificial Neural Networks*, volume 4131 of *Lecture Notes in Computer Science*, pages 830–839. Springer, 2006.

[SH06] Mark D. Skowronski and John G. Harris. Minimum mean squared error time series classification using an echo state network prediction model. In *ISCAS*. IEEE, 2006.

[Sit91] H. Sitaraman. Approximation of some markov-modulated poisson processes. *INFORMS Journal on Computing*, 3(1) :12–22, 1991.

[SKS06] Sudha Srinivasan, Savitha Krishnamoorthy, and P. Sadayappan. Robust scheduling of moldable parallel jobs. *Int. J. of High Performance Computing and Networking*, 2 :120–132, feb 2006.

[Smi02] Andrew James Smith. Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15 :1107–1124, 2002.

[SMSM99] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063. The MIT Press, 1999.

[Son71] Edward Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

[SOR+03] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the 9^{th} ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435, 2003.

[SS04] Malgorzata Steinder and Adarshpal S. Sethi. Probabilistic fault localization in communication systems using belief networks. *IEEE/ACM Transactions on Networking*, 12(5) :809–822, 2004.

[SS05] Ulf D. Schiller and Jochen J. Steil. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63 :5–23, 2005.

[SSP04] David G. Sullivan, Margo I. Seltzer, and Avi Pfeffer. Using probabilistic reasoning to automate software tuning. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, pages 404–405, 2004.

[ST94] S. Sehad and C. Touzet. Self-organizing map for reinforcement learning : obstacle-avoidance with khepera. In P. Gaussier and J. D. Nicoud, editors, *Proceedings. From Perception to Action Conference*, pages 420–3. LERI-EERIE, Nimes, France, IEEE Comput. Soc. Press, 1994.

[Sta06] Garrick Staples. Torque resource manager. In *Super Computing*. ACM Press, 2006.

[Sut88] Richard S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Kluwer Academic Publishers, 1988.

[Sut92] Richard S. Sutton, editor. *Reinforcement Learning*, volume 173 of *SECS*. Kluwer Academic Publishers, 1992.

[SW98] Sergey V. Sevastianov and Gerhard J. Woeginger. Makespan minimization in open shops : A polynomial time approximation scheme. *Math. Program*, 82 :191–198, 1998.

[Tak69] Lajos Tak‘acs. On erlang’s formula. *The annals of mathematical Statistics*, 40 :71–78, 1969.

[TDC+08] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems 20*, pages 1497–1504, 2008.

[Tes92] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8 :257, 1992.

[Tes02] Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134 :181–199, 2002.

[Tes07] Gerald Tesauro. Reinforcement learning in autonomic computing : A manifesto and case studies. *IEEE Internet Computing*, 11(1) :22–30, 2007.

[tH01] Stephan ten Hagen. *Continuous State Space Q-Learning for Control of Nonlinear Systems*. PhD thesis, Computer Science Institute, University of Amsterdam, Nederland, February 2001.

[TJDB07] Gerald Tesauro, Nicholas Jong, Rajarshi Das, and Mohamed Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3) :287–299, 2007.

[TK04] Gerald J. Tesauro and Jeffrey O. Kephart. Utility functions in autonomic systems. In *Proceedings of the 1st International Conference on Autonomic Computing(ICAC'04)*, pages 70–77, 2004.

[T'K06] Vincent T'Kindt. J.Y.-T. leung, handbook of scheduling : Algorithms, models and performance analysis, computer and information science series. *European Journal of Operational Research*, 170(3) :1018–1022, 2006.

[Tol32] Edward C. Tolman. Purposive behavior in animals and men. *The Psychoanalytic review*, 1932.

[TWK05] Gerald J. Tesauro, William E. Walsh, and Jeffrey O. Kephart. Utility-Function-Driven Resource Allocation in Autonomic Systems. In *Proceedings of the 2nd International Conference on Autonomic Computing(ICAC'05)*, pages 342–343, 2005.

[UV98] William T. B. Uther and Manuela M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *In Proceedings of AAAI-98*, pages 769–774, 1998.

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning*. Springer Verlag, 1995.

[VC74] V. Vapnik and A. Chervonenkis. *Theory of pattern recognition*. Nauka, Moscow, 1974.

[Ven05] David Vengerov. A Reinforcement Learning Framework for Utility-Based Scheduling in Resource-Constrained Systems. Technical Report TR-2005-141, Sun Labs, 2005. to appear in the Grid Computing Journal.

[Ven07a] Ganesh K. Venayagamoorthy. Online design of an echo state network based wide area monitor for a multimachine power system. *Neural Networks*, 20(3) :404–413, 2007.

[Ven07b] D. Vengerov. A reinforcement learning approach to dynamic resource allocation. *Engineering Applications of Artificial Intelligence*, 20(3), 2007.

[vEvW08] Nees Jan van Eck and Michiel C. van Wezel. Application of reinforcement learning to the game of othello. *Computers & OR*, 35(6), 2008.

[VM02] Ricardo Vilalta and Sheng Ma. Predicting rare events in temporal domains. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 474–481, 2002.

[WAA+04] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with cart models. In *Proceedings of the 12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 588–595, 2004.

[Wan92] L. X. Wang. Fuzzy systems are universal approximators. In *Procedings of IEEE International Conference on Fuzzy Systems*, pages 1163–1169, San Diego, CA, 1992.

[Wat87] Raymond Watrous. Learning algorithms for connectionist networks : Applied gradient methods of nonlinear optimization. In Maureen Caudill and Charles Butler, editors, *IEEE First International Conference on Neural Networks*, volume 2, pages 619–627, San Diego, CA, June 21–24 1987.

[Wat89] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, England, 1989.

[WCS06] Xuesong Wang, Yuhu Cheng, and Wei Sun. Q learning based on self-organizing fuzzy radial basis function network. In *Advances in Neural Networks - ISNN 2006, Third International Symposium on Neural Networks.*, volume 3971 of *Lecture Notes in Computer Science*, pages 607–615. Springer, 2006.

[WFRT07] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning : a hierarchical bayesian approach. In *ICML '07 : Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, New York, NY, USA, 2007. ACM.

[Wil] Andreas Willig.

[Wil99] Andreas Willig. A short introduction to queueing theory, 1999.

[WPA+03] Gary A. S. Whittle, Jehan-François Pâris, Ahmed Amer, Darrell D. E. Long, and Randal C. Burns. Using multiple predictors to improve the accuracy of file access predictions. In *IEEE Symposium on Mass Storage Systems*, pages 230–240, 2003.

[WS06] S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7 :877–917, 2006.

[WSH99] Richard Wolski, Neil T. Spring, and Jim Hayes. The network weather service : a distributed resource performance forecasting service for metacomputing. *Future Generation Comp. Syst*, 15(5-6) :757–768, 1999.

[WSWD07] Jonathan Wildstrom, Peter Stone, Emmett Witchel, and Michael Dahlin. Machine learning for on-line hardware reconfiguration. In *Proceedings of the 20^{th} International Joint Conference on Artificial Intelligence*, pages 1113–1118, 2007.

[WSY07] Yilei Wu, Qing Song, and Xulei Yang. Robust recurrent neural network control of biped robot. *J. Intell. Robotics Syst.*, 49(2) :151–169, 2007.

[WTT+05] Ph. Wieder, N. Tonellotto, N. Tonellotto, R. Yahyapour, and R. Yahyapour. A proposal for a generic grid scheduling architecture. In *Proceedings of the Integrated Research in Grid Computing Workshop*, pages 337–346, 2005.

[YM05] Kenji Yamanishi and Yuko Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceeding of the 11^{th} ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 499–508, New York, USA, 2005. ACM Press.

[Zel05] A. Zeller. *Why Programs Fail : A guide to Systematic Debugging*. Morgan Kaufman, 2005.

[ZJLA03] Alice X. Zheng, Michael I. Jordan, Ben Liblit, and Alexander Aiken. Statistical debugging of sampled programs. In *Advances in Neural Information Processing Systems 15*, 2003.