Contrôle de qualité optimal d'applications multimédia

Loïc Strus

Université Joseph Fourier

THESE

pour obtenir le grade de

DOCTEUR DE L'UJF

préparée au laboratoire VERIMAG

dans le cadre de l'école doctorale « Mathématiques et Informatique »

présentée et soutenue publiquement par

Loïc STRUS

le 19 septembre 2008

Contrôle de qualité optimal d'applications multimédia

JURY

Président	Jean-François MEHAUT	UJF
Rapporteurs	Isabelle PUAUT	Rennes
	Marc POUZET	Paris Sud
Examinateurs	Hervé MARCHAND,	INRIA Rennes
Co-Directeur de thèse	Joseph SIFAKIS,	CNRS
	Jean-Claude FERNANDEZ,	UJF

Résumé

Ce manuscrit présente une méthode de contrôle de qualité de service à grain fin d'applications multimédia. Celle-ci permet le contrôle d'applications dont les actions sont paramétrées par des niveaux de qualité et dont les durées d'exécution ne sont pas connues. Le contrôle consiste en la construction d'un ordonnancement et le choix des niveaux de qualité respectant des propriétés de sûreté et d'optimalité. C'est-à-dire que l'on cherche à maximiser l'utilisation du budget de temps sans pour autant le dépasser tout en ayant un choix de qualité régulier.

Le contrôleur utilise une politique de gestion de qualité permettant de choisir pour chaque action l'ordonnancement et le niveau de qualité respectant les contraintes de qualité de service.

Nous étendons et améliorons les résultats précédents dans deux directions. La première propose une approche symbolique de la politique de gestion de qualité. Celle-ci utilise un diagramme de vitesses qui est une représentation graphique du comportement de l'application contrôlée. À partir de cette représentation, nous avons proposé une technique de contrôle permettant de relâcher le nombre d'appels au contrôleur tout en respectant les propriétés de sûreté et d'optimalité.

Nous avons ensuite proposé une approche stochastique du problème basée sur des fonctions de distribution de probabilités pour les durées d'exécution des actions. Notre méthode donne la possibilité à l'utilisateur de fixer la criticité des contraintes temps-réel. Elle permet aussi de calculer à priori le taux attendu de dépassement des échéances.

Ces résultats théoriques ont été appuyés par des expériences réalisées sur un encodeur vidéo s'exécutant sur machine nue.

Mots-clés : Systèmes embarqués, Systèmes temps-réel, Contrôle de qualité de service, Modèle Stochastique.

Title : Optimal QoS control for multimedia applications

Abstract

We present a method of fine grain QoS control for multimedia applications. The method allows the control of application software whose actions are parameterized by a quality level parameter and whose execution times are unknown. The method allows the construction of a Controller which computes adequate action schedules and corresponding quality levels, so as to meet safety and optimality for a given platform. In other words, we want to have a maximal utilization of the available time budget without exceed it.

The Controller uses a quality management policy for choosing, for each action, a schedule and quality levels meeting the QoS requirements.

We extend and improve results of previous into two directions. We propose a symbolic quality management method using speed diagrams, a representation of the controlled system's dynamics. From this representation we propose a method allowing relaxing the number of call to the Controller with respect to the QOS requirements.

In a second time, we developed a stochastic approach based on probability distribution functions for the execution time of actions. Our method is parameterized according to the importance attributed to deadline misses. This means that the user is given the possibility to express how hard the real time constraints are. Besides, given a value of the parameter, we can compute the expected deadline miss ratio for the controlled application.

We present experimental results including the implementation of the method and benchmarks for a video MPEG4 encoder running on a bare machine.

Keywords : Embedded System, Real-time Systems, QoS Control, Stochastic model.

Remerciements

Je tiens tout d'abord à remercier tous les membres du jury, plus particulièrement Isabelle Puaut et Marc Pouzet les deux rapporteurs pour avoir accepté cette lourde tâche de relecture minutieuse de mon manuscrit. Je remercie aussi Hervé Marchand qui a participé au jury de ma soutenance et Jean-François Méhaut d'avoir bien voulu en être le président.

Je remercie mes deux co-directeurs Joseph Sifakis et Jean-Claude Fernandez qui m'ont proposé ce travail de thèse. Ils m'ont aiguillé tout au long de ces trois années par leurs connaissances scientifiques et leur rigueur. Ils m'ont aussi orienté et conseillé dans mon travail de recherche.

Je tiens tout particulièrement à remercier Jacques Combaz qui a été mon co-bureau durant cette période. Je lui témoigne toute ma gratitude pour son soutien moral, scientifique et technique. Toujours disponible, il a pris le temps de répondre à mes nombreuses questions. Je le remercie aussi pour tous les très bons moments que l'on a partagé.

Je remercie également tous les membres du laboratoire Vérimag de m'avoir accueilli et de m'avoir fourni le matériel me permettant de réaliser toutes les expérimentations indispensables à mon travail. Je remercie l'équipe DCS qui m'a permis de présenter à plusieurs reprises mes travaux et ainsi de faire avancer mes recherches. Sans oublier toutes les personnes qui ont contribué à la bonne ambiance nécessaire à la réalisation de ma thèse : Manuel, Hugo, Guillaume, François-Xavier, Simon, Marcelo, Sophie, Pascal, Marion, et tous ceux que j'oublie.

Je remercie le département d'informatique pédagogique de l'université Stendhal, qui m'a donné la possibilité d'enseigner et de prendre part à toutes les tâches qui incombent au métier d'enseignant chercheur.

Tout cela n'aurait pas été possible sans les moments de détente. Je remercie donc tous les handballeurs que j'ai pu rencontrer sur les terrains comme en soirée : Joris, Pierro, Stephane, Greg, Alex, Andy et tous ceux que j'ai oubliés.

Je tiens pour finir à remercier tout particulièrement mes parents pour leur soutien, mais aussi Erika pour sa présence de tous les instants.

Table des matières

CHAPITRE

Introduction

1

1.1 Contexte

Les applications multimédia sont le plus souvent coûteuses que ce soit en termes de consommation mémoire, mais surtout en termes de puissance de calcul. De nos jours, on peut voir l'émergence de ces applications dans le domaine des systèmes embarqués temps-réel. Ce contexte donne lieu, pour des traitements multimédia, à deux problèmes pour les développeurs. Le premier concerne l'aspect embarqué de telles applications. En effet les systèmes embarqués sont souvent des systèmes complexes [1] (accélérations matérielles, architectures parallèles, caches, etc...) mais contraints en termes de ressources, c'est-à-dire qu'ils possèdent des processeurs limités en termes de performances, une taille de mémoire réduite, mais aussi une batterie adaptée à une certaine utilisation. Cependant les algorithmes des applications multimédia sont très coûteux et les adapter à des ressources limitées constitue une réelle difficulté. Ainsi les systèmes embarqués conduisent à créer des applications dont la prédictabilité est très faible en ce qui concerne l'utilisation des ressources en particulier dans la prévision des durées d'exécution. Le deuxième problème concerne l'aspect temps-réel. Il est évident qu'une application multimédia dont l'objectif est d'encoder le flux d'un média, un son ou une vidéo par exemple, doit s'exécuter dans un délai satisfaisant. Ces applications sont le plus souvent réalisées en direct et donc doivent s'exécuter à un rythme précis afin de garder une qualité de service satisfaisante [2, 3]. A partir de ces deux problèmes, nous comprenons l'opposition qu'il peut exister entre garantir des contraintes temporelles et la difficulté d'obtenir une prédiction des durées d'exécution. De ce fait réaliser des applications respectant le temps-réel sans avoir de certitude en ce qui concerne les durées d'exécution devient difficile, mais constitue un réel challenge.

Ce problème n'est pas nouveau et la solution choisie dans le monde industriel est de l'éviter. En effet les développeurs reprogramment des applications multimédia génériques de manière à les remanier pour qu'elles puissent être adaptées à la plateforme d'exécution cible. Cette méthodologie est d'un point de vue théorique assez simple à réaliser, mais de nombreux problèmes apparaissent. En effet le fait de séparer le développement du logiciel de la conception matérielle va permettre l'émergence de problèmes de bogue non pas dûs à une mauvaise programmation, mais plutôt à des erreurs de transmission des données. Il est donc évident que le travail du développeur n'est pas focalisé sur l'optimisation de son code mais plutôt sur une partie fastidieuse qui est la validation et le déboggage.

Il est donc intéressant de proposer une méthodologie de génération d'applications

basée sur le contrôle du comportement réel de celles-ci sur des plateformes. Ainsi la méthodologie que nous proposons est basée sur l'adaptation dynamique du comportement de l'application. Elle prend en entrée une description de celle-ci ainsi que ses durées d'exécution observées sur la plateforme cible. Cette méthodologie est basée sur du code applicatif que l'on veut générique, c'est-à-dire un code qui peut être utilisé pour différentes plateformes. Son objectif est de garantir la bonne adéquation entre l'application et la plateforme, c'est-à-dire de respecter les contraintes temporelles.

1.1.1 Systèmes embarqués et temps-réel

Les systèmes embarqués temps-réel sont de nos jours présents partout, cependant pour la plupart des personnes ces mots ne leurs sont pas familiers et ils n'en comprennent donc pas le sens. Nous allons dans cette première partie définir ces systèmes et ainsi situer le contexte dans lequel se place cette thèse.

1.1.1.1 Système embarqué

Un système embarqué peut être défini comme un système informatique et électronique autonome, qui est dédié à une seule ou plusieurs tâches spécifiques. A l'inverse un ordinateur personnel classique peut réaliser plusieurs tâches différentes. Ces systèmes font le plus souvent partie d'un système complet incluant des parties logicielles mais aussi matérielles. Par exemple un téléphone est composé de plusieurs systèmes embarqués lui permettant de communiquer et de réaliser toutes les autres fonctions dont il dispose. Les systèmes embarqués sont le plus souvent utilisés pour contrôler ces systèmes complets.

De nos jours les systèmes embarqués se retrouvent dans des domaines très différents comme le matériel tout public, tels que des lecteurs multimédia, des téléphones portables ou encore des télévisions. Ils sont aussi intégrés dans des domaines plus spécifiques et critiques tels que le système de pilotage des avions, les contrôleurs de centrales nucléaires ou les appareils médicaux informatisés. Ces systèmes peuvent donc avoir des complexités très différentes. En effet cela peut varier de systèmes composés d'un seul processeur et d'une LED à des systèmes possédant plusieurs microprocesseurs, des périphériques (écran ...) et une connexion réseau.

A la différence des ordinateurs personnels, pouvant évoluer selon l'utilisation et les besoins, les systèmes embarqués ont la particularité de posséder des ressources matérielles qui ne peuvent être modifiées tout au long de leur vie. Les systèmes embarqués sont donc assez limités au niveau de leurs ressources que ce soit au niveau de la mémoire, de la puissance de calcul et des batteries. Les ressources sont choisies en fonction des besoins initiaux du système embarqué. La réalisation des systèmes embarqués allie donc les deux entités des systèmes. C'est-à-dire qu'il est nécessaire de créer conjointement la partie matérielle et la partie logicielle.

Les systèmes embarqués grand public, comme les téléphones et les GPS par exemple,

sont soumis à de nombreuses contraintes que ce soit au niveau :

- du coût, du fait de l'importance du nombre d'exemplaires demandés,
- de la consommation d'énergie, lorsqu'il s'agit d'appareil électriquement autonomes,
- de la performance, avec l'explosion des fonctionnalités demandées à chaque système (un téléphone n'est plus de nos jours seulement un téléphone). Les fonctionnalités assurées par les systèmes embarqués sont toujours plus nombreuses, et toujours plus complexes, à l'image du phénomène actuel de convergence [4],
- du délai de mise en vente sur le marché, la concurrence est de plus en plus importante.

Compte tenu de cette compétition et de la rapidité d'évolution des technologies, la conception de ces systèmes embarqués exige une très grande efficacité de leur processus de développement, sans pour autant tolérer de pertes au niveau de la qualité des fonctionnalités, de la fiabilité et des performances.

Dans la plupart des cas, les systèmes embarqués sont réalisés en respectant des contraintes temporelles. Ce type de système peut être désigné par le terme de systèmes temps-réel.

1.1.1.2 Système temps-réel

Un système temps-réel représente un système informatique dont le résultat doit être fourni dans un temps imparti. En effet, ce type de système doit respecter des contraintes temporelles qui sont le plus souvent définies en termes de débit ou de budget de temps. Nous pouvons retrouver ce type de systèmes en tant que contrôleur (ou pilote) de systèmes embarqués. En effet on peut les utiliser dans l'industrie de production par exemple, au travers des systèmes de contrôle de procédé (usines, centrales nucléaires), dans les salles de marché au travers du traitement des données boursières en temps réel, dans l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites), ou encore dans le secteur de la nouvelle économie au travers du besoin, toujours croissant, du traitement et de l'acheminement de l'information (vidéo, données, pilotage à distance, réalité virtuelle, etc). Le bon fonctionnement de tels systèmes est souvent régi non seulement par l'exactitude du résultat qu'il calcule mais aussi par le respect des contraintes de temps. En d'autres termes il doit diffuser le résultat dans les délais imposés, ou avant une échéance. Il est inconcevable d'obtenir des retards ou erreurs dans la diffusion des calculs pour le pilote automatique d'un avion. Mais l'on peut également citer le problème de nombreux joueurs lorsque l'image de leur jeu favori ralentie et se fige pendant un certain temps. Ces exemples illustrent bien le fait qu'il existe des contraintes strictes dont le non respect peut provoquer des conséquences catastrophiques tant au niveau pécunier qu'humain, mais aussi des contraintes souples qui, si elles ne sont pas respectées, entraînent une diminution de la qualité de service. Celle-ci désigne la capacité à fournir un service conforme à des exigences en matière de temps de réponse, de qualité d'image, etc. Ces systèmes temps-réel sont souvent composés de plusieurs tâches (ou actions) que l'on doit ordon-

nancer pour permettre le respect de ces échéances. Pour réaliser cet ordonnancement il est nécessaire de connaître les informations temporelles de chaque tâche. Celles-ci sont composées des durées d'exécution des tâches mais aussi des contraintes de temps avant lesquelles elles doivent être exécutées. Il est cependant difficile d'obtenir les durées d'exécution réelles des actions que ce soit à cause des données traitées ou de la complexité du matériel utilisé. En effet l'exécution des tâches est très difficile à prévoir et ne permet pas de garantir un certain comportement.

Puisque la validité d'une application temps-réel dépend des durées d'exécution, il est très difficile dans ce contexte de concevoir des systèmes corrects. Actuellement, il existe deux approches divergentes dans l'ingénierie de tels systèmes. En effet nous avons vu précédemment que toutes les applications n'ont pas les mêmes besoins concernant le respect des contraintes temporelles.

Temps-réel critique

L'approche temps-réel critique, appelée parfois "temps-réel dur", sert comme son nom l'indique à garantir que les contraintes temporelles soient respectées dans tous les cas de figure. Cette approche est utilisée à chaque fois que la violation d'une contrainte de temps peut remettre en cause l'intégrité du système, et plus particulièrement dès qu'une notion de sécurité est en jeu. L'exemple du système de pilotage automatique d'un avion doit forcément respecter toutes les contraintes de temps qui lui sont données.

Par exemple, pour garantir ces contraintes de temps, il est nécessaire de connaître les durées d'exécution des actions au préalable. Le caractère imprévisible de l'exécution des applications ne permet pas cette connaissance. C'est la raison pour laquelle l'ingénierie de ces systèmes est basée sur une analyse pire-cas des durées d'exécution, ce qui conduit à une surévaluation de la valeur du temps de la tâche [5].

Il est cependant très difficile d'obtenir ces pire-cas pour des applications logicielles. En effet ces pire-cas d'exécution se produisent sous l'effet conjugué :

- du programme lui-même et de son comportement en fonction des données qu'il prend en entrée ;
- d'un certain comportement de l'architecture matérielle sur laquelle s'exécute le programme.

La complexité du logiciel et du matériel rend le calcul exact des durées pire-cas très difficile. Il est donc nécessaire d'utiliser des outils pour approcher ces durées d'exécution pire-cas. Il existe dans ce vaste domaine deux catégories de méthodes d'estimation du pire temps d'exécution d'un programme.

- La première catégorie regroupe les méthodes dites d'*analyses dynamiques* qui consistent à exécuter le programme sur une machine en lui fournissant différents jeux d'entrées pour mesurer la durée de chaque exécution. Le principe de cette méthode est simple mais reste très approximatif. En effet il est nécessaire d'avoir

une connaissance parfaite de la machine pour pouvoir simuler les bons jeux de test. De ce fait, ces derniers ne peuvent couvrir l'ensemble des exécutions possibles. Ces estimations peuvent donc être dépassées et ne constituent qu'une borne inférieure du pire-cas.

- La seconde catégorie regroupe les méthodes d'*analyses statiques*. Elles fournissent un résultat sans exécuter le programme, simplement en examinant sa structure. Cette approche consiste à modéliser le système, logiciel et matériel, afin d'en déduire les durées d'exécution du programme. Contrairement aux méthodes dynamiques, elles fournissent une borne supérieure des durées d'exécution de l'application, ce qui peut être très éloigné des pire-cas réels et de ce fait plus approximatif.

Ces analyses conduisent le plus souvent à surdimensionner les ressources nécessaires à l'exécution du système, que ce soit en termes de puissance de calcul, de consommation mémoire et électrique. C'est le prix à payer pour garantir le respect des contraintes de temps, ce qui est une nécessité absolue en ce qui concerne les systèmes temps-réel critiques.

Temps-réel "souple"

A la différence du temps-réel critique, les approches temps-réel "souples" (ou "molles") considèrent que les contraintes temps-réel peuvent être violées. Ces approches sont utilisées lorsque la violation des contraintes temps-réel ne remet pas en cause l'intégrité du système, et/ou encore lorsque certains dysfonctionnements du système sont tolérés étant donné qu'il s'agit d'un système non critique, n'influant pas sur la sécurité d'autrui. On rencontre de tels systèmes dans les télécommunications ou encore le multimédia. Ces applications s'accommodent du non-respect des contraintes temporelles dans certaines limites au-delà desquelles le système devient inutilisable. Prenons l'exemple d'une application de visioconférence : une erreur d'encodage sur une image survient durant une discussion. Pour rattraper l'erreur, le système a la possibilité de ne pas encoder l'image suivante. Ce phénomène est acceptable s'il est peu fréquent. Laisser la possibilité au système de violer les contraintes temps-réel nécessite des mécanismes additionnels, soit pour réduire le taux de contraintes violées, soit pour éviter que le système ne bogue lorsque ces contraintes ne sont pas respectées. Une application issue des approches temps-réel mou doit respecter les contraintes pour une moyenne de ses exécutions. Un dépassement exceptionnel est toléré. En effet il sera peut-être rattrapé à l'exécution suivante. Même s'il serait préférable de garantir le bon fonctionnement du système dans tous les cas de figure possibles, la mise en place d'une approche temps-réel critique serait trop pénalisante pour les performances et/ou le coût du système.

Ainsi, les approches temps-réel souple s'intéressent le plus souvent à l'optimisation de certains critères tels que la consommation mémoire et électrique, ou encore la vitesse d'exécution. Ces approches tentent de réaliser un compromis idéal entre le taux de contraintes temps-réel violées et l'optimisation des critères de performance du système.

Les deux approches temps-réel critique et temps-réel souple sont par nature opposées. Elles concernent deux communautés différentes, qui utilisent des méthodologies, des outils et des plateformes en général différents. Il est souvent admis que cette séparation reste inévitable, puisque l'utilisation d'analyses pire-cas pour assurer les contraintes temps-réel critiques limite inévitablement l'efficacité de l'application lorsque l'incertitude sur le système et/ou l'environnement est grande. Cette séparation tend à s'accentuer, en particulier pour les systèmes embarqués grand public, pour deux raisons principales :

- L'absence de techniques efficaces d'analyse des comportements pire-cas, tant en termes de durée d'exécution, de consommation mémoire que de consommation électrique.
- La différence entre le comportement moyen et le comportement pire-cas ne fait que s'accroître du fait de l'utilisation de logiciel mais aussi de matériels sophistiqués (technologies de cache, spéculation, ou de *pipeline*). Le matériel récent permet certes de réduire de façon très importante la durée d'exécution moyenne d'une application, mais tend à augmenter la durée d'exécution pire-cas, ou du moins à ne pas réduire autant la durée d'exécution moyenne et pire-cas [6]. La présence d'environnements d'exécution peu prévisibles ne fait que renforcer la différence qui existe entre les analyses basées sur le cas moyen et celles qui sont basées sur le pire-cas.

1.1.2 Application multimédia embarquée

Au début des années 80 le mot *multimédia* désignait les applications qui, grâce à la mémoire du CD et aux capacités de l'ordinateur, pouvaient générer, utiliser ou piloter différents médias simultanément : musique, son, image, vidéo, et interface homme-machine interactive. Les techniques de stockage ont considérablement évoluées dans la fin des années 90, avec le développement de techniques de compression d'image, de son et de vidéo. Ces techniques assurent un compromis idéal entre la qualité d'enregistrement du média et la quantité d'information nécessaire pour son stockage. Cependant les algorithmes utilisés pour réaliser ces compromis sont souvent très coûteux et complexes tant au niveau de l'encodage, de la transmission que de la restitution des données. En effet ces algorithmes sont souvent "intelligents" et permettent d'adapter la qualité du rendu en fonction des données et de la durée d'exécution. Ce qui a pour influence de donner lieux à des durées d'exécution variables dans toute la chaîne d'encodage du flux multimédia. Ce comportement n'est pas viable pour de telles applications étant donnée que le traitement multimédia est directement lié au systèmes temps-réel. En effet une application de type visiophonie doit afficher un nombre d'images par seconde suffisamment élevé pour que la vidéo soit fluide. Il est donc nécessaire de contraindre les algorithmes au niveau de leurs durées d'exécution. Dans les applications multimédia, les contraintes temporelles proviennent aussi bien des exigences de réactivité par rapport à l'utilisateur que des flux multimédia eux-mêmes, comme le son ou la vidéo qui sont par nature temporisés. Il est en effet particulièrement pénible dans une vidéo d'avoir le son et l'image désynchronisés ou encore pire lorsque l'image est saccadée.

Enjeu du contrôle d'application multimédia

De nos jours nous voyons l'émergence de plus en plus de lecteurs multimédia, c'est-à-dire de systèmes embarqués dédiés au décodage de média. Ces systèmes sont notamment contraints par le matériel utilisé et il devient nécessaire de réaliser le logiciel conjointement avec le matériel. Cependant l'approche industrielle se base sur des applications génériques que le développeur adapte à une plateforme cible. Il est donc nécessaire de modifier le code pour le mettre en relation avec la puissance de calcul de la plateforme, mais aussi de faire appel à toutes autres unités de celle-ci permettant d'accélérer le traitement, comme des accélérateurs matériels par exemple. Ce type de réalisation donne naissance à plusieurs problèmes, notamment le traitement "à la main" des contraintes temps-réel qui devient très difficile et qui est donc une source d'erreur permanente. De ce fait, il n'est pas possible de construire des systèmes robustes en un temps raisonnable. Aujourd'hui, près de 80% du temps de développement d'une application embarquée temps-réel est utilisé pour la validation et le déboggage [7]. La plupart des erreurs ne proviennent pas d'une mauvaise implémentation d'un algorithme ou une fonctionnalité particulière, mais plutôt des communications et synchronisations entre les différentes tâches qui sont exécutées en parallèle. Ce type d'erreur est directement dépendant des durées d'exécution réelles de l'application. En particulier, une simulation imprécise quant aux durées d'exécution peut cacher des erreurs qui n'apparaissent que sur le produit final. De plus, il est nécessaire de reprendre complètement le code applicatif à chaque fois qu'une nouvelle plateforme d'exécution est développée. Ainsi, le code applicatif est peu réutilisable, ce qui est très pénalisant pour le temps de mise sur le marché et le coût de développement, surtout dans un contexte de renouvellement permanent des technologies.

La concurrence accrue et l'innovation technologique incessante dans le domaine des systèmes embarqués grand public induisent des temps de mise sur le marché de plus en plus courts. Ainsi le développement de méthodologie et d'outils adaptés à la conception de ce type de logiciel est devenu une nécessité pour les industriels. Pour arriver à concevoir des systèmes à la fois sûrs et performants il est donc essentiel de développer des techniques basées non pas sur des choix statiques mais au contraire sur une adaptation dynamique de l'application à la plateforme en fonction de son comportement réel. Une telle adaptation ne peut être réalisée que si l'application contient suffisamment de libertés de contrôle. Les algorithmes de traitement multimédia, et plus particulièrement d'encodage de flux multimédia, sont généralement basés sur des heuristiques sur lesquelles il est possible d'intervenir sans altérer leur fonctionnalité. Les approches temps-réel souples existantes utilisent le plus souvent des techniques de contrôle. Cependant, celui-ci est réalisé à gros grain sur des paramètres globaux comme le taux d'utilisation du processeur ou encore le nombre d'échéances ratées. Un tel contrôle ne permet donc pas d'assurer le respect des contraintes temps-réel.

1.2 État de l'art

Cette thèse s'intéresse au domaine des techniques de contrôle d'application temps-réel. Celles-ci permettent d'adapter le comportement de l'application afin de respecter des contraintes de qualité de service. Pour se conformer à ces dernières, il est possible de choisir l'ordre d'exécution des actions, ce qui correspond à des problèmes d'ordonnancement, mais il est aussi possible d'intervenir sur des paramètres inhérents au système comme la complexité des algorithmes utilisés ou encore la fréquence du processeur. Dans cet état de l'art, nous définirons premièrement ce qu'est l'ordonnancement. Dans une deuxième partie, nous décrirons les principales techniques d'ordonnancement statique. Puis nous conclurons par des techniques d'ordonnancement dynamique, où le contrôle est fait à l'exécution par rapport à des paramètres de l'application.

1.2.1 Techniques d'ordonnancement et de contrôle

Des techniques de contrôle ont permis d'adapter le comportement des systèmes en choisissant l'ordre d'exécution des actions. On les appelle les techniques d'ordonnancement. Celles-ci ont un champ d'application très large. En effet dès qu'il s'agit d'effectuer un travail global qui pour être réalisé doit être subdivisé en plusieurs tâches et que l'accomplissement de ces tâches est régi par des contraintes qu'elles soient temporelles ou au niveau d'autres ressources, un problème d'ordonnancement est posé. Ainsi l'ordonnancement n'a pas toujours été une activité informatique, on la trouve dans différentes usines, comme les ateliers de montage, où trouver le meilleur moyen en termes de qualité, de coût ou de temps pour fabriquer une pièce est un enjeu considérable. C'est pour répondre à ces problèmes que l'informatique s'est placée comme outil pour calculer les ordonnancements possibles.

Avec l'apparition des systèmes multi-tâches, le domaine d'application s'est considérablement agrandi. Ainsi l'ordonnancement est devenu une activité informatique à part entière. Le problème d'ordonnancement est devenu un problème de gestion des ressources de l'ordinateur, comme le processeur ou la mémoire. Il s'agit maintenant de trouver comment allouer à des instants fixés les ressources nécessaires à une tâche pour permettre le bon fonctionnement du système.

Le problème d'ordonnancement couvre, dans les systèmes informatiques, plusieurs grands domaines comme le parallélisme, les systèmes d'exploitation ou le temps-réel. Nous allons nous intéresser aux systèmes temps-réel dont le bon fonctionnement nécessite le respect de contraintes temporelles.

Depuis l'apparition de tels systèmes des théories ont été élaborées, les travaux de [8] ont réellement lancé le domaine de l'ordonnancement temps-réel.

Terminologie

Avant de présenter quelques techniques d'ordonnancement, il est nécessaire de fixer la terminologie utilisée pour les travaux d'ordonnancement. Nous allons donc décrire les deux entités sur lesquels sont basés ces techniques, c'est-à-dire les tâches et les ressources.

Tâche

Un système temps-réel est composé de plusieurs tâches, notés $T_0 \cdots T_n$. Ces tâches représentent des morceaux de code qui vont s'exécuter sur un processeur. Elles sont caractérisées par des impératifs fonctionnels et temporels qui décrivent son comportement de sa date d'activation à sa date de terminaison, en passant par ses interactions diverses et l'utilisation d'autres ressources.

Chaque tâche est caractérisée par plusieurs informations qui diffèrent selon les systèmes. On peut citer les échéances, notées D, qui correspondent à une contrainte de temps avant laquelle la tâche doit s'exécuter ; les périodes notées P, elles représentent le budget de temps qu'il y a entre deux instances d'une même tâche ; la durée d'exécution C de la tâche.

Ressources

Une ressource concerne souvent un composant matériel du système dont la tâche a besoin pour s'exécuter, comme une mémoire ou un processeur. Nous considérons qu'une ressource ne peut être utilisée que par une seule tâche à la fois, c'est le principe d'exclusion mutuelle. Une ressource est dite préemptable si, alors qu'une tâche l'utilise, cette utilisation peut être momentanément suspendue à tout instant pour être reprise par la suite. On dira d'une ressource qu'elle est non-préemptable si elle doit toujours être utilisée d'une traite.

Exécution d'une tâche

Voici comment un morceau de code s'exécute sur un processeur non préempté [9] : la tâche est activée, puis allouée à un processeur, le contexte initial est sauvegardé et celui de la tâche chargé. Maintenant elle commence vraiment à s'exécuter ; c'est la fin de cette phase qui marque le dernier évènement observable, celui qui permet de mesurer si l'échéance est respectée ou non. Ensuite se passent quelques actions internes à la tâche puis le contexte est déchargé. C'est seulement alors que la tâche termine et libère le processeur.

Cette description marque bien le caractère non-déterministe et incontrôlable de l'exécution d'une tâche.

1.2.2 Techniques d'ordonnancement statiques

On appelle ordonnancement statique, tout ordonnancement dont les choix sont fait avant l'exécution du système, et dont l'ordre n'est pas remis en cause tout au long de

celle-ci. Le problème peut être posé de la manière suivante, trouver un ordre d'exécution tel que toutes les tâches respectent leurs contraintes de temps. On peux dresser une liste non exhaustive des techniques d'ordonnancement répondant à ce problème.

Soient n tâches périodiques T_i, $i \in \{ 1, \ldots, n \}$, de périodes respectives P_i, c'est-à-dire si t_i est la date d'activation d'une des occurrences de T_i, alors la date d'activation de la prochaine occurrence de T_i est $t_i + P_i$. Supposons de plus que pour tout i, la durée d'exécution C_i de la tâche T_i est connue, et que toute occurrence de T_i doit terminer avant l'activation de l'occurrence suivante, c'est-à-dire dans un laps de temps de P_i à partir de son activation. Ainsi, si t_i est la date d'activation d'une occurrence de T_i, alors $D_i = t_i + P_i$ est sa date d'échéance. Il s'agit donc d'un problème temps-réel dur dans lequel les tâches ne doivent pas dépasser leurs échéances. Bien entendu, pour tout i nous avons $C_i \leq P_i$ sinon, le problème n'a pas de solution.

Rate Monotonic, R.M.

La politique *Rate Monotonic Scheduling* [8] est une politique à priorité fixe et procure des résultats intéressants en termes d'utilisation du processeur. On suppose que l'échéance d'une tâche est déterminée par sa période, $P_i = D_i$. Elle consiste à affecter la plus haute priorité à la tâche ayant la plus petite période et ainsi de suite. La priorité d'une tâche est d'autant plus élevée que sa période d'activation est petite. [8] donne aussi une condition sur l'utilisation maximale du processeur :

$$\sum_{1 \leq i \leq n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1), \tag{1.1}$$

est une condition nécessaire pour que le système puisse être ordonnancé, c'est-à-dire que les tâches respectent leurs échéances. La quantité $n(2^{\frac{1}{n}} - 1)$ qui borne l'utilisation du processeur dans (1.1) tend en décroissant vers $ln\ 2 \approx 0\ .\ 693$. Mais cette borne est loin d'être nécessaire. Il a en effet été montré dans [10] que l'utilisation du processeur peut monter jusqu'à 88% en moyenne, en se basant sur une analyse stochastique.

Earliest Deadline First, E.D.F

Cette politique est une extension de l'algorithme d'ordonnancement classique E.D.D. [11] qui ordonnance les tâches en ordre croissant des dates d'échéances et qui minimise le plus grand des retards. Une politique E.D.F. [8] choisit, au moment d'allouer une tâche au processeur, la tâche dont l'échéance est la plus proche de l'instant d'allocation parmi les tâches en attente. Une telle politique nécessite donc l'usage d'un mécanisme de préemption du processeur, supposé instantané dans cette analyse. Dans ce cas, [8] montre que l'analyse EDF est optimale et que l'ensemble des tâches peut être ordonnancé si et seulement si :

$$\sum_{1 \leq i \leq n} \frac{C_i}{P_i} \leq 1\ . \tag{1.2}$$

Least Laxity First, L.L.F.
La dernière politique que je vais citer est une variante de la précédente offrant des résultats moins satisfaisants en termes d'utilisation du processeur. La politique LLF [12] prend en compte les échéances mais aussi les durées d'exécution des tâches. Il ordonnance les tâches en fonction de leur laxité. On appelle laxité la différence entre l'échéance et la durée d'exécution de la tâche.

A partir de ces politiques d'ordonnancement, une problématique importante apparaît. En effet, les durées d'exécution des applications multimédia sont difficilement calculables à l'avance, il est donc difficile de réaliser un ordonnancement en fonction des durées d'exécution réelles de l'application. C'est la raison pour laquelle ces politiques sont basées sur des durées d'exécution pire-cas pour les remplacer. Celles-ci vont mener à une utilisation du processeur très inférieure à celle attendue.

D'autre part, les seuls paramètres sur lesquels il est possible d'intervenir concerne l'ordre d'exécution. Il peut être intéressant de voir des techniques d'ordonnancement permettant d'agir non seulement sur cet ordre mais aussi sur d'autres paramètres comme la fréquence du processeur ou encore le niveau de calcul d'un algorithme. Différentes extensions de R.M. et E.D.F. ont été proposées. Par exemple [13] propose une extension dans le cadre de l'optimisation de la consommation d'énergie. L'approche repose sur le fait qu'en cas de sous charge du processeur, il est possible de réduire sa fréquence et sa tension, ce qui réduit sa consommation électrique.

1.2.3 Techniques d'ordonnancement dynamique sous incertitude

Nous l'avons vu, les techniques d'ordonnancement statique sont intéressantes lorsque les durées d'exécution des tâches sont connues à l'avance. Il est évident que dans le cas contraire elles ne sont pas performantes, étant donné qu'elles ne peuvent pas remettre en question l'ordre d'exécution du système pour l'adapter au comportement réel de celui-ci. Les approches dynamiques ou adaptatives peuvent quant à elles se confronter aux incertitudes sur les durées d'exécution ou périodes d'activation inconnues pour ne citer qu'elles. Il est souvent possible de caractériser les valeurs incertaines d'un modèle par un encadrement ou encore par un modèle stochastique [14]. Ces informations permettent à l'ordonnanceur de réaliser des prévisions sur l'avenir, augmentant ainsi la pertinence de ses décisions.

Slack Scheduling

Les techniques de *slack scheduling* étendent les politiques d'ordonnancement statique en se basant sur le constat suivant. Puisque les durées d'exécution pire-cas sont souvent sur-estimées par rapport aux durées d'exécution réelles, une partie du budget de temps pré-allouée statiquement est disponible dynamiquement au moment où la tâche termine. Le *slack scheduling* se fait donc en deux temps : tout d'abord, il convient d'évaluer le

budget de temps libéré par les tâches qui ont terminé avant leur échéance, et ensuite il faut utiliser ce budget de temps. [15] considère le même modèle de tâches que celui de [8], augmenté d'un ensemble de tâches apériodiques considérées comme étant "temps-réel souples". La politique d'ordonnancement R.M. est modifiée de façon à calculer le budget de temps qui peut être alloué aux tâches apériodiques sans remettre en cause le respect des échéances des tâches périodiques. [15] explique également comment prendre en compte dynamiquement le budget de temps qui est pré-alloué statiquement par la politique d'ordonnancement R.M., et qui est rendue disponible lorsque les tâches périodiques terminent avant leur échéance. L'algorithme proposé permet de minimiser le temps de réponse des tâches apériodiques. [16] propose différentes extensions comme le partage de ressources (autres que le processeur) entre les tâches ou encore des échéances arbitraires. Dans [17], Thuel et Lehoczky considèrent un problème d'ordonnancement à priorités fixes d'un ensemble de tâches périodiques et apériodiques, toutes contraintes par des échéances strictes. L'idée est de n'accepter que les tâches apériodiques dont on a l'assurance qu'elles respecteront leur échéance, les autres étant rejetées par l'ordonnanceur. L'algorithme proposé permet de prendre en compte les durées d'exécution réelles des tâches.

Gain Time

Les techniques de *gain time* s'intéressent à la durée d'exécution du code qui reste à exécuter. Le but est de raffiner la durée d'exécution pire-cas d'une tâche en fonction de la connaissance de ses données d'entrée ainsi que de son état courant. A partir de la durée d'exécution pire-cas du code qui reste à exécuter et de la durée d'exécution réelle du code qui a déjà été exécutée, les choix d'ordonnancement peuvent être anticipés [18]. Pour que l'analyse de *gain time* soit possible, il est nécessaire de travailler à un niveau de granularité plus fin que celui de la tâche système, c'est-à-dire en s'immisçant directement dans le code applicatif de la tâche. Pour ce faire, la durée d'exécution pire-cas de la tâche est ré-évaluée au cours de son exécution, sur des points de programme particuliers appelés *gain points*. Le choix de ces *gain points* conditionne en grande partie l'efficacité du *gain time* dans la mesure où ralentir le système par une sur-instrumentation ne ferait que diminuer la qualité de service. L'objectif est donc de choisir des points du programme pertinents quant à la mise à jour de la durée d'exécution pire-cas.

[19] montre qu'il est intéressant d'insérer un *gain time* à chaque fois qu'une partie non négligeable d'incertitude sur la durée d'exécution pire-cas peut être levée. Une autre optimisation concerne l'ordre d'exécution du code. En effet, ordonnancer en premier les parties du code les plus incertaines, lorsque cela est possible, assure que les durées d'exécution pire-cas soient les plus précises possibles le plus tôt possible [20].

Contrôle dynamique de la fréquence d'horloge du processeur

Lorsque les durées d'exécution des tâches sont incertaines, la possibilité de contrôler d'autres paramètres que l'ordre d'exécution devient particulièrement intéressant pour donner une plus grande flexibilité au système en augmentant sa capacité d'adaptation. [21] propose par exemple d'adapter dynamiquement la fréquence d'horloge du processeur, sur la base d'un modèle de tâches périodiques et d'un ordonnancement de type R.M. ou E.D.F. Deux algorithmes sont proposés. Dans le premier, appelé *Cycle-conserving*, la fréquence d'horloge initiale est issue d'un calcul statique, dans lequel les durées d'exécution sont supposées au pire-cas. Si l'exécution d'une tâche mène à une économie au niveau du budget de temps, c'est-à-dire si une tâche finit avant sa durée d'exécution pire-cas, cette économie est intégrée dynamiquement dans le calcul de la fréquence courante. Ainsi le budget économisé est utilisé non pas pour ordonnancer les tâches mais pour baisser la fréquence du processeur.

Le second algorithme de contrôle dynamique de fréquence est appelé *Look-ahead*. Il propose une heuristique qui part du principe que les tâches terminent presque toujours bien avant leur durée d'exécution pire-cas. La fréquence initiale est calculée de telle sorte qu'il soit toujours possible de respecter les échéances, en l'augmentant si besoin est par la suite. Cette idée est également à la base du calcul dynamique de la fréquence d'horloge. A la différence de l'algorithme *Cycle-conserving*, la fréquence initiale ne peut pas être maintenue si les durées d'exécution réelles sont toutes égales aux durées d'exécution pire-cas. En effet, dans ce cas, l'ordonnanceur sera forcé d'augmenter la fréquence d'horloge dans la mesure où il démarre avec une fréquence volontairement basse. L'ordonnanceur *Look-ahead* anticipe ainsi le fait que les durées d'exécution réelles sont presque toujours inférieures aux durées d'exécution pire-cas. Le système reste néanmoins sûr, les échéances sont toujours respectées, puisque l'ordonnanceur a toujours la possibilité d'augmenter la fréquence si les durées d'exécution réelles sont trop élevées. En pratique, l'ordonnanceur *Look-ahead* se montre plus performant que l'ordonnanceur *Cycle-conserving*, par une réduction plus agressive de la fréquence d'horloge.

Imprecise computation

Les approches dites d'*imprecise computation* [22, 23, 24] étendent elles aussi les techniques d'ordonnancement temps-réel dur de type R.M. ou E.D.F. [8]. Même dans un système de type temps-réel dur, il est possible que certains calculs effectués par certaines tâches puissent être dégradés sans remettre en cause le bon fonctionnement du système. Pour ce faire, la durée d'exécution de chaque tâche périodique T_i est décomposée en deux parties : la première donne la durée d'exécution minimale C_i^m de T_i pour obtenir un résultat acceptable. Le système est considéré comme ordonnançable si les tâches respectent leur échéance, et si la durée d'exécution de chaque tâche T_i est supérieure ou égale à C_i^m. Lorsque la durée d'exécution d'une instance d'une tâche périodique est inférieure à sa durée d'exécution maximale $C_i \geq C_i^m$, le résultat produit par T_i est considéré comme

imprécis. Cette imprécision est mesurée par ϵ, qui est une fonction décroissante de la durée d'exécution de T_i, nulle en C_i. Deux types de tâches sont proposées dans [25] : pour les tâches de type N, seule l'erreur moyenne est pertinente ; pour les tâches de type C, l'erreur produite par les différentes instances d'une tâche est accumulée.

Temps-réel souple

Lorsqu'il existe une part d'incertitude sur les durées d'exécution dans le système, les analyses de type pire-cas des approches temps-réel critique conduisent à de mauvaises performances quant à l'utilisation du processeur. Pour cette raison, de nombreux travaux de type "temps-réel souple" ont été menés pour traiter les problèmes d'ordonnancement sous incertitude.

Dans [26], Stankovic et al. partent de l'hypothèse que les tâches du système sont paramétrées par un niveau de qualité entier (*QoS level*). Certaines applications — multimédia [27, 28], 3D [29], WEB, télécommunication — supportent une réduction progressive de la précision de leurs calculs (*graceful degradation*). Cette dernière s'accompagne d'une réduction de la consommation en ressources de calcul et/ou de mémoire. Il est important de noter que la fonctionnalité de l'application n'est pas affectée par cette dégradation, seule la qualité de service diminue. Les niveaux de qualité du modèle proposé dans [26] correspondent donc à différentes qualités de service possibles. Ils peuvent aussi contrôler l'admission d'une tâche : dans ce cas le niveau de qualité minimal correspond à la non-admission de la tâche. L'approche de Stankovic et al. constitue un cadre très général de conception des systèmes temps-réel souples, basée sur un ordonnanceur et un Gestionnaire de Qualité qui visent à optimiser les performances globales du système. Cette optimisation est faite par le biais de l'observation de paramètres globaux comme le nombre d'échéances ratées. Le Gestionnaire de Qualité de service proposé détermine le niveau de qualité des tâches à l'aide de techniques de contrôle classiques comme les contrôleurs PID [30].

Dans [31], Wüst et al. s'intéressent au contrôle de la qualité de service d'un décodeur vidéo. Ils partent du principe que les applications multimédia, comme le décodage vidéo, ont des durées d'exécution très variables, fortement dépendantes des données d'entrée. Pour restituer correctement une séquence vidéo, les images qui la constituent (ou *frames*) doivent être affichées à des dates précises. Par exemple, pour une vidéo encodée à 25 images par seconde, une image doit être produite toutes les 40 ms. Dans les situations de surcharge du processeur, le décodeur ne peut plus satisfaire les contraintes temps-réel. La pratique courante est d'utiliser un tampon mémoire en entrée du processus de décodage pour absorber les surcharges temporaires du processeur. Lorsque le tampon est plein, certaines images ne sont pas décodées (*frame skip* [32]) afin de continuer à satisfaire les contraintes de temps, au détriment de qualité de la vidéo. [31] propose de réduire progressivement la qualité des algorithmes de décodage afin d'éviter que le taux de *frame skips* ne devienne trop élevé lorsque le processeur est surchargé. Le décodeur

comporte quatre niveaux de qualité, et [31] présente plusieurs algorithmes de contrôle de ces niveaux de qualité basés sur une modélisation en termes de chaînes de Markov et des techniques d'apprentissage. Dans [33], Steffens et al. montrent comment intégrer cette technique au sein d'un système complet composé de plusieurs tâches.

1.3 Objectifs et Plan

Dans cette section, nous présenterons l'approche de contrôle d'application à partir de laquelle nous allons travailler. Ensuite, nous fixerons les objectifs du travail de thèse qui ont été réalisés. Pour finir nous détaillerons le plan de ce manuscrit.

1.3.1 Notre Approche

Ce travail de thèse se situe dans la continuité d'un travail préalablement réalisé [34]. Dans celui-ci, nous proposons une approche où un contrôleur est directement embarqué dans l'application. Celui-ci assure le respect à la fois des propriétés de type temps-réel critique, c'est-à-dire le respect des échéances, mais aussi des propriétés de type temps-réel mou, ce qui correspond à une utilisation optimale des ressources. Plus clairement, il s'agit de réunir les propriétés des deux mondes du temps-réel. Il est donc nécessaire de développer des techniques adaptatives permettant de les satisfaire.

Nous considérons une application logicielle initiale qui traite, de manière cyclique, des actions sur un flux de données d'entrée. Cette application est décrite par un graphe de précédence, qui modélise les dépendances entre les tâches (ou actions), autrement dit les traitements décrits par des fonctions (en C). A partir de ce graphe de précédence, il est possible d'extraire tous les ordonnancements possibles de l'application. Chaque action de l'application est paramétrée par un niveau de qualité. Ce dernier représente le niveau de précision auquel le traitement va être effectué. On peut considérer que le niveau de qualité correspond à l'algorithme qui sera utilisé pour le traitement. Ainsi ce choix de qualité influera sur les durées d'exécution des actions.

Nous considérons aussi une plateforme mono-processeur que l'on supposera "nue", c'est-à-dire que seule notre application s'exécutera dessus. Nous supposerons que cette plateforme nous donne accès à une horloge temps-réel suffisamment précise pour pouvoir récupérer le temps de chaque action avec un surcoût négligeable. Par exemple, dans la partie expérimentale nous utiliserons le compteur de cycle de la plateforme STm8010 qui remplit ces conditions. Il est donc non seulement possible de récupérer le temps réel de l'application pour réaliser le contrôle, mais aussi de construire un modèle de temps précis. Ce modèle de temps est composé des durées d'exécution moyennes et pire-cas pour chaque action. Pour les obtenir nous pouvons utiliser soit des techniques d'analyse statiques, soit des techniques d'observation des traces d'exécution réelles. Nous supposons que les actions sont atomiques et que chacune possède une échéance D.

Un outil permet de générer automatiquement le code du contrôleur d'application en C à partir du graphe de précédence, des durées d'exécution moyennes C^{av} et pire-cas C^{wc} et des échéances D (cf. figure 1.1). Il s'agit maintenant de générer des applications contrôlées. Cette étape sera effectuée par le compilateur qui instrumentera le code de l'application par des appels aux contrôleur.

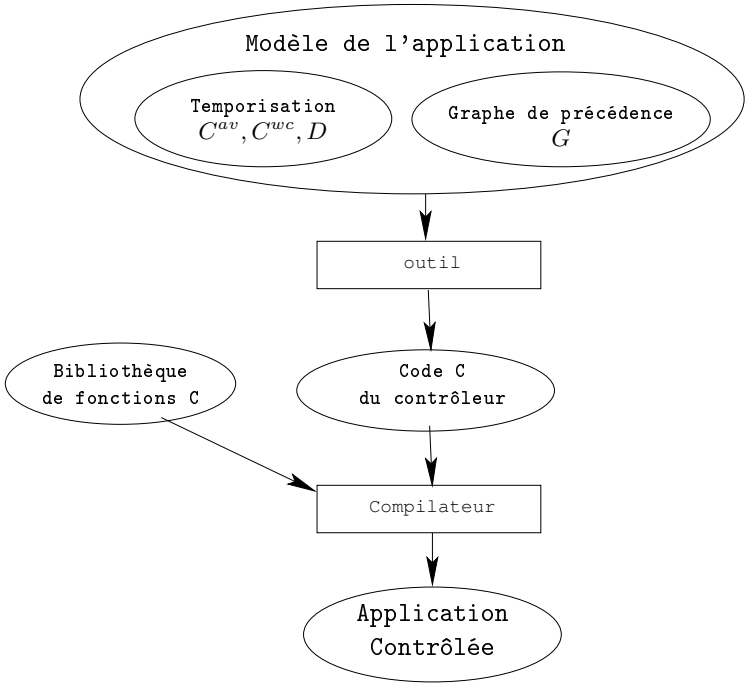


Figure 1.1 — Génération de l'application contrôlée

Maintenant que nous avons brièvement décrit la technique de génération d'application contrôlée, nous allons présenter la technique de contrôle proposée dans [35]

Le contrôleur est associé à une technique de contrôle nommée politique de gestion de qualité. L'objectif de celle-ci est d'adapter le comportement du système en ajustant le paramètre de niveau de qualité des actions. Ainsi il pourra assurer les exigences de qualité de services de l'application. Ces exigences sont de trois types :

- **Sûreté**, qui correspond au respect des contraintes temps-réel. Cette propriété est directement issue du monde du temps-réel dur.
- **Optimalité**, c'est le critère permettant l'utilisation optimale des ressources. Il sera, dans notre cas, associé à l'utilisation du budget de temps. En maximisant son utilisation, nous pourrons activer les niveaux de qualité les plus élevés, ce qui correspondra à un meilleur traitement du flux de données.
- **Régularité**, en effet il s'agit d'obtenir une régularité dans le choix des niveaux de qualité tout au long de l'application. Il a été montré que cette régularité dans le choix des niveaux de qualité était importante dans la qualité finale, par exemple d'une séquence vidéo. En effet la perception de l'utilisateur est influencée non seule-

ment par la qualité individuelle de chaque image qui compose la séquence mais aussi par les fluctuations de celle-ci tout comme de sa qualité minimale [36, 37].

L'application contrôlée peut être considérée comme la composition entre l'application logicielle initiale et le contrôleur (cf. figure 1.2). Le contrôleur surveille la progression d'un cycle de l'application et choisit la prochaine action à exécuter et son niveau de qualité. Ce contrôle est réalisé à grain fin, c'est-à-dire pour chaque action [38].

Le contrôleur est composé de deux entités, un Gestionnaire de Qualité et un Ordonnanceur. Le premier effectue un choix sur les niveaux de qualité. Ce choix est guidé par une politique de gestion de qualité, qui est une contrainte de la forme $t_p \geq t$ où t est le temps réel de l'application et t_p est une fonction donnant une estimation du temps actuel dépendant de la politique de gestion de qualité. A travers cette contrainte, les critères de sûreté et d'optimalité sont respectées. En effet la sûreté signifie qu'aucune échéance ne sera manquée, ce qui implique que $t_p \geq t$. La propriété d'optimalité est quant à elle satisfaite quand la différence $t_p - t$ est minimale. Plus intuitivement, l'utilisation du budget de temps restant pour terminer les actions sera maximale. Ainsi on permettra à l'application d'obtenir un meilleur niveau de qualité sans violer les échéances. Le Gestionnaire de Qualité est utilisé en parallèle d'un Ordonnanceur qui fournit pour chaque niveau de qualité un ordonnancement optimal, c'est-à-dire un ordonnancement maximisant $t_p \geq t$.

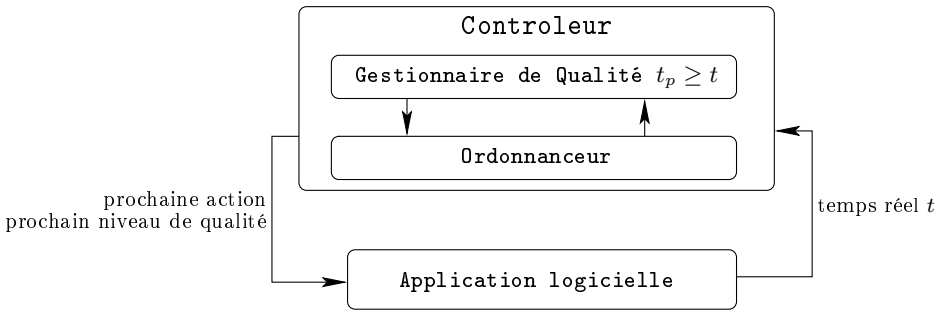


Figure 1.2 — Architecture du contrôleur

De plus cette approche diffère des approches précédemment citées dans la mise en place d'un contrôle à grain fin du comportement de l'application.

1.3.2 Contributions du travail

Dans ce travail de thèse nous sommes parti de la technique de contrôle décrite précédemment (cf. section 1.3.1) pour améliorer et étendre les résultats dans deux principales directions. Nous proposons premièrement une technique de gestion de qualité symbolique, et étudions ensuite une technique basée sur un modèle de temps stochastique. Les contributions principales de l'approche symbolique sont les suivantes.

– Elle définit et étudie les diagrammes des vitesses, une représentation graphique de l'espace d'état de l'application contrôlée. Un état est défini par un point dans un espace à deux dimensions. Une dimension représente le temps actuel alors que l'autre

dimension représente un temps dit virtuel. Ce temps virtuel représente l'avancée dans les actions de l'application. La pente d'un vecteur dans cette représentation détermine la "vitesse" de l'application. C'est donc un moyen de comparaison entre la vitesse actuelle et la vitesse virtuelle. Dans le diagramme des vitesses, la diagonale représentent le vecteur où la vitesse actuelle est égale à la vitesse virtuelle. En conséquence, les points situés sur cette diagonale coïncident avec le comportement optimal de l'application. Nous pouvons aussi dire que les points en dessous de celle-ci représentent des états où l'application est en retard et inversement les points au dessus de celle-ci sont des états en avance. Ainsi nous allons voir que le Gestionnaire de Qualité va s'efforcer de suivre cette diagonale. En d'autres termes au moment de contrôler un état du système, si celui-ci est au dessus (respectivement en dessous) de la diagonale il va augmenter (respectivement diminuer) la vitesse d'exécution de l'application en choisissant une qualité plus faible (respectivement élevée).

- Elle introduit, pour un état de l'application donné et une qualité q, deux sortes de vitesses. La vitesse idéale caractérise l'évolution estimée si toutes les actions sont exécutées avec le niveau de qualité q. La vitesse optimale caractérise, quant à elle, l'évolution optimale du système, c'est-à-dire l'évolution respectant les contraintes de temps mais aussi ayant une utilisation maximale du budget de temps. Nous montrerons que la politique de gestion de qualité définit dans [38, 35] est satisfaite si et seulement si le niveau de qualité choisi pour un état est tel que la vitesse idéale est supérieure à la vitesse optimale.
- Elle montre que les diagrammes des vitesses donnent naissance à une politique de gestion de qualité symbolique. Pour une échéance donnée, il est possible de spécifier l'ensemble des états pour lesquels le Gestionnaire de Qualité choisit une qualité constante q. Ces états forment une région définie par un ensemble d'inégalités qui impliquent le temps actuel, les durées d'exécution moyennes mais aussi les durées d'exécution pire-cas des actions. La connaissance de telles régions permet une implémentation plus efficace des politiques de gestion de qualité. Il est d'autre part possible de réaliser une implémentation beaucoup plus efficace en ayant la connaissance des régions où le Gestionnaire de Qualité choisit un niveau de qualité constant q pour les r prochaines actions. Ces régions peuvent caractériser l'ensemble des états où le Gestionnaire de Qualité peut être relâché.

En ce qui concerne les principales contributions de l'approche stochastique nous pouvons citer les suivantes.

- Elle définit une approche basée sur des fonctions de distributions de probabilités des actions. A partir de ces distributions, il est possible de récupérer tous les comportements possibles de chaque action. Ces distributions nous permettent d'obtenir un modèle de temps d'entrée plus riche. L'utilisation d'un tel modèle de temps va permettre de créer une politique de gestion de qualité basée sur les probabilités.
- Elle caractérise le seuil de tolérance, c'est-à-dire une valeur qui permet de générer les durées d'exécution pire-cas des actions. Ce seuil de tolérance représente la probabilité de la durée d'exécution réelle d'une action de dépasser la durée d'exécution

pire-cas. Le seuil de tolérance permet de caractériser la souplesse de l'échéance. En effet plus cette valeur est basse, plus l'échéance sera considérée comme dure,c'est-à-dire qu'il ne sera pas possible de dépasser l'échéance. Au contraire plus elle est élevée, plus la probabilité de dépasser les contraintes de temps est forte. Ainsi le programmeur va pouvoir définir le comportement de son application.

– Elle permet l'étude à priori du comportement de l'application en fonction des données d'entrée. En effet, nous proposons un algorithme permettant de générer la distribution de probabilité de l'application pour la date de terminaison de sa dernière action. A partir de cette distribution l'approche stochastique propose une estimation des performances de l'application contrôlée. Il est en effet possible d'étudier le taux d'utilisation du budget de temps mais aussi le taux de violation des échéances. A partir de ces estimations, le programmeur va pouvoir adapter le seuil de tolérance pour obtenir les taux qu'il souhaiterait que son application suive. En quelque sorte cette approche permet de satisfaire le monde du temps réel mou en ayant une utilisation optimale des ressources, mais elle apporte une amélioration de ce monde en ayant un contrôle sur le nombre de contraintes de temps que l'application peut violer. Nous pouvons donc nommer cette approche stochastique comme étant une approche du *temps-réel mou maîtrisé*.

1.3.3 Plan de ce document

Ce manuscrit est organisé de la façon suivante.

Au chapitre 2, nous présentons la technique de contrôle sur laquelle ce travail de thèse s'est basé ainsi que les politiques de gestion de qualité permettant de résoudre le problème de contrôle sous incertitude. Dans un premier temps, nous proposerons une étude incrémentale du problème de contrôle de qualité de service. En effet nous partirons d'un problème simple basé sur des durées d'exécution connues pour finir par présenter le problème sous incertitude. Nous introduirons aussi les notations importantes qui seront utilisées tout au long de ce manuscrit. Dans un deuxième temps, nous présenterons les politiques de gestion de qualité permettant de résoudre le problème de contrôle sous incertitude.

Au chapitre 3, nous évoquerons la première contribution de ce travail de thèse. Elle sera organisée en trois parties. Tout d'abord, nous proposerons une représentation graphique permettant de comprendre le comportement d'une application contrôlée. Elle sera nommée diagramme des vitesses. Ensuite, nous présenterons une méthode symbolique basée sur ce diagramme augmenté de région de qualité. Enfin nous parlerons de l'apport principal de cette contribution qui est le relâchement du contrôleur. Ce qui permettra la réduction du surcoût dû aux appels à ce contrôleur.

Au chapitre 4, nous présentons la deuxième contribution de ce travail de thèse. Elle consiste en la définition d'une nouvelle technique de contrôle basée sur des distributions de probabilité des temps d'exécution. Nous énoncerons tout d'abord les motivations d'une telle amélioration, en donnant quelques situations où les politiques de contrôle ne réagissent pas de la manière voulue. Ensuite nous présenterons en détail la technique de contrôle. Une troisième partie permettra de réaliser une analyse de performances basée sur les distributions de probabilité. Pour conclure nous proposerons une amélioration de l'approche symbolique toujours basée sur les distributions de probabilité.

Au chapitre 5, nous proposons des résultats expérimentaux confirmant les résultats théoriques obtenus. Nous présenterons tout d'abord l'environnement d'exécution, c'est-à-dire la plateforme d'expérimentation mais aussi l'application cible. Ensuite nous présenterons l'outil permettant de générer l'application contrôlée. Enfin nous proposerons plusieurs résultats expérimentaux divisés en trois parties présentant les applications pratiques des trois chapitres précédents. Premièrement nous montrerons pourquoi la technique de contrôle est intéressante pour les applications multimédia ainsi que des résultats sur les politiques de gestion de qualité présentées. La deuxième série d'expériences proposent des résultats sur l'approche symbolique. Puis nous finirons par traiter les expérimentations sur l'approche stochastique et l'analyse de performances.

Nous conclurons ce manuscrit par un travail de synthèse et un récapitulatif des contributions apportées. Ainsi nous pourrons nous attacher à de nouvelles perspectives avec pour certaines d'entre elles des idées permettant de les réaliser.

CHAPITRE 2

Contrôle de qualité de service pour la sûreté et l'optimalité

Ce premier chapitre va définir le point de départ de ce travail de thèse. Celui-ci est basé sur des recherches préalablement réalisées sur le contrôle de qualité de service pour garantir des propriétés de sûreté et d'optimalité. Ce chapitre reprend essentiellement les articles [38, 35]. Dans une première partie nous définirons le problème autour duquel se sont orientés ces travaux, pour ensuite donner les solutions qui ont été proposées pour le résoudre.

Les problèmes de contrôle de qualité de service que nous cherchons à résoudre peuvent être définis par la recherche d'ordonnancement et le contrôle de l'exécution d'une application sur une machine mono-processeur. Ce chapitre va permettre de le définir de manière incrémentale. L'ordonnancement consiste à trouver un ordre d'exécution d'un ensemble de tâches (ou actions) en respectant leurs contraintes de précédence et temporelles, de type échéance. C'est un problème simple et connu qui possède déjà un bon nombre de solutions (cf. section 1.2). Nous allons l'étendre en ajoutant un paramètre de qualité associé aux actions. Dans cette extension nous considérerons que les durées d'exécution des actions sont connues. Cette première partie donnera les notations importantes qui seront utilisées tout au long de ce manuscrit. Elle permettra aussi de donner les bases du problème sous incertitude.

En effet une deuxième partie définira une extension du problème dans laquelle les durées d'exécution ne sont pas connues. Cela se rapproche plus du monde des applications multimédia réelles, où la prédictabilité du comportement n'est pas possible. Ces durées d'exécution seront bornées par des valeurs pire-cas, et nous considérerons qu'il est possible de connaître les durées d'exécution réelles une fois l'action exécutée. Cela correspond donc à un problème d'ordonnancement paramétrée dynamique, ou de contrôle de qualité de service. Dans le cadre des applications multimédia embarquées, il faut réaliser un ordonnancement répondant à trois propriétés :

- une propriété relative à la sûreté, comme le respect des échéances ;
- une propriété relative à l'optimalité, qui est l'exploitation optimale des ressources dans le but de maximiser la qualité de service ;
- une propriété propre aux applications multimédia qui est d'avoir une certaine régularité dans le choix des qualités.

Une autre partie sera consacrée à l'énoncé de la solution qui a été proposée pour résoudre ce problème. Nous la nommerons politique de contrôle *mixte*. Nous définirons cette

politique là aussi de manière incrémentale. En effet nous commencerons par proposer une politique de contrôle assurant uniquement la sûreté. Cette politique est uniquement basée sur les durées d'exécution pire-cas et possède donc des limitations en ce qui concerne l'optimalité. Ensuite nous proposerons une politique mélangeant les durées d'exécution pire-cas avec une estimation des durées d'exécution moyennes. Cette politique va permettre de se rapprocher de l'optimalité et de garantir la sûreté. Mais nous verrons les limitations qu'elle peut avoir en ce qui concerne la régularité des choix de niveau de qualité. Nous finirons par la présentation de la politique mixte qui réalise un mélange des comportements pire-cas et moyens beaucoup plus fins et qui permet ainsi de répondre convenablement au problème de contrôle de qualité de service que nous avons défini.

2.1 Contrôle de qualité de service

2.1.1 Problème de contrôle de qualité pour des durées d'exécution connues

Nous présentons ici un problème d'ordonnancement sur une machine mono-processeur. Il s'agit d'ordonnancer un ensemble fini A de tâches (ou *actions*), dont les durées d'exécution sont connues et paramétrées par des niveaux de qualité Q. Ces durées d'exécution sont données par la fonction $C : A \times Q \to \mathbb{R}^+$. L'ordre d'exécution des actions est contraint par un graphe de précédence G, et des dates d'échéance absolues données par la fonction $D : A \to \mathbb{R}^+ \cup \{ +\infty \}$. Résoudre le problème revient à trouver un ordre d'exécution mais aussi une affectation de qualité qui respecte les contraintes de précédence et tel que pour toute action a sa date de terminaison soit inférieure à son échéance $D(a)$.

2.1.1.1 Le problème

Un graphe de précédence est utilisé pour décrire le comportement fonctionnel de l'application logicielle. Il modélise les dépendances entre ses actions (fonctions en C), à partir desquelles il est possible d'extraire tous les ordonnancements possibles.

Définition 2.1 (Graphe de précédence):
Un ***graphe de précédence*** *est un couple* $G = (A, \prec)$ *où* A *représente l'ensemble des* actions *et* $\prec \subseteq A \times A$ *est un ordre partiel sur* A.

(sémantique) *Le graphe de précédence* $G = (A, \prec)$ *définit un* ***système de transition*** $(S, A, \longrightarrow)$ *où* S *est un ensemble d'****états*** *et* $\longrightarrow \subseteq S \times A \times S$ *est une* ***relation de transition étiquetée*** *définie par :*

- *un état* $s_i \subseteq A$ *est un ensemble fermé en arrière d'actions, c'est-à-dire pour tout* $a_1 \in s_i$, $a_2 \prec a_1 \Rightarrow a_2 \in s_i$
- *pour deux états* s_i *et* s_j, *nous avons* $s_i \xrightarrow{a_n} s_j$ *si* $s_i = \{ a_1, \ldots, a_{n-1} \}$ *et* $s_j = \{ a_1, \ldots, a_{n-1}, a_n \}$.

Une séquence d'action a_1 .. a_n est un ***ordonnancement*** *de G si $n = |A|$ et si il existe des états s_0, ..., s_n tels que $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$. Notons que si $n = |A|$ nous avons $s_0 = \emptyset$ et $s_n = A$.*

Nous représentons par $\Sigma(G)$ l'ensemble des ordonnancements de G. Soit un état $s_{i-1} = \{ a_1, \ldots, a_{i-1} \}$ de G, une séquence d'actions a_i .. a_n est un ordonnancement à partir de l'état s_{i-1} si il existe des états s_i, ..., s_n tels que $n = |A|$ et $s_{i-1} \xrightarrow{a_i} s_i \xrightarrow{a_{i+1}} \ldots \xrightarrow{a_n} s_n$. Nous représentons par $\Sigma(G, s_{i-1})$ l'ensemble des ordonnancements à partir de l'état s_{i-1}. Notons que $\Sigma(G, s_0) = \Sigma(G)$.

Soient $G = (A, \prec)$ un graphe de précédence et A' un sous-ensemble d'actions tel que $A' \subseteq A$, nous définissons la restriction de G par A' par $G/A' = (A', \prec \cap (A' \times A'))$.

Exemple 2.1: *Considérons le graphe de précédence $G = (A, \prec)$ avec cinq actions $A =$ { `Quant`, `IQuant`, `IntraP`, `IDCT`, `Coding` } (cf. Figure 2.1). Ce graphe de précédence est un fragment du modèle de l'encodeur vidéo présenté dans le chapitre 5.*

La relation $\prec$ est une fermeture transitive de la relation $\rightsquigarrow =$ {(`Quant`, `IQuant`), (`Quant`, `IntraP`), (`IntraP`, `Coding`), (`IQuant`, `IDCT`)} représentée sur la figure 2.1.

Puisque $s =$ { `Quant`, `IntraP`, `Coding` } est un ensemble fermé en arrière d'actions, c'est un état de G et la séquence `IQuant IDCT` *est le seul ordonnancement possible à partir de cet état. La séquence d'actions* `Quant IntraP Coding IQuant IDCT` *est donc un ordonnancement de G.*

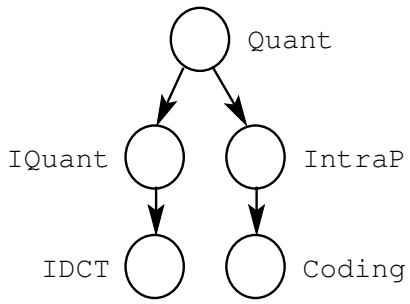


Figure 2.1 — Exemple de graphe de précédence

Un système permet de modéliser une application logicielle s'exécutant sur une plateforme. Ce système est représenté par un graphe de précédence définissant l'ensemble d'actions dont il est composé. Les durées d'exécution de celles-ci sont connues et paramétrées par des niveaux de qualité. Pour finir chaque action est contrainte par une échéance.

Définition 2.2 (Système):
Un **système** *est un n-uplet $SY = (G, Q, C, D)$ où :*

- *G est un graphe de précédence.*

- $Q = [q_{min}, q_{max}]$ *est un intervalle fini d'entier correspondant aux* ***niveaux de qualité****.*
- $C : A \times Q \to \mathbb{R}^+$ *($\mathbb{R}^+$ représente l'ensemble des réels positifs) est une fonction qui donne la* ***durée d'exécution*** $C(a, q)$ *d'une action a pour un niveau de qualité q. Nous supposons que, pour tout $a \in A$, $q \mapsto C(a, q)$ est une fonction croissante.*
- $D : A \to \mathbb{R}^+ \cup \{ +\infty \}$ *est une fonction qui pour chaque action a donne son* ***échéance*** $D(a)$.

(sémantique) *Le système SY définit un* ***système de transition*** $(S \times \mathbb{R}^+, A \times Q, \longrightarrow)$ *tel que :*

- *les états sont donnés par des couples (s_i, t_i) où s_i est un état de G et $t_i \in \mathbb{R}^+$ est une valeur du temps ; nous prenons $t_0 = 0$ pour $s_0 = \emptyset$;*
- *pour deux états (s_i, t_i) et (s_j, t_j), une action a et un niveau de qualité q, nous avons $(s_i, t_i) \xrightarrow{a,q} (s_j, t_j)$ si $s_i \xrightarrow{a} s_j$ dans G et $t_j - t_i = C(a, q)$.*

Le comportement de ces systèmes peut varier selon deux points, l'ordonnancement des actions mais aussi le choix des niveaux de qualité. Pour restreindre ces comportements, il est nécessaire d'utiliser des contrôleurs permettant de gérer le comportement du système tout en respectant des propriétés sur les niveaux de qualité.

Définition 2.3 (Contrôleur):
Un ***contrôleur*** *d'un système $SY = (G, Q, C, D)$ est une fonction $\Gamma : S \times \mathbb{R}^+ \to A \times Q$ qui pour chaque état (s_i, t_i), donne une action a_{i+1} et son niveau de qualité q_{i+1} tel qu'il existe (s_i, t_i) et $(s_{i-1}, t_{i-1}) \xrightarrow{a_i, q_i} (s_i, t_i)$.*

$SY||\Gamma$ représente le ***système contrôlé*** *obtenu comme la composition du système SY et du contrôleur Γ. Il possède une séquence d'exécution simple $\{ (s_{i-1}, t_{i-1}) \xrightarrow{a_i, q_i} (s_i, t_i) \}_{1 \leq i \leq |A|}$ tel que $(a_i, q_i) = \Gamma(s_{i-1}, t_{i-1})$.*

Pour un système $SY = (G, Q, C, D)$, une **affectation de qualité** est une fonction $\theta : A \to Q$ qui pour une action a donne sa qualité q. Un contrôleur Γ de SY calcule un ordonnancement $a_1 \,..\, a_n$ et une affectation de qualité θ tels que $SY||\Gamma$ possède une séquence d'exécution : $\{ (s_{i-1}, t_{i-1}) \xrightarrow{a_i, \theta(a_i)} (s_i, t_i) \}_{1 \leq i \leq |A|}$.

Dans le problème d'ordonnancement d'un tel système, nous cherchons non seulement à respecter les échéances, mais aussi à utiliser au mieux les ressources de calcul. Le premier point correspond au respect des contraintes de temps définies par l'utilisateur. En ce qui concerne l'utilisation des ressources de calcul, nous nous intéressons aux deux critères suivants :

- Le premier concerne l'utilisation de la plateforme, que nous mesurons grâce aux durées d'exécution de la séquence d'actions représentant le système. Puisque ces durées d'exécution sont des fonctions croissantes du niveau de qualité, en les maximisant nous maximisons aussi les niveaux de qualité qui sont choisis par le contrôleur. Nous supposons de plus que la précision des traitements et calculs qui sont implémentés

dans l'application correspond aussi à une fonction croissante des niveaux de qualité.
- Le second critère est la régularité de l'affectation de qualité. En évitant les fluctuations des niveaux de qualité choisis par le contrôleur, nous cherchons à garantir une qualité constante des traitements mis en oeuvre dans l'application.

Définition 2.4 (Problème de contrôle de qualité pour des durées d'exécution connues): *Soit un système $SY = (G, Q, C, D)$, trouver un contrôleur Γ qui calcule un ordonnancement $a_1 .. a_n$ et une affectation de qualité θ, tels que :*
- *le contrôleur est **sûr** c'est-à-dire, les échéances des actions ne sont pas violées, ce qui signifie que pour chaque état (s_i, t_i) de $SY||\Gamma$ nous avons $D(a_i) \geq t_i$. C'est le critère de sûreté.*
- *le **taux d'utilisation du budget de temps** est **maximal**, c'est-à-dire que pour chaque contrôleur sûre Γ', $t_n \geq t'_n$, où t_n (resp. t'_n) est la durée d'exécution de la dernière action de $SY||\Gamma$ (resp. $SY||\Gamma'$). C'est un critère d'optimalité.*
- *L'affectation de qualité θ est la plus régulière possible, c'est-à-dire que les fluctuations des niveaux de qualité sont minimales parmi les ordonnancements qui maximisent le taux d'utilisation du budget de temps. C'est le critère de régularité.*

Exemple 2.2: *Considérons le système $SY = (G, Q, C, D)$ donné par la figure 2.1. Nous prenons $Q = \{ q_{min}, q_{max} \}$, une fonction de durée d'exécution C et une échéance D(cf. figure 2.2.*

action a	$C(a, q_{min})$	$C(a, q_{max})$	$D(a)$
`Quant`	10	20	$+\infty$
`IQuant`	25	75	$+\infty$
`IDCT`	25	75	D_1
`IntraP`	5	25	$+\infty$
`Coding`	5	25	D_2

Figure 2.2 — Fonctions de durée d'exécution C et d'échéance D.

Si $D($`IDCT`$) = D_1 = 180$ et $D($`Coding`$) = D_2 = 240$, un contrôleur Γ calculant un ordonnancement `Quant IntraP Coding IQuant IDCT` *et l'affectation de qualité $\theta = q_{max}$ n'est pas sûre. En effet, on a :*

$$C(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max}) = 20 + 75 + 75 + 25 + 25 = 220$$
$$C(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max}) > D(\texttt{IDCT}) = D_1 = 180\ .$$

Une solution pour le problème du contrôle de qualité pour SY est un contrôleur Γ calculant un ordonnancement `Quant IQuant IDCT IntraP Coding` *avec une affectation de qualité constante $\theta = q_{max}$. Puisque les niveaux de qualité sont maximaux, la durée d'exécution totale $t_n = 220$ est maximale. De plus le contrôleur est sûr :*

$$C(\texttt{Quant IQuant IDCT}, q_{max}) = 170 \leq D(\texttt{IDCT}) = D_1 = 180, \text{ et}$$
$$C(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max}) = 220 \leq D(\texttt{Coding}) = D_2 = 240\ .$$

Comme $C : A \times Q \rightarrow \mathbb{R}^+$ est un fonction de durée d'exécution connue, le contrôleur Γ peut être calculé statiquement. Maintenant que les bases du problème de contrôle de qualité ont été posées, nous allons présenter un algorithme permettant de calculer un ordonnancement a_1 .. a_n et une affectation de qualité θ qui est une solution pour le problème de contrôle de qualité.

2.1.1.2 Conception du contrôleur de qualité de service pour des durées d'exécution connues

Nous allons résoudre le problème de contrôle de qualité sans incertitude (définition 2.4) en utilisant un algorithme de contrôle ad-hoc. Ce dernier est basé sur le principe E.D.F.(*Earliest Deadline First*), qui consiste à ordonnancer en priorité les actions dont l'échéance est la plus proche.

Dans un premier temps nous définirons la politique de gestion de qualité t_p qui permet de représenter la durée d'une séquence d'exécution pour un niveau de qualité donné. Cette politique va nous permettre de déterminer si la séquence d'exécution est sûre dans le temps qu'il reste à l'application pour s'exécuter. Nous définirons ensuite la notion d'ordonnancement E.D.F. et verrons que de tels ordonnancements sont optimaux.

Politique de gestion de qualité

Pour un ordonnancement $a_1 \ldots a_n$ et une affectation de qualité θ, la politique de gestion de qualité $t_p(a_1 \ldots a_n, \theta)$ représente la marge dans une exécution sans temps mort, selon l'ordre des actions et démarrant à la date 0. Ainsi si $t_p(a_1 \ldots a_n, \theta)$ est négatif, l'ordonnancement n'est pas sûr. Au contraire si $t_p(a_1 \ldots a_n, \theta)$ est positif, il est possible de retarder la date de démarrage à $t_p(a_1 \ldots a_n, \theta)$ tout en respectant les échéances.

Définition 2.5 (politique de gestion de qualité):
Soient un système $SY = (G, Q, C, D)$, un ordonnancement a_1 .. a_n et une affectation de qualité θ, la ***politique de gestion de qualité*** *est définie par :*

$$t_p(a_1 \,..\, a_n, \theta) = \min_{\mathbf{1 \leq k \leq n}} D(a_k) - C(a_1 \,..\, a_k, \theta),$$

où $C(a_1$ .. $a_k, \theta)$ représente la durée d'exécution de la séquence d'actions a_1 .. a_k pour le niveau de qualité θ, c'est-à-dire :

$$C(a_1 \,..\, a_k, \theta) = \sum_{1 \leq i \leq k} C(a_i, \theta(a_i)) \ .$$

La valeur de $t_p(a_1$ .. $a_n, \theta)$ représente la marge au plus tard à laquelle l'ordonnancement a_1 .. a_n peut s'exécuter tout en respectant les échéances pour l'affectation de qualité θ.

Proposition 2.6:
Soit $SY = (G, Q, C, D)$ un système et Γ un contrôleur de SY qui calcule un ordonnancement $a_1 .. a_n$ et une affectation de qualité θ. Le contrôleur Γ est sûr si et seulement si $t_p(a_1 .. a_n, \theta) \geq 0$.

Preuve (de la proposition 2.6):
Nous avons :

$$\begin{aligned} & t_p(a_1 .. a_n, \theta) \geq 0 \\ \Leftrightarrow \quad & \min_{\mathbf{1 \leq k \leq n}} D(a_k) - C(a_1 .. a_k, \theta) \geq 0 \\ \Leftrightarrow \quad & \forall k \in \{\ 1, \ldots, n\ \}\ .\ D(a_k) \geq C(a_1 .. a_k, \theta) \\ \Leftrightarrow \quad & \forall k \in \{\ 1, \ldots, n\ \}\ .\ D(a_k) \geq t_k\ . \end{aligned}$$

□

Exemple 2.3: *Pour deux ordonnancements du système donné dans l'exemple 2.2,* `Quant IntraP Coding IQuant IDCT` *et* `Quant IQuant IDCT IntraP Coding`*, nous avons :*

$$\begin{aligned} & t_p(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max}) \\ = \quad & \min\ \{\ D_2 - C(\texttt{Quant IntraP Coding}, q_{max})\ ,\ D_1 - C(\texttt{Quant IntraP Coding IQuant IDCT}, q_{max})\ \} \\ = \quad & \min\ \{\ 240 - 70\ ,\ 180 - 220\ \} = -40, \text{ et} \end{aligned}$$

$$\begin{aligned} & t_p(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max}) \\ = \quad & \min\ \{\ D_1 - C(\texttt{Quant IQuant IDCT}, q_{max})\ ,\ D_2 - C(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max})\ \} \\ = \quad & \min\ \{\ 180 - 170\ ,\ 240 - 220\ \} = 10\ . \end{aligned}$$

Nous pouvons conclure que l'ordonnancement `Quant IntraP Coding IQuant IDCT` *viole l'échéance D_1 de l'action* `IDCT` *pour un niveau de qualité constant q_{max}, alors que l'ordonnancement* `Quant IQuant IDCT IntraP Coding` *respecte toutes les échéances pour un même niveau de qualité q_{max}.*

Politique d'ordonnancement optimal

Dans ce qui précède nous avons donné la définition d'une politique de gestion de qualité. Afin de définir correctement une politique de contrôle, nous devons à présent définir la notion de politique d'ordonnancement optimal. Celle-ci est définie par une fonction $Best\_Sched$ qui fournit pour un ordonnancement $a_1 \ldots a_n$ et une affectation de qualité θ un ordonnancement optimal $a_1^\theta .. a_n^\theta$, c'est-à-dire un ordonnancement des actions qui n'ont pas encore été exécutées. On dit que cette ordonnancement est optimal car chaque ordonnancement $a_1^\theta .. a_n^\theta$ est optimisé par rapport au niveau de qualité q. C'est-à-dire qu'il maximise la fonction t_p ce qui permet d'optimiser l'ensemble des niveaux de qualité admissibles.

Définition 2.8 (Politique d'ordonnancement optimal):
Pour un système $SY = (G, Q, C, D)$, une ***politique d'ordonnancement optimal*** *est définie par une fonction $Best\_Sched$ qui pour chaque affectation de qualité θ donne un ordonnancement $a_1^\theta \,..\, a_n^\theta = Best\_Sched(\theta)$ tel que $a_1^\theta \,..\, a_n^\theta$ maximise $t_p(a_1^\theta \,..\, a_n^\theta, \theta)$, c'est-à-dire*

$$t_p(a_1^\theta \,..\, a_n^\theta, \theta) = \mathbf{max} \{\ t_p(a_1 \,..\, a_n, \theta) \mid a_1 \,..\, a_n \in \Sigma(G)\ \}\ .$$

Soit θ une affectation de qualité. On peut constater que, pour chaque ordonnancement $a_1 \,..\, a_n$, la durée d'exécution totale $C(a_1 \,..\, a_n, \theta)$ est indépendante de $a_1 \,..\, a_n$. Nous représentons par $\ll$ la relation binaire sur Θ tel que $\theta \ll \theta' \Leftrightarrow C(a_1 \,..\, a_n, \theta) < C(a_1 \,..\, a_n, \theta')$. La relation $\ll$ est un ordre total strict sur les classes d'affectation de qualité $\{\ \theta \mid C(a_1 \,..\, a_n, \theta) = \text{constant}\ \}$.

Nous allons maintenant présenter un algorithme répondant au problème de contrôle de qualité sans incertitude. Cet algorithme permet de combiner les deux notions que nous avons vu précédemment qui sont le choix de l'ordonnancement optimal, ainsi que la gestion des niveaux de qualité utilisés.

Proposition 2.9:
Pour un système donnée $SY = (G, Q, C, D)$ et un Ordonnanceur optimal $Best\_Sched$, l'algorithme 2.1 fournit une solution au problème de contrôle de qualité pour des durées d'exécution connues (définition 2.4).

```
1 forall θ ∈ Θ do
2 |  a_1^θ .. a_n^θ := Best_Sched(θ);
3 end
4 θ_M = max_≪ { θ : A → Q | t_p(a_1^θ .. a_n^θ, θ) ≥ 0 };
5 return (a_1^{θ_M} .. a_n^{θ_M}, θ_M)
```

Algorithme 2.1 : Algorithme de contrôle pour des durées d'exécution connues.

Preuve (de la proposition 2.9):
Soit $a_1 \ldots a_n$ un ordonnancement et une affectation de qualité θ vérifiant $t_p(a_1 \ldots a_n, \theta) \geq 0$. Par définition de $Best\_Sched$, on a

$$t_p(a_1^\theta \ldots a_n^\theta) \geq t_p(a_1 \ldots a_n) \geq 0\ .$$

Comme $\theta_M = \mathbf{max}_{\ll} \{\ \theta : A \to Q \mid t_p(a_1^\theta \,..\, a_n^\theta, \theta) \geq 0\ \}$, on a $\theta \ll \theta_M$ □

Ordonnancement E.D.F.

Nous allons maintenant définir le principe d'ordonnancement E.D.F., puis montrer

qu'il est optimal. Son calcul est basé sur une propagation arrière des échéances critiques dans le graphe de précédence.

Définition 2.11 (Ordonnancement E.D.F.):
Soit un système $SY = (G, Q, C, D)$, nous définissons la fonction d'échéance **globale** $D^* : A \to \mathbb{R}^+$ *par :*

$$D^*(a) = \min \{ D(a') \mid a \prec a' \} \cup \{ D(a) \} .$$

Un ordonnancement $a_1 .. a_n$ de G est un ordonnancement ***E.D.F.*** *si pour tout $i \in \{ 1, \ldots, n-1 \}$ nous avons $D^*(a_i) \leq D^*(a_{i+1})$. Nous représentons par $E . D . F . (G, D)$ l'ensemble des ordonnancements E.D.F. de G qui respectent la fonction d'échéance D.*

Exemple 2.4: *Considérons un système $SY = (G, Q, C, D)$ donné dans les exemples 2.2 et 2.3. La fonction d'échéance globale D^* est telle que $D^*(\texttt{Quant}) = \min \{ D_1, D_2 \}$, $D^*(\texttt{IQuant}) = D(\texttt{IDCT}) = D_1$ et $D^*(\texttt{IntraP}) = D(\texttt{Coding}) = D_2$ (cf. figure 2.3). Pour les actions* `IDCT` *et* `Coding`*, les fonctions D^* et D sont identiques.*
Si $D_1 < D_2$, le seul ordonnancement E.D.F. est donné par la séquence d'actions `Quant IQuant IDCT IntraP Coding`*. Si $D_1 > D_2$, le seul ordonnancement E.D.F. est donné par la séquence d'actions* `Quant IntraP Coding IQuant IDCT`*.*

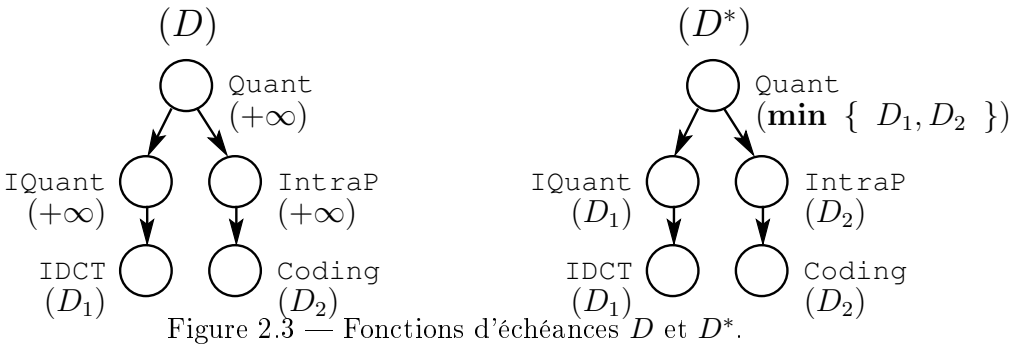


Figure 2.3 — Fonctions d'échéances D et D^*.

Proposition 2.12:
Les systèmes $SY = (G, Q, C, D)$ et $SY^* = (G, Q, C, D^*)$ possèdent la même politique de contrôle :

$$t_p(a_1 .. a_n, \theta) = \min_{\mathbf{1 \leq k \leq n}} D(a_k) - C(a_1 .. a_k, \theta) = \min_{\mathbf{1 \leq k \leq n}} D^*(a_k) - C(a_1 .. a_k, \theta) .$$

Preuve (de la proposition 2.12):
(par l'absurde ou contradiction) Sans perte de généralité, nous supposons que $C > 0$. Supposons que :

$$\min_{\mathbf{1 \leq k \leq n}} D^*(a_k) - C(a_1 .. a_k, \theta) \neq \min_{\mathbf{1 \leq k \leq n}} D(a_k) - C(a_1 .. a_k, \theta) .$$

Comme $D^* \leq D$, nous obtenons :

$$\min_{\mathbf{1 \leq k \leq n}} D^*(a_k) - C(a_1 .. a_k, \theta) < \min_{\mathbf{1 \leq k \leq n}} D(a_k) - C(a_1 .. a_k, \theta) .$$

Il existe un index i tel que :

$$D^*(a_i) - C(a_1 .. a_i, \theta) = \min_{\mathbf{1 \leq k \leq n}} D^*(a_k) - C(a_1 .. a_k, \theta) .$$

et

$$D^*(a_i) - C(a_1 .. a_i, \theta) < D(a_i) - C(a_1 .. a_i, \theta)$$

c'est-à-dire, $D^*(a_i) < D(a_i)$. Par définition de D^*, nous pouvons conclure qu'il existe $j > i$ tel que $D^*(a_i) = D^*(a_j)$. Alors, nous obtenons :

$$\begin{aligned} D^*(a_j) - C(a_1 .. a_j, \theta) &= D^*(a_j) - C(a_1 .. a_i, \theta) - C(a_{i+1} .. a_j, \theta) \\ D^*(a_j) - C(a_1 .. a_j, \theta) &< D^*(a_j) - C(a_1 .. a_i, \theta) \\ D^*(a_j) - C(a_1 .. a_j, \theta) &< D^*(a_i) - C(a_1 .. a_i, \theta) (\text{Contradiction}). \end{aligned}$$

□

La proposition 2.16 permet le calcul d'une fonction *Best_Sched* qui renvoie un ordonnancement E.D.F. $a_1 .. a_n = Best\_Sched(\theta)$. Notons que cette fonction *Best_Sched* est constante et son calcul peut être réalisé dans un temps polynomial. Pour démontrer la proposition, nous aurons besoin du lemme suivant.

Lemme 2.14:
Soit $a_1 .. a_n$ un ordonnancement tel qu'il existe deux actions consécutives et indépendantes a_i et a_{i+1} ($a_i \not\prec a_{i+1}$ et $a_{i+1} \not\prec a_i$) tel que $D(a_i) \geq D(a_{i+1})$. Pour chaque affectation de qualité θ nous avons :

$$t_p(a_1 .. a_{i-1} a_{i+1} a_i a_{i+2} .. a_n, \theta) \geq t_p(a_1 .. a_n, \theta) .$$

Preuve (du lemme2.14):
Soit I_1, I_2 et I_3 des sous-ensembles d'index tels que $I_1 = \{ 1, \ldots, i-1 \}$, $I_2 = \{ i, i+1 \}$ et $I_3 = \{ i+2, \ldots, n \}$. Nous avons :

$$t_p(a_1 .. a_n, \theta) = \min \{ D(a_k) - C(a_1 .. a_k, \theta) \mid k \in I_1 \cup I_2 \cup I_3 \} \text{ et}$$

$t_p(a_1 .. a_{i-1} a_{i+1} a_i a_{i+2} .. a_n, \theta) =$
$\min \{ D(a_k) - C(a_1 .. a_k, \theta) \mid k \in I_1 \} \cup \{ D(a_{i+1}) - C(a_1 .. a_{i-1} a_{i+1}, \theta) \} \cup$
$\{ D(a_i) - C(a_1 .. a_{i-1} a_{i+1} a_i, \theta) \} \cup \{ D(a_k) - C(a_1 .. a_{i-1} a_{i+1} a_i a_{i+2} .. a_k, \theta) \mid k \in I_3 \} .$

Comme $C(a_1 .. a_{i-1} a_{i+1} a_i a_{i+2} .. a_k, \theta) = C(a_1 .. a_k, \theta)$ pour chaque $k \in I_3$, nous avons :

$$\{ D(a_k) - C(a_1 .. a_{i-1} a_{i+1} a_i a_{i+2} .. a_k, \theta) \mid k \in I_3 \} = \{ D(a_k) - C(a_1 .. a_k, \theta) \mid k \in I_3 \} .$$

Donc, le lemme reste vrai si :

$$D(a_{i+1}) - C(a_1 .. a_{i-1} a_{i+1}, \theta) \geq \min \{ D(a_k) - C(a_1 .. a_k, \theta) \mid k \in I_2 \} \text{ et } (2.1)$$
$$D(a_i) - C(a_1 .. a_{i-1} a_{i+1} a_i, \theta) \geq \min \{ D(a_k) - C(a_1 .. a_k, \theta) \mid k \in I_2 \} . \quad (2.2)$$

Comme $C(a_1 .. a_{i-1}a_{i+1}, \theta) \leq C(a_1 .. a_{i+1}, \theta)$ *nous avons :*

$$D(a_{i+1}) - C(a_1 .. a_{i-1}a_{i+1}, \theta) \geq D(a_{i+1}) - C(a_1 .. a_{i+1}, \theta) .$$

Cela démontre (2.1).

Puisque $D(a_i) \geq D(a_{i+1})$ *et* $C(a_1 .. a_{i-1}a_{i+1}a_i, \theta) = C(a_1 .. a_{i+1}, \theta)$ *nous avons :*

$$D(a_i) - C(a_1 .. a_{i-1}a_{i+1}a_i, \theta) \geq D(a_{i+1}) - C(a_1 .. a_{i+1}, \theta) .$$

Cela démontre (2.2). □

Proposition 2.16:
Soient $SY = (G, Q, C, D)$ *un système et* $a_1 .. a_n$ *un ordonnancement E.D.F. de* G*. Pour chaque affectation de qualité* θ *nous avons :*

$$t_p(a_1 .. a_n, \theta) = \mathbf{max} \{ t_p(a'_1 .. a'_n, \theta) \mid a'_1 .. a'_n \in \Sigma(G) \} .$$

Preuve (de la proposition 2.16):
Nous appliquons le lemme 2.14 comme suit. Soit $a_1 .. a_n$ *un ordonnancement E.D.F. et* $a'_1 .. a'_n$ *un ordonnancement arbitraire. Nous pouvons obtenir* $a_1 .. a_n$ *à partir* $a'_1 .. a'_n$ *par échanges successifs de deux actions consécutives indépendantes telles que leurs échéances* D^* *sont inversées.* □

Exemple 2.5: *Considérons un système* $SY = (G, Q, C, D)$ *donné dans les exemples 2.2, 2.3 et 2.4, et une fonction d'échéance* D *telle que* $D(\texttt{IDCT}) = D_1 = 180$ *et* $D(\texttt{Coding}) = D_2 = 240$*. Le seul ordonnancement E.D.F. est* `Quant IQuant IDCT IntraP Coding` *et nous avons* $t_p(\texttt{Quant IQuant IDCT IntraP Coding}, q_{max}) = 10$ *(cf. exemple 2.3), qui est maximale pour le niveau de qualité* q_{max}*. Nous donnons dans la suite une preuve.*

Considérons un ordonnancement arbitraire $a_1 .. a_5$ *de* G*, et soit* $i \in \{ 1, \ldots, 5 \}$ *l'index tel que* $a_i =$ `IDCT`*. Etant donné les contraintes de précédence de* G*,* `Quant` *et* `IQuant` *doivent être exécutées avant* `IDCT`*. De ce fait, on a :*

$$C(a_1 .. a_i, q_{max}) \geq C(\texttt{Quant}, q_{max}) + C(\texttt{IQuant}, q_{max}) + C(\texttt{IDCT}, q_{max}) = 170 .$$

Cela démontre que :

$$t_p(a_1 .. a_5, q_{max}) \leq D(a_i) - C(a_1 .. a_i, q_{max}) \leq 180 - 170 = 10 .$$

2.1.2 Problème du contrôle de qualité sous incertitude

Les durées d'exécution peuvent varier considérablement durant le déroulement d'une application, étant donné qu'elles dépendent des contenus des données. De plus, la non prévisibilité du comportement de la plateforme est un facteur additionnel de cette incertitude. Nous allons nous intéresser dans cette section à une extension du problème de contrôle de qualité de la section précédente.

2.1.2.1 Le problème

Dans ce nouveau problème, les durées d'exécution ne sont pas connues mais seulement bornées par des durées d'exécution pire-cas. Ainsi la fonction C des durées d'exécution réelles n'est, elle aussi, pas connue, cependant nous connaissons une fonction C^{wc} telle que $C \leq C^{wc}$. De plus les durées d'exécution réelles C peuvent être connues à posteriori, c'est-à-dire après l'exécution de chaque action.

Définition 2.18 (système paramétré incertain):
Un ***système paramétré incertain*** *est un n-uplet* $SPI(C) = (G, Q, C^{wc}, D, C)$ *où :*

- G *est un* ***graphe de précédence****.*
- $Q = [q_{min}, q_{max}]$ *est un intervalle fini d'entiers correspondant aux* ***niveaux de qualité****.*
- $C^{wc} : A \times Q \to \mathbb{R}^+$ *est une fonction qui pour une action* a *et un niveau de qualité* q *donne sa durée d'exécution* ***pire-cas*** $C^{wc}(a, q)$*. Nous supposons que, pour tout* $a \in A$*,* $q \mapsto C^{wc}(a, q)$ *est une fonction croissante.*
- $D : A \to \mathbb{R}^+ \cup \{ +\infty \}$ *est une fonction qui pour une action* a *donne son* ***échéance*** $D(a)$*.*
- *Le paramètre* $C : A \times Q \to \mathbb{R}^+$ *est une fonction qui pour une action* a *et un niveau de qualité* q *donne sa durée d'exécution* ***réelle*** $C(a, q)$*. Nous supposons que, pour tout* $a \in A$*,* $q \mapsto C(a, q)$ *est une fonction croissante telle que* $C \leq C^{wc}$*.*

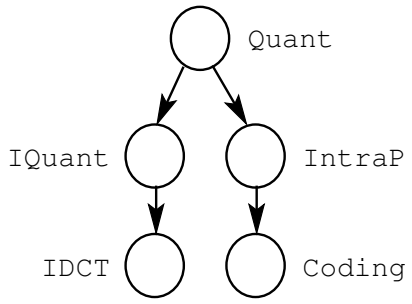
(sémantique) *Le système paramétré incertain* $SPI(C)$ *définit une famille de système de transition* $(S \times \mathbb{R}^+, A \times Q, \longrightarrow)$ *dépendant du paramètre* C *:*

- *les états sont donnés par les paires* (s_i, t_i) *où, pour* $i > 0$*,* s_i *est un état de* G *et* $t_i \in \mathbb{R}^+$ *est une valeur du temps ; nous prenons* $t_0 = 0$ *pour* $s_0 = \emptyset$ *;*
- *pour deux états* (s_i, t_i) *et* (s_j, t_j)*, une action* a *et* q*, nous avons* $(s_i, t_i) \xrightarrow{a,q} (s_j, t_j)$*, si* $s_i \xrightarrow{a} s_j$ *dans* G *et* $t_j - t_i = C(a, q)$*.*

La notion d'ordonnancement d'un système paramétré incertain est similaire à celle que nous avions donnée pour les systèmes (cf. proposition 2.6). C'est-à-dire qu'un ordonnancement $a_1 \dots a_n$ est sûr si les échéances sont respectées par rapport aux durées d'exécution réelles, c'est-à-dire celles qui sont données par le paramètre C. Cependant ces durées d'exécution réelles n'étant pas connues, la recherche d'ordonnancement sûr d'un système paramétré incertain ne peut se faire en comparant simplement les ordonnancements possibles de celui-ci. Puisque $C \leq C^{wc}$, les ordonnancements sûrs pour un système $SY = (G, Q, C^{wc}, D)$ sont aussi sûrs pour un système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$. C'est-à-dire que nous approchons la durée d'exécution inconnue C par la durée d'exécution pire-cas C^{wc}. Cette méthode donne cependant des résultats souvent très éloignés de l'optimalité du fait de la surestimation des durées d'exécution pire-cas par rapport aux durées d'exécution réelles.

Exemple 2.6: *Considérons un système paramétré incertain* $SPI(C) = (G, Q, C^{wc}, D, C)$,

tel que $A = \{\texttt{QUANT}, \texttt{IQUANT}, \texttt{IDCT}, \texttt{IntraP}, \texttt{Coding}\}$, $G = (A, \prec)$ *représenté par la figure 2.4,* $D(\texttt{IDCT}) = \texttt{D}_1 = 180$, $D(\texttt{Coding}) = \texttt{D}_2 = 240$ *et* $Q = \{q_{min}, q_{max}\}$. *Considérons la fonction de durée d'exécution réelle* $C = C^{wc}$. *Alors* $(\texttt{QUANT} \cdot \texttt{IQUANT} \cdot \texttt{IDCT}, q_{min}, 20)$ *est un état de* $SPI(C)$ *puisque* $\texttt{QUANT} \cdot \texttt{IQUANT} \cdot \texttt{IDCT}$ *est une trace de* G, *et* $C(\texttt{QUANT} \cdot \texttt{IQUANT} \cdot \texttt{IDCT}, q_{min}) = 115$.



action a	$C^{wc}(a, q_{min})$	$C^{wc}(a, q_{max})$	$D(a)$
`Quant`	15	25	$+\infty$
`IQuant`	50	100	$+\infty$
`IDCT`	50	100	D_1
`IntraP`	100	100	$+\infty$
`Coding`	100	100	D_2

Figure 2.4 — Exemple de système paramétré incertain

La définition d'un contrôleur pour un système paramétré incertain est identique à celle donnée pour le problème sans incertitude (cf. définition 2.3).

Définition 2.19 (Contrôleur):
Soit un système paramétré incertain $SPI(C)$, *un* ***contrôleur*** *est une fonction* $\Gamma : S \times \mathbb{R}^+ \to A \times Q$ *qui, pour un état* (s_{i-1}, t_{i-1}) *de* $SPI(C)$, *donne une action* a_i *et son niveau de qualité* q_i *tel qu'il existe* (s_i, t_i) *et* $(s_{i-1}, t_{i-1}) \xrightarrow{a_i, q_i} (s_i, t_i)$.

Si les durées d'exécution réelles ne sont pas connues à priori, nous supposons par contre qu'il est possible de connaître la durée d'exécution d'une action à posteriori, c'est-à-dire après avoir été exécutée. Ainsi, pour résoudre le problème, nous allons mettre en oeuvre des techniques adaptatives qui consistent à prendre en compte ces informations au cours de l'exécution du système. Un contrôleur associe donc à tout état un choix d'ordonnancement concernant la prochaine action à exécuter. Il détermine ainsi l'exécution d'un système paramétré incertain de telle sorte que, pour une fonction de durée d'exécution C donnée, il n'existe qu'une et une seule séquence d'exécution complète possible.

Définition 2.20 (Système paramétré incertain contrôlé):
Soient $SPI(C) = (G, Q, C^{wc}, D, C)$, *un système paramétré incertain et* Γ *un contrôleur de* $SPI(C)$. *Nous définissons le système de transitions étiquetées associé au* ***système paramétré incertain contrôlé****, noté* $SPI(C)||\Gamma$ *par :*

1. *Les états de* $SPI(C)||\Gamma$ *sont les mêmes que ceux de* $SPI(C)$.
2. *Pour une fonction de durée d'exécution réelle* C, $SPI(C)||\Gamma$ *possède une unique séquence d'exécution* $\{ (s_{i-1}, t_{i-1}) \xrightarrow{a_i, q_i} (s_i, t_i) \}_{1 \leq i \leq |A|}$ *telle que* $(a_i, q_i) = \Gamma(s_{i-1}, t_{i-1})$.

Le problème de contrôle de qualité pour une système paramétré incertain donné $SPI(C) = (G, Q, C^{wc}, D, C)$ consiste à trouver un contrôleur Γ tel que le système contrôlé

respecte les échéances tout en gardant un niveau de qualité maximal et régulier. Nous pouvons le formaliser comme suit.

Définition 2.21 (Problème de contrôle de qualité sous incertitude):
Soit un système paramétré incertain $SPI(C)$, il faut trouver un contrôleur Γ tel que pour chaque fonction de durée d'exécution réelle $C \leq C^{wc}$:

1. *Γ est sûr (les échéances ne sont pas violées), c'est-à-dire que pour tout état (s_{i-1}, t_{i-1}) de $SPI(C)||\Gamma$, nous avons $(a_i, q_i) = \Gamma(s_{i-1}, t_{i-1})$, c'est à dire $(s_{i-1}, t_{i-1}) \xrightarrow{a_i,q_i} (s_i, t_i)$ et $D(a_i) \geq t_i$.*
2. *Le taux d'utilisation de la plateforme est maximal, c'est-à-dire que pour tout contrôleur sûr Γ', $t_n \geq t'_n$, où t_n (resp. t'_n) est la durée d'exécution de la dernière action de $SPI(C)||\Gamma$ (resp. $SPI(C)||\Gamma'$).*
3. *L'affectation du niveau de qualité est régulière. Par exemple, nous chercherons à ce que l'écart type ou la variance de l'affectation puissent être utilisés pour mesurer la régularité de l'affectation de qualité.*

La troisième propriété du problème ne sera pas formalisée, mais elle reste essentielle dans le cadre des applications multimédia [36, 37].

Étant donné que les durées d'exécution réelles ne sont pas connues, le contrôleur que nous cherchons ne doit pas dépendre de C. Ainsi un tel contrôleur ne pourra pas atteindre exactement l'optimal. C'est la raison pour laquelle le problème sera de trouver un contrôleur Γ dont les performances sont les plus proches possibles de l'optimal. Pour résoudre ce problème, nous nous sommes orientés vers une approche qui consiste à concevoir un contrôleur ad-hoc satisfaisant une propriété. Il convient alors de vérifier que le système contrôlé vérifie la propriété de départ. Dans le cas où la propriété ne serait pas satisfaite, il est nécessaire de modifier le contrôleur, et de vérifier à nouveau si le système satisfait la propriété de départ avec ce contrôleur modifié. Ce processus itératif peut prendre un temps non négligeable.

2.1.2.2 Conception du contrôleur de qualité de service sous incertitude

Du fait de l'incertitude sur les durées d'exécution, le calcul de l'ordonnancement et de l'affectation de la qualité que l'on peut lui associer est fait en ligne. Nous adaptons l'algorithme abstrait proposé dans la section 2.1.1.2 (proposition 2.9). Pour faire face au problème d'explosion d'état, l'algorithme considère à chaque état des affectations de qualité constantes pour les actions restantes. Cette hypothèse ne restreint pas la pertinence du problème dans le sens où l'on cherche à obtenir une affectation de qualité régulière.

L'algorithme proposé est paramétré par une politique X caractérisée par une fonction de durée d'exécution C^X donnée pour une séquence d'actions $a_1 \dots a_n$ et un niveau de qualité q. Celle-ci représente une estimation $C^X(a_1 \dots a_n, q)$ de la durée d'exécution de $a_1 \dots a_n$ pour une qualité q. L'algorithme utilise une estimation adéquate t_p^X de t_p, et

$Best\_Sched^X$ de $Best\_Sched$, défini à partir d'une fonction de durée d'exécution C^X basée sur une politique X de gestion de qualité.

Définition 2.22 (Politique X de gestion de qualité):
Soient $SPI(C) = (G, Q, C^{wc}, D, C)$ un système paramétré incertain et C^X une fonction qui donne pour une séquence d'actions $a_i \,..\, a_n$ et un niveau de qualité q, une estimation $C^X(a_i \,..\, a_n, q)$ de la durée d'exécution de $a_i \,..\, a_n$ pour le niveau de qualité q.

Soient un état s_{i-1} de G, un ordonnancement $a_i \,..\, a_n$ à partir de s_{i-1}, et un niveau de qualité q. Nous définissons la politique X de gestion de qualité t_p^X associée à la fonction de durée d'exécution C^X par :

$$t_p^X(a_i \,..\, a_n, s_{i-1}, q) = \min_{\mathbf{i \le k \le n}} D(a_k) - C^X(a_i \,..\, a_k, q) \; .$$

Définition 2.23 (Durée d'exécution croissante):
*Soit $C^X : A^* \times Q \to \mathbb{R}^+$ une fonction de durée d'exécution. Nous dirons que C^X est **croissante** si pour toutes séquences $a_1 \ldots a_n \ldots a_m$ et pour tout niveau de qualité q nous avons :*

$$C^X(a_1 \ldots a_n, q) \le C^X(a_1 \ldots a_m, q) \; .$$

Une fonction de durée d'exécution croissante C^X est telle que la durée d'exécution d'une séquence augmente si des actions lui sont ajoutées, ce qui semble une hypothèse raisonnable. Dans la suite, les fonctions C^X qui serviront à bâtir les fonctions d'ordonnancement proposées seront toujours croissantes.

Définition 2.24 (Politique X d'ordonnancement optimal):
*Pour un système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$ et une politique de gestion de qualité t_p^X, un **Ordonnanceur optimal** $Best\_Sched^X$ est une fonction donnant, pour une état s_{i-1} de G et pour un niveau de qualité q, un ordonnancement $a_i^q \,..\, a_n^q = Best\_Sched^X(s_{i-1}, q)$ à partir de l'état s_{i-1} tel que $a_i^q \,..\, a_n^q$ maximise la politique de gestion de qualité t_p^X c'est-à-dire :*

$$t_p^X(a_i^q \,..\, a_n^q, s_{i-1}, q) = \mathbf{max} \; \{ \; t_p^X(a_i \,..\, a_n, s_{i-1}, q) \mid a_i \,..\, a_n \in \Sigma(G, s_{i-1}) \; \} \; .$$

La figure 2.5 montre l'interaction entre le contrôleur Γ, et un système paramétré incertain $SPI(C)$ représentant une application s'exécutant sur une plateforme. Le contrôleur est composé de deux entités qui fonctionnent en interaction : le Gestionnaire de Qualité et l'Ordonnanceur (voir figure 2.5).

Soit (s_{i-1}, t_{i-1}) l'état courant du système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$.

Gestionnaire de Qualité. Le rôle du Gestionnaire de Qualité est de déterminer le niveau de qualité courant $q \in Q$.

Le choix de q est basé sur un *critère d'admission* de la forme $t_p^X(a_i^q \ldots a_n^q, s_{i-1}, q) \ge t_{i-1}$, dans lequel $a_i^q \ldots a_n^q$ représente l'ordre d'exécution des actions qui restent à

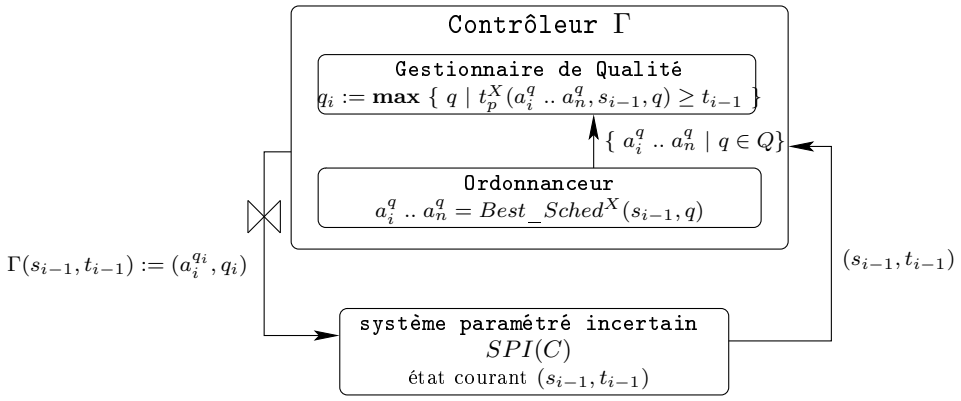


Figure 2.5 — Architecture du contrôleur.

exécuter, s_{i-1} un état de l'application logicielle et q leur niveau de qualité. La fonction $t_p^X : A \times S \times Q \to \mathbb{R}^+$ définie la *politique de gestion de qualité* du contrôleur Γ^X. Lorsque le temps théorique $t_p^X(a_i^q \ldots a_n^q, s_{i-1}, q)$ dépasse le temps-réel t_{i-1}, nous considérons que $a_i^q \ldots a_n^q$ est admissible au niveau de qualité q, c'est-à-dire que l'application est suffisamment en avance pour pouvoir ordonnancer au niveau de qualité q et selon l'ordre imposé par $a_i^q \ldots a_n^q$. Ainsi il est donc possible de terminer l'exécution de l'application contrôlée sans violer d'échéances. Si aucun niveau de qualité ne permet d'obtenir d'ordonnancements admissibles, le Gestionnaire de Qualité choisira le niveau de qualité minimal q_{min}. Plus formellement le Gestionnaire de Qualité choisit le niveau de qualité courant tel que :

$$q_i := \mathbf{max}(\{q | t_p^X(a_i \ldots a_n, s_{i-1}, q) \geq t_{i-1}\} \cup \{q_{min}\})$$

Ordonnanceur. Le rôle de l'Ordonnanceur est de calculer, pour chaque niveau de qualité $q \in Q$, un ordonnancement optimisé $a_i^q \ldots a_n^q = Best\_Sched^X(s_{i-1}, q)$ pour le niveau de qualité q. La fonction $Best\_Sched^X$ définit la *politique d'ordonnancement* du contrôleur Γ^X. Puisque le critère d'admission du Gestionnaire de Qualité est de la forme $t_p^X(a_i^q \ldots a_n^q, s_{i-1}, q) \geq t_{i-1}$, l'optimisation consiste à trouver un ordonnancement $a_i^q \ldots a_n^q$ qui maximise $t_p^X(a_i^q \ldots a_n^q, s_{i-1}, q)$. Ceci permet de maximiser l'ensemble des valeurs du temps-réel t pour lesquelles le $a_i^q \ldots a_n^q$ est admissible au niveau de qualité q à partir de l'état s_{i-1} de l'application logicielle.

Bien que nous limitions l'exploration aux ordonnancements de la forme $(a_i^q \ldots a_n^q, q)$, c'est-à-dire dont le niveau de qualité est constant, nous obtenons quand même de bons résultats quant à l'utilisation des ressources de calcul. En effet, le contrôle étant réalisé à grain fin, les choix du contrôleur, et en particulier ceux qui concernent le niveaux de qualité, sont remis en cause et réévalués après l'exécution de chaque action, en fonction de l'état courant du système paramétré incertain.

Le Contrôleur surveille l'état courant (s_{i-1}, t_{i-1}) de $SPI(C)$ et calcule la prochaine action a_i ainsi que le niveau de qualité q_i associé à celle-ci, comme spécifié par l'algorithme suivant qui généralise le premier donné dans la proposition 2.1.

```
1 Γ^X(s_{i-1},t_{i-1})
2 q_m := q_min
3 forall q ∈ Q do
4 |   a_i^q .. a_n^q := Best_Sched^X(s_{i-1}, q) ;
5 |   if t_p^X(a_i^q .. a_n^q, s_{i-1}, q) ≥ t_{i-1} ∧ q > q_m then
6 |   |   q_m := q;
7 |   end
8 end
9 return (a, q) := (a_i^{q_m}, q_m).
```

Algorithme 2.2 : Algorithme de calcul du contrôleur Γ^X.

L'algorithme de contrôle est donné par l'algorithme 2.2. Il prend en entrée l'état courant (s_{i-1}, t_{i-1}) du système paramétré incertain considéré $SPI(C) = (G, Q, C^{wc}, D, C)$. Pour chaque niveau de qualité q, l'Ordonnanceur calcule les ordonnancements admissibles par rapport à la politique X de gestion de qualité en évaluant le critère d'admission $t_p^X(a_i^q \ldots a_n^q, s_{i-1}, q) \geq t_{i-1}$. L'algorithme renvoie finalement la première action $a = a_i^{q_m}$ et le niveau de qualité q_m qui représente le niveau de qualité maximal parmi les niveaux de qualité q où l'ordonnancement $a_i^q \ldots a_n^q$ est admissible. S'il n'existe pas d'ordonnancement admissible, c'est le niveau de qualité minimal q_{min} qui est choisi.

Le reste du chapitre sera consacré à la définition et à l'étude de différentes politiques de contrôle permettant de résoudre ce problème. Nous présentons plusieurs instanciations du contrôleur Γ^X, notamment Γ^{sf}, Γ^{sp} et Γ^{mx} qui correspondent respectivement aux politiques de contrôle *sûre*, *simple* et *mixte*.

2.2 Politique de gestion de qualité

Dans cette section nous définissons une politique X de gestion de qualité t_p^X, qui permet de répondre au problème de contrôle sous incertitude défini dans la section précédente (cf. définition 2.21). Plus concrètement nous allons présenter une politique permettant de satisfaire à la fois les critères d'optimalité et de sûreté. L'optimalité consiste en deux propriétés qui sont l'utilisation maximale du budget de temps disponible, et la régularité des niveaux de qualité choisis par le contrôleur.

Soit un système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$. Comme les ordonnancements sont calculés et fournis au Gestionnaire de Qualité par l'Ordonnanceur, nous considérons, sans perte de généralité, des politiques de gestion de qualité pour des ordonnancements statiques $a_1 .. a_n$.

2.2.1 Politique de contrôle sûre

Nous allons voir une première politique permettant de montrer comment contraindre la politique de gestion de qualité afin d'assurer que le système paramétré incertain respecte ses échéances. Nous l'appelons politique de gestion de qualité *sûre*.

2.2.1.1 Politique de gestion de qualité sûre

Définition 2.25 (Fonction de durée d'exécution sûre):
Soit $SPI(C) = (G, Q, C^{wc}, D, C)$ un système paramétré incertain. Nous appelons fonction de durée d'exécution sûre la fonction partielle $C^{sf} : A^* \times Q \to \mathbb{R}^+$ définie pour toute séquence d'actions $a_1 \dots a_n$ et toute affectation de qualité q par :

$$C^{sf}(a_i \,..\, a_k, q) = C^{wc}(a_i, q) + C^{wc}(a_{i+1} \,..\, a_k, q_{min}) \,.$$

où $C^{wc}(a_{i+1} \,..\, a_k, q_{min})$ représente la durée d'exécution totale de la séquence $a_{i+1} \,..\, a_k$, c'est-à-dire,

$$C^{wc}(a_{i+1} \,..\, a_k, q_{min}) = \sum_{i+1 \leq j \leq k} C^{wc}(a_j, q_{min}) \,.$$

Définition 2.26 (Politique de gestion de qualité sûre):
*La **politique de gestion de qualité sûre** est donnée par la fonction partielle $t_p^{sf} : A^* \times Q :\to \mathbb{R}$ associée à la fonction de durée d'exécution sûre C^{sf}.*

Comme le contrôleur peut changer le niveau de qualité après l'exécution de la première action a_i, nous prenons le niveau de qualité q pour la première action, et q_{min} pour les actions restantes. Donc $t_p^{sf}(s_{i-1}, q) \geq t_{i-1}$ garantit que la première action a_i satisfait son échéance quand on l'exécute avec le niveau de qualité q, et les actions restantes de l'ordonnancement satisfont elles aussi leurs échéances en s'exécutant avec la qualité q_{min}. Ceci est garanti par l'utilisation des durées d'exécution pire-cas. La politique de gestion de qualité sûre sert de base à la construction de contrôleurs sûrs, c'est-à-dire qu'ils assurent à coup sûr le respect des échéances. Par abus de notation, on notera $t_p^{sf} \geq t$ le fait que $\forall i \in \{\, 1, \dots, n \,\},\ t_p^{sf}(a_i \dots a_n, s_i, q_i) \geq t_{i-1}$.

Proposition 2.27 (Sûreté):
Soit un système paramétré $SPI(C) = (G, Q, C^{wc}, D, C)$ tel que les échéances sont satisfaites pour le niveau de qualité minimal et les durées d'exécution pire-cas, c'est-à-dire,

$$\forall k \in \{\, 1, \dots, n \,\} \ .\ D(a_k) \geq C^{wc}(a_1 \,..\, a_k, q_{min}) \,.$$

Alors, le contrôleur Γ^{sf} qui suit la politique de gestion de qualité $t_p^{sf} \geq t$ est sûre.

Cette proposition est vraie pour toute politique de gestion de qualité $t_p^X \geq t$ telle que $t_p^{sf} \geq t_p^X$.

Lemme 2.28:
Pour chaque contrôleur Γ^{sf} satisfaisant les hypothèses de la proposition 2.27, nous avons $q_{min} \in \{\ q \mid t_p^{sf}(a_i \dots a_n, s_{i-1}, q) \geq t_{i-1}\ \}$ à chaque état atteignable (s_{i-1}, t_{i-1}) de $SPI(C)||\Gamma^{sf}$.

Preuve (du lemme 2.28):
La preuve est réalisée par induction sur $i \in \{\ 0, \dots, n-1\ \}$. Soit $\mathcal{P}(i)$ l'assertion $q_{min} \in \{\ q \mid t_p^{sf}(s_i, q) \geq t_i\ \}$.

- $\mathcal{P}(0)$: *nous avons :*

$$t_p^{sf}(s_0, q_{min}) = \min_{\mathbf{1 \leq k \leq n}} D(a_k) - C^{wc}(a_1 \,..\, a_k, q_{min})\ . \tag{2.3}$$

Comme nous avons pour tout $k \in \{\ 1, \dots, n\ \}$, $D(a_k) \geq C^{wc}(a_1 \,..\, a_k, q_{min})$, nous pouvons conclure à partir de (2.3) que comme $t_0 = 0$, $t_p^{sf}(s_0, q_{min}) \geq t_0$ c'est-à-dire, $\mathcal{P}(0)$.

- $\mathcal{P}(i) \Rightarrow \mathcal{P}(i+1)$: *Soit $(a, q) = (a_{i+1}, q_{i+1}) = \Gamma^{sf}(s_i, t_i)$. Il faut montrer que $t_p^{sf}(a_{i+2} \dots a_n, s_{i+1}, q) \geq t$. Nous allons pour cela examiner deux cas, $q = q_{min}$ et $q > q_{min}$.*

Cas 1 *: $q = q_{min}$,*
Par hypothèse d'induction, nous avons $t_p^{sf}(a_{i+1} \dots a_n, s_i, q_{min}) \geq t_i$, c'est-à-dire

$$\min_{\mathbf{i+1 \leq k \leq n}} D(a_k) - C^{sf}(a_{i+1} \,..\, a_k, q_{min}) \geq t_i$$

c'est-à-dire :

$$\begin{aligned}
& \forall k \in \{i+1 \dots n\}\ D(a_k) - C^{sf}(a_{i+1} \,..\, a_k, q_{min}) \geq t_i \\
\Rightarrow\ & \forall k \in \{i+2 \dots n\}\ D(a_k) - C^{wc}(a_{i+1}, q_{min}) - C^{wc}(a_{i+2} \,..\, a_k, q_{min}) \geq t_i \\
& \textit{par définition de } C^{sf} \\
\Rightarrow\ & \forall k \in \{i+2 \dots n\}\ D(a_k) - C^{wc}(a_{i+2} \,..\, a_k, q_{min}) \geq t_i + C^{wc}(a_{i+1}, q_{min}) \geq t_{i+1}\ .
\end{aligned}$$

Donc $t_p^{sf}(a_{i+2} \dots a_n, s_{i+1}, q_{min}) \geq t_{i+1}$.
Cas 2 *: $q > q_{min}$,*

Nous avons $t_p^{sf}(a_{i+1} \dots a_n, s_i, q) \geq t_i$ c'est à dire

$$\min_{\mathbf{i+1 \leq k \leq n}} D(a_k) - C^{sf}(a_{i+1} \,..\, a_k, q) \geq t_i$$

c'est à dire :

$$\begin{aligned}
& \forall k \in \{i+1 \dots n\}\ D(a_k) - C^{sf}(a_{i+1} \,..\, a_k, q) \geq t_i \\
\Rightarrow\ & \forall k \in \{i+2 \dots n\}\ D(a_k) - C^{wc}(a_{i+1}, q) - C^{wc}(a_{i+2} \,..\, a_k, q_{min}) \geq t_i \\
\Rightarrow\ & \forall k \in \{i+2 \dots n\}\ D(a_k) - C^{wc}(a_{i+2} \,..\, a_k, q_{min}) \geq t_i + C^{wc}(a_{i+1}, q) \geq t_{i+1}\ .
\end{aligned}$$

Donc $t_p^{sf}(a_{i+2} \dots a_n, s_{i+1}, q) \geq t_{i+1}$.

□

Preuve (de la proposition 2.27):
A partir du lemme 2.28, nous avons :

$$q_{min} \in \{\ q \mid t_p^{sf}(a_i \dots a_n, s_{i-1}, q) \geq t_{i-1}\ \}$$

pour chaque état de $SPI(C)||\Gamma^{sf}$ *c'est-à-dire,* $\{\ q \mid t_p^{sf}(a_i \dots a_n, s_{i-1}, q) \geq t_{i-1}\ \}$ *est un ensemble non vide. Soit* $(a, q) = (a_i, q_i) = \Gamma^{sf}(s_{i-1}, t_{i-1})$. *Alors nous avons* $t_p^{sf}(a_i \dots a_n, s_{i-1}, q) \geq t_{i-1}$ *et l'action a satisfaisant son échéance :*

$$\begin{aligned} & t_p^{sf}(a_i \dots a_n, s_{i-1}, q) = \min_{\mathbf{i \leq k \leq n}} D(a_k) - C^{sf}(a_i \,..\, a_k, q) \geq t_{i-1} \\ \Rightarrow\ & D(a_i) - C^{wc}(a, q) \geq t_{i-1} \\ \Rightarrow\ & D(a_i) \geq t_{i-1} + C^{wc}(a, q) \geq t_i\ . \end{aligned}$$

Cela démontre que chaque action a satisfait son échéance. □

La proposition 2.27 est intéressante dans la mesure où elle permet de caractériser une classe de politiques de contrôle qui assurent le respect des échéances. Les politiques de contrôle X qui appartiennent à cette classe sont telles que leur politique de gestion de qualité est au moins aussi contraignante que la politique de gestion de qualité sûre, c'est-à-dire $t_p^X \geq t_p^{sf}$.

2.2.1.2 Limitations de la politique sûre

Le contrôleur Γ^{sf} basé sur la politique de contrôle sûre assure à la fois le respect des échéances et un bon taux d'utilisation des ressources. Nous constatons cependant que les niveaux de qualité choisis par le contrôleur ne sont pas réguliers. En effet, en étant trop permissive, la politique de gestion de qualité sûre conduit le Gestionnaire de Qualité à choisir des niveaux de qualité très élevés pour les premières actions, et donc des niveaux très faibles pour les dernières actions, afin d'assurer le respect des échéances.

L'exemple suivant montre que la politique de gestion de qualité sûre peut mener à une réduction du niveau de qualité avant une échéance critique.

Exemple 2.7: *Considérons un* $SPI(C)$ *avec trois actions* { `QUANT`, `IQUANT`, `IDCT` }, *quatre qualités* $Q = \{1, \dots, 4\}$, *et une échéance unique* $D = 9$ *(pour tout i,* $D(a_i) = 9$*). Supposons que* $SPI(C)$ *possède un ordonnancement* `Quant IQuant IDCT` *tel que les durées d'exécution réelles et pire-cas soient données par le tableau de la figure 2.6, où* $a \in$ { `Quant`, `IQuant`, `IDCT` }.

Le choix des niveaux de qualité pour l'ordonnancement en utilisant la politique de gestion de qualité sûre $t_p^{sf} \geq t$ *n'est pas régulier (voir figure 2.6). En effet nous voyons que si nous déroulons le calcul pour chacune des actions de l'application, nous obtenons*

un niveau de qualité égale à 4 pour l'action QUANT et un niveau de qualité de 1 pour les actions suivantes.

Durées d'exécution

q	$C^{wc}(a,q)$	$C(a,q)$ $= C^{av}(a,q)$
1	1	1
2	5	2
3	6	3
4	7	5

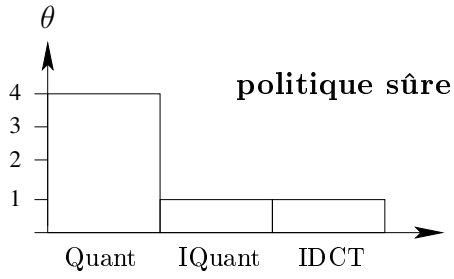


Figure 2.6 — Comportement d'une exécution contrôlée par la politique de gestion de qualité sûre

Dans ce qui suit, nous proposons d'améliorer le contrôleur afin d'obtenir une meilleure régularité des niveaux de qualité choisis par le contrôleur. Pour ce faire, nous modifions la politique de contrôle dans le but d'obtenir une meilleur prévision du comportement de l'application. Ainsi, la politique de contrôle sera non seulement basée sur une estimation des durées d'exécution pire-cas, mais également sur une estimation des durées d'exécution moyennes.

2.2.2 Politique de contrôle simple

La politique de contrôle simple est basée non seulement sur des estimations pire-cas des durées d'exécution, mais également sur des estimations moyennes des durées d'exécution. Ces estimations peuvent être obtenues par analyse statique et/ou par des techniques de *profiling* [39].

2.2.2.1 Politique de gestion de qualité simple

L'utilisation des durées d'exécution pire-cas est très pénalisante du point de vue des performances, mais aussi d'un point de vue de la régularité des niveaux de qualité (cf. section 2.2.1.2). Il est clair que nous ne pouvons pas nous passer des durées d'exécution pire-cas pour garantir le respect des échéances, mais il est tout de même possible d'utiliser d'autres estimations afin de mieux renseigner le contrôleur. Ainsi, les politiques présentées dans ce qui suit seront non seulement basées sur les durées d'exécution pire-cas, c'est-à-dire C^{wc}, mais également sur des estimations *moyennes*, notées C^{av}. Ces dernières représentent les durées d'exécution "probables" ou "typiques" des actions.

Définition 2.31 (fonction de durée d'exécution moyenne):
Soit $SPI(C) = (G, Q, C^{wc}, D, C)$, un système paramétré incertain. Nous appelons fonc-

tion de durée d'exécution moyenne une fonction de la forme $C^{av} : A \times Q \to \mathbb{R}^+$*, telle que :*

1. $C^{av} \leq C^{wc}$.
2. *Pour toute action* $a \in A$*, la fonction* $q \mapsto C^{av}(a, q)$ *est croissante.*

Nous pouvons à partir de cette fonction de durée d'exécution construire la politique de gestion de qualité moyenne t_p^{av}. Celle-ci est conservatrice par rapport aux durées d'exécution moyenne mais étant donné que nous n'avons pas $t_p^{av} \leq t_p^{sf}$, elle ne garantit pas que le contrôleur choisisse des qualités permettant le respect des échéances. Nous allons donc voir comment mélanger les durées d'exécution moyennes et pire-cas afin de satisfaire les trois critères auxquelles nous nous intéressons, c'est-à-dire le respect des échéances, une utilisation maximale des ressources mais aussi une bonne régularité des niveaux de qualité.

La politique de gestion de qualité simple est construite de façon à prendre en compte à la fois le comportement pire-cas de l'application, mais aussi son comportement moyen. La combinaison de ces deux comportements permet d'améliorer la régularité des niveaux de qualité choisis par le Gestionnaire de Qualité. La politique de gestion de qualité simple, définie ci-après, peut donc être utilisée pour améliorer cette régularité tout en respectant les contraintes de temps du fait de l'utilisation de la politique de gestion de qualité sûre.

Définition 2.32 (Fonction de durée d'exécution *simple*):
Soit un système paramétré $SPI(C) = (G, Q, C^{wc}, D, C)$ *et une fonction de durée d'exécution moyenne* $C^{av} : A \times Q \to \mathbb{R}^+$*, la politique de gestion de qualité simple* t_p^{sp} *correspond à la fonction de durée d'exécution simple* $C^{sp} : A^+ \times Q \to \mathbb{R}^+$ *définie par :*

$$C^{sp} = \mathbf{max} \{ C^{sf}, C^{av} \} .$$

Proposition 2.33:
La politique de gestion de qualité simple t_p^{sp} *satisfait* $t_p^{sp} = \min \{ t_p^{sf}, t_p^{av} \}$.

Preuve (de la proposition 2.33):
Pour tout $i \in \{ 0, \ldots, n-1 \}$ *et* $q \in Q$*, nous avons :*

$$\begin{aligned}
t_p^{sp}(a_i \ldots a_n, s_{i-1}, q) &= \min_{\mathbf{i \leq k \leq n}} D(a_k) - C^{sp}(a_i \,..\, a_k, q) \\
&= \min_{\mathbf{i \leq k \leq n}} D(a_k) \; - \mathbf{max} \{ C^{av}(a_i \,..\, a_k, q) \;,\; C^{sf}(a_i \,..\, a_k, q) \} \\
&= \min_{\mathbf{i \leq k \leq n}} \min \{ D(a_k) - C^{av}(a_i \,..\, a_k, q) \;,\; D(a_k) - C^{sf}(a_i \,..\, a_k, q) \} \\
&= \min \{ \min_{\mathbf{i \leq k \leq n}} (D(a_k) - C^{av}(a_k, q)) \;, \\
&\qquad \min_{\mathbf{i \leq k \leq n}} (D(a_k) - C^{sf}(a_i \,..\, a_k, q)) \} \\
t_p^{sp}(s_{i-1}, q) &= \min \{ t_p^{av}(a_i \ldots a_n, s_{i-1}, q), t_p^{sf}(a_i \ldots a_n, s_{i-1}, q) \} .
\end{aligned}$$

□

A l'instar de la politique de gestion de qualité sûre, la politique de gestion de qualité simple n'est conservatrice ni par rapport à la fonction de durée d'exécution moyenne, ni par rapport à la fonction de durée d'exécution pire-cas.

Exemple 2.8: *Considérons un système dans lequel, pour toutes les actions $a \in A$ et tous les niveaux de qualité $q \in Q = \{0, 1\}$, nous avons $C^{av}(a, q) = q + 1$, $C^{wc}(a, q) = 2q + 1$ et $D(a) = 4$. Calculons d'abord $t_p^{sp}(a_1a_2, s_0, 1)$:*

$$\begin{aligned} t_p^{sp}(a_1a_2, s_0, 1) &= D(a_2) - \mathbf{max}\{C^{av}(a_1a_2, 1), C^{sf}(a_1a_2, 1)\} \\ &= D(a_2) - \mathbf{max}\{C^{av}(a_1a_2, 1), C^{wc}(a_1, 1) + C^{wc}(a_1, 0)\} \\ &= D(a_2) - \mathbf{max}\{4, 3 + 1\} \\ &= 4 - \mathbf{max}\{4, 4\} = 0 \ . \end{aligned}$$

Ensuite, nous calculons $t_p^{sp}(a_2, s_1, 1)$:

$$\begin{aligned} t_p^{sp}(a_2, s_1, 1) &= D(a_2) - \mathbf{max}\{C^{av}(a_2, 1), C^{sf}(a_2, 1)\} \\ &= 4 - \mathbf{max}\{2, 3\} = 1 < 2 = C^{av}(a_1, 1) \ . \end{aligned}$$

Ainsi, la politique de gestion de qualité simple n'est pas conservatrice par rapport à la fonction de durée d'exécution moyenne. Puisque $C^{av} \leq C^{wc}$, elle n'est pas non plus conservatrice par rapport à C^{wc}.

2.2.2.2 Les limitations de la politique simple

La politique de contrôle simple présente l'intérêt de tenir compte à la fois d'un critère de sûreté, du fait de l'utilisation de la fonction des durées d'exécution pire-cas, et d'un critère de régularité des niveaux de qualité, du fait de l'utilisation des durées d'exécution moyennes. Cependant, elle peut conduire dans certains cas à des affectations de qualité très irrégulières, et en particulier à une réduction du niveau de qualité avant une échéance critique.

Exemple 2.9: *En reprenant l'exemple 2.7, nous pouvons voir que l'ordonnancement calculé en utilisant la politique de gestion de qualité simple, $t_p^{sf} \geq t$ donne des affectations de qualité plus régulières que la politique de gestion de qualité sûre.*

Ainsi, le contrôleur Γ^{sp} choisit parfois un niveau de qualité q à un instant donné même s'il n'est pas capable de tenir ce niveau de qualité par la suite, bien que les durées d'exécution réelles soient égales ou même inférieures aux durées d'exécution moyennes. Nous considérons que ce phénomène n'est pas acceptable dans la mesure où, les durées d'exécutions moyennes étant connues d'avance, le contrôleur devrait être capable d'anticiper et choisir des niveaux de qualité réguliers lorsque les durées d'exécution réelles s'approchent des durées moyennes.

Nous allons dans la section suivante présenter la politique de gestion de qualité mixte

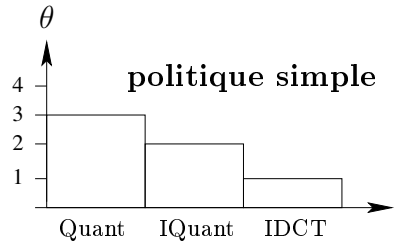


Figure 2.7 — Comportement d'une exécution contrôlée par la politique de gestion de qualité simple.

qui permet de palier à ce phénomène en combinant elle aussi les durées d'exécution moyennes et pire-cas.

2.2.3 Politique de contrôle mixte

Nous présentons ici la politique de gestion de qualité mixte. A l'instar de la politique de gestion de qualité simple, la politique mixte est construite à partir d'un mélange entre les estimations pire-cas et les estimations moyennes, et plus précisément entre les fonctions C^{av} et C^{sf}. Cependant, ce mélange est réalisé de façon beaucoup plus fin dans le cas de la politique mixte. La figure 2.8 illustre la différence entre le calcul de la fonction C^{sp} et le calcul de la fonction C^{mx}. Nous obtenons de la sorte une meilleure estimation du comportement de l'application, et donc une plus grande régularité des niveaux de qualité choisis par le contrôleur. La figure 2.8 compare la construction de la politique de contrôle simple et celle de la politique de contrôle mixte. Contrairement à la politique simple qui choisit le maximum entre la durée d'exécution sûre et la durée d'exécution moyenne pour l'action courante, la politique simple fait une estimation de ce qu'il peut arriver par la suite. En effet la politique mixte effectue un mélange entre les durées d'exécution sûres et moyennes pour toutes les actions restant à exécuter dans l'ordonnancement. C'est à dire que la politique mixte choisira le maximum entre la durée d'exécution sûre, la durée d'exécution moyenne et des mélanges entre celles-ci.

Pour construire la politique de gestion de qualité mixte, nous introduisons d'abord la fonction de durée d'exécution mixte C^{mx}. Celle-ci combine l'utilisation des deux fonctions de durée d'exécution C^{sf} et C^{av}. Cette dernière est utilisée pour améliorer la régularité dans le choix des niveaux de qualité.

Définition 2.35 (Marge de sécurité):
Nous introduisons δ^{max} comme le maximum de la différence entre le comportement pire-cas et le comportement moyen, c'est-à-dire :

$$\delta^{max}(a_i \,..\, a_k, q) = \mathbf{max}_{i \leq j \leq k}\ \delta(a_j \,..\, a_k, q),$$

où $\delta(a_j \,..\, a_k, q) = C^{sf}(a_j \,..\, a_k, q) - C^{av}(a_j \,..\, a_k, q)$.

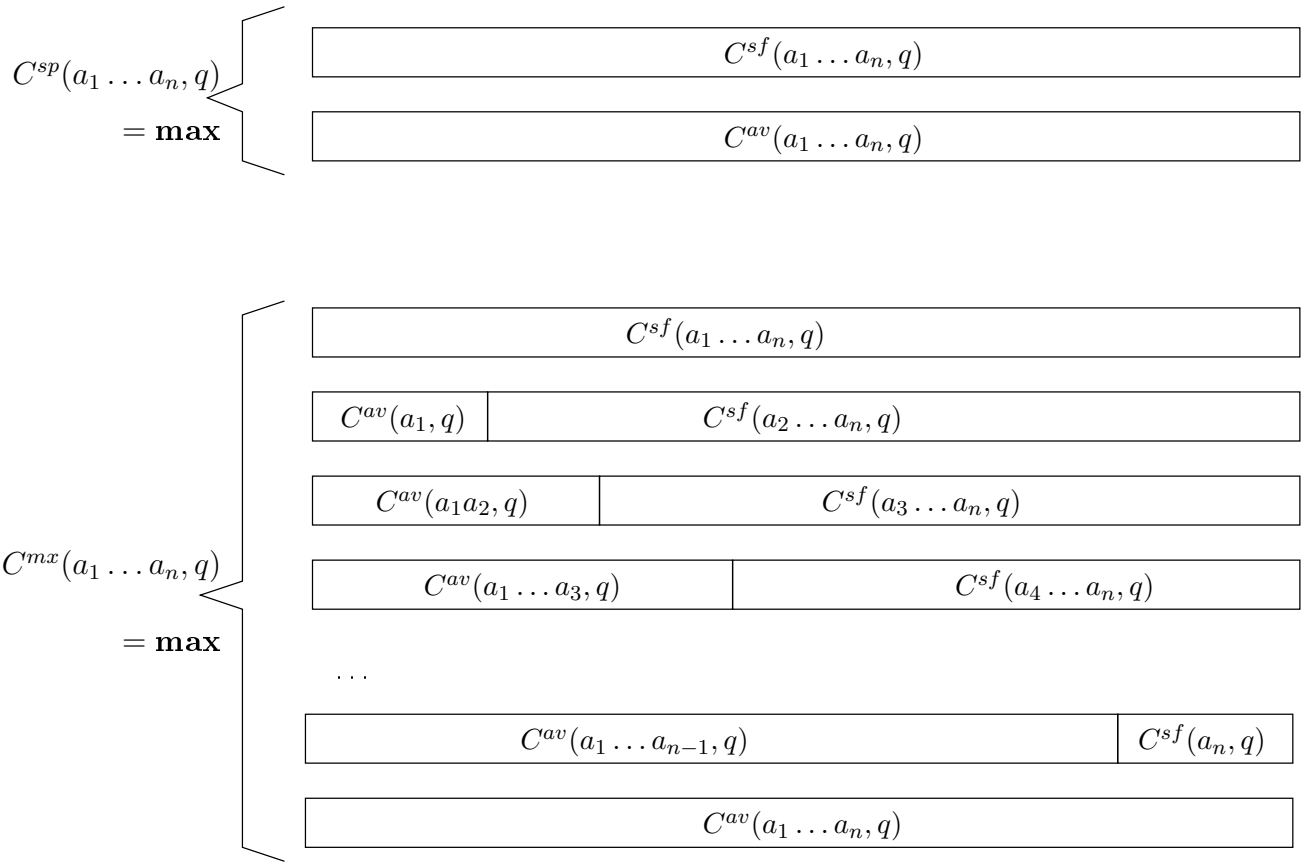


Figure 2.8 — Comparaison entre le calcul des politiques de contrôle mixte et simple.

Exemple 2.10: *Soit un ordonnancement $\alpha = abcdefg$ et $q > q_{min}$ un niveau de qualité, tels que les durées d'exécution des actions sont données par la figure 2.9. Dans ce cas, les fonctions $k \mapsto \delta(a_k \dots a_n, q)$ et $k \mapsto \delta^{max}(a_k \dots a_n, q)$ sont représentées dans la figure 2.10.*

action x	$C^{wc}(x,q) - C^{av}(x,q)$	$C^{wc}(x,q_{min}) - C^{av}(x,q)$
a	100	-10
b	10	-2
c	10	-4
d	80	-10
e	10	-10
f	20	-30
g	10	-10

Figure 2.9 — Durées d'exécution.

Notons que, pour une séquence d'action a_i .. a_k et un niveau de qualité q, $\delta^{max}(a_i$ .. $a_k, q)$ est une sorte de marge de sécurité dont le contrôleur a besoin pour respecter les échéances. Ceci est dû à l'incertitude sur les durées d'exécution.

Définition 2.36 (Fonction de durée d'exécution mixte):
*Nous appelons **fonction de durée d'exécution mixte** la fonction partielle $C^{mx} : A^* \times \Theta \to \mathbb{R}^+$ définie de la façon suivante :*

$$C^{mx} = C^{av} + \delta^{max} .$$

Définition 2.37 (Politique de gestion de qualité mixte):
*La **politique de gestion de qualité mixte** est définie par la fonction partielle $t_p^{mx} : A^* \times \Theta \to \mathbb{R}$ associée à la fonction de durée d'exécution mixte C^{mx}.*

La politique de gestion de qualité mixte est bien entendu basée sur la fonction de durée d'exécution mixte. Cette dernière est issue d'un mélange à grain fin entre les fonctions de durée d'exécution moyennes et sûres. Le but de ce mélange est d'anticiper le surcoût induit par la sûreté, par rapport au comportement moyen de l'application. Ce surcoût est mesuré par la fonction δ^{max}, dont nous avons donné la définition ci-dessus.

La politique de gestion de qualité simple n'est par conservatrice par rapport aux estimations moyennes, ce qui explique le manque de régularité des niveaux de qualité obtenus parfois avec le contrôleur Γ^{sp}. La politique de gestion de qualité mixte est, quant à elle, conservatrice par rapport aux durées d'exécution moyennes. Elle conduira à une meilleure régularité des niveaux de qualité.

Exemple 2.11: *En reprenant l'exemple 2.7, nous pouvons voir que l'ordonnancement calculé en utilisant la politique de gestion de qualité mixte, $t_p^{mx} \geq t$ donne des affectations de qualité plus régulières que les deux autres politiques de gestion de qualité.*

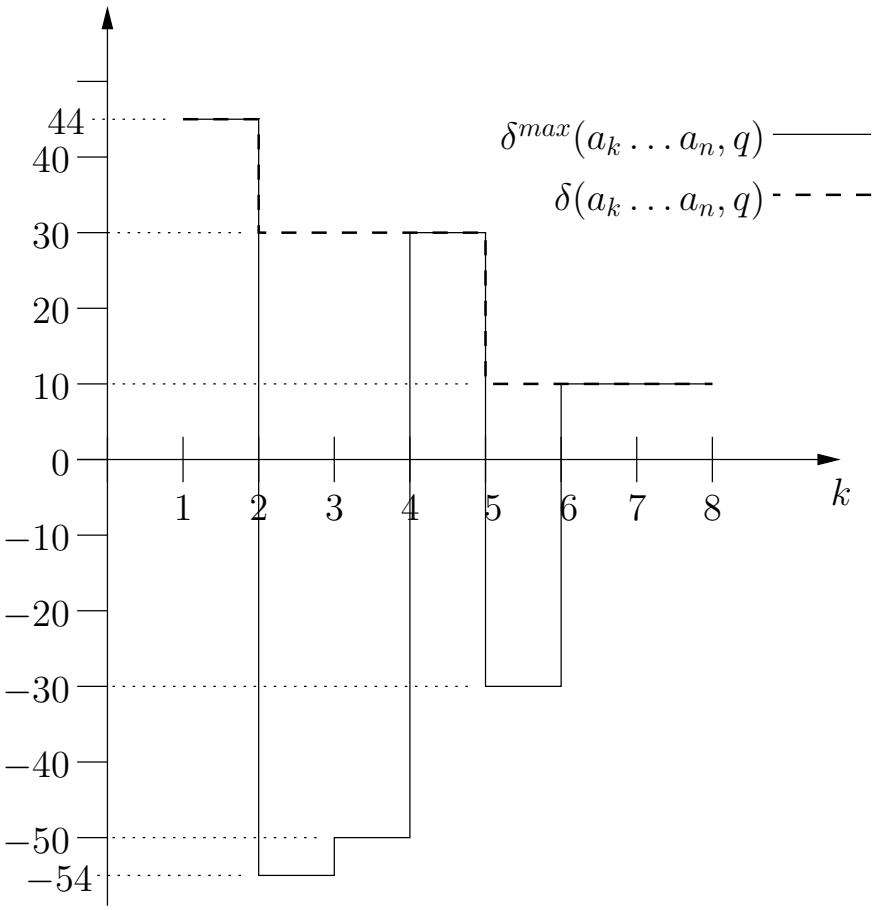


Figure 2.10 — Fonctions $k \mapsto \delta(a_k \ldots a_n, q)$ et $k \mapsto \delta^{max}(a_k \ldots a_n, q)$.

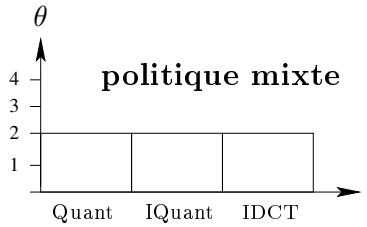


Figure 2.11 — Comportement d'une exécution contrôlée par la politique de gestion de qualité mixte.

2.3 Conclusion

Dans ce chapitre nous avons présenté un problème de contrôle de qualité de service. Il s'agit à partir d'un système constitué d'actions paramétrées par des niveaux de qualité, de contraintes de précédence et de temps, de fournir un ordonnancement et un choix de qualité satisfaisant des propriétés. Le premier énoncé de ce problème était basé sur la connaissance des durées d'exécution des actions.

Nous l'avons ensuite étendu pour le présenter dans un contexte plus réaliste et plus proche des application que nous étudions. En effet dans le cadre des applications multimédia, la durée d'exécution des actions n'est pas prévisible, elle est influencée par de nombreux phénomènes qui la rende relativement aléatoire. Nous avons donc présenté un problème où les durées d'exécution ne sont pas connues mais bornées par un pire-cas.

Pour le résoudre nous avons considérer trois *politiques de contrôle*. La première, la politique *sûre*, permet de satisfaire uniquement aux exigences des propriétés de sûreté. Ensuite nous avons présenté la politique *simple* correspondant à un contrôle non seulement sûre mais aussi optimal. Nous avons montré que ces deux politiques n'étaient pas satisfaisantes et qu'elles possédaient des limites. C'est la raison pour laquelle, nous avons proposé la politique de contrôle *mixte*.

Cette technique permet à la fois de répondre aux propriétés de sûreté et d'optimalité, mais permet aussi d'avoir une régularité au niveau du choix des niveaux de qualité. Celle-ci est basée sur une combinaison fine des durées d'exécution moyennes et pire-cas. Elle permet d'obtenir une meilleure estimation des durées d'exécution réelles. Elle est aussi basée sur la construction d'une marge de sécurité lui permettant de ne pas réduire subitement le niveau de qualité.

Du fait de l'utilisation de cette marge et de sa méthode de combinaison des durées d'exécution, cette technique n'est pas facilement interprétable. En effet le comportement de l'application contrôlée n'est pas intuitif. De plus, son calcul est complexe et le fait d'avoir un contrôle à grain fin mène à un surcoût du contrôleur trop important pour être appliqué directement.

Dans la suite, nous allons proposer un outil permettant de comprendre le comportement d'une application contrôlée par la politique mixte. Puis nous verrons à partir de cet outil comment réduire le surcoût lié à cette politique.

CHAPITRE

Contrôle symbolique de qualité de service

3

Le chapitre 2 a permis de définir non seulement les bases du problème de contrôle de qualité de service mais aussi d'en donner une solution. La politique mixte permet de répondre aux trois propriétés, la sûreté, l'optimalité et la régularité des affectations de niveau de qualité. Cependant cette politique n'est pas intuitive, et donc sa compréhension reste difficile. En effet le mélange entre les durées d'exécution pire-cas et moyennes est réalisé à grains fins mais quel comportement cela induit-il ? Comment l'application se comporte-t-elle réellement ? Ceci est une partie des questions qui reste en suspens et qui bloque donc l'utilisation d'une telle technique de contrôle. De plus le calcul des durées d'exécution mixtes est coûteux en terme de temps. En effet il faut calculer les marges de sécurité pour chacune des actions restantes. Etant donné que le contrôle est effectué pour chaque action, ce calcul sera réalisé lui aussi à chaque action et cela entraînera un surcoût du contrôleur pour l'exécution de l'application non négligeable.

La politique de contrôle est composée de deux actions différentes, la première est l'ordonnancement et la deuxième concerne l'affectation de qualité. Dans ce chapitre nous nous focaliserons sur la deuxième partie à savoir comment le niveau de qualité est choisi. Pour les deux principales raisons que nous venons de citer, ce chapitre va proposer d'étudier une représentation graphique à deux dimensions, le *diagramme des vitesses*. Celui-ci permet de schématiser l'évolution du comportement de l'application contrôlée et d'observer les choix faits par le gestionnaire de qualité. A partir de cette représentation graphique nous donnerons une interprétation de la politique de contrôle mixte.

La suite de ce chapitre sera consacrée à la réduction du surcoût lié au contrôleur. Nous avons imaginé une technique qui à partir des diagrammes de vitesses choisit un niveau de qualité pour les actions suivantes. Nous avons pu définir des régions où le gestionnaire de qualité choisit un niveau de qualité constant. En fonction de ces régions, nous verrons qu'il est assez aisé d'observer quel sera le comportement du contrôleur en fonction de l'état du système. Nous proposerons ainsi un algorithme de contrôle symbolique qui au lieu de calculer les fonctions de durées d'exécution mixtes comparera le temps réel de l'application avec des bornes modélisant les *régions de qualité*.

Nous avons observé à partir de ces régions que dans la plupart des exécutions, il est possible que le contrôleur choisisse le même niveau de qualité. La dernière partie proposera donc une méthode permettant de construire sur le diagramme des vitesses des *régions de relâchement*. C'est-à-dire des régions où le Gestionnaire de Qualité choisit un niveau de qualité constant pour un certain nombre d'actions. Nous proposerons pour finir un algorithme de relâchement du contrôleur basé sur des comparaisons entre le temps réel

de l'application et les bornes des zones de relâchement.

Les propositions de ce chapitre ont donné lieu à un article [40] et à une partie d'un papier journal [41].

Dans ce chapitre, nous nous intéressons au choix du gestionnaire de qualité. Sans perte de généralité, nous considérerons un ordonnancement fixe $a_1 \dots a_n$. Etant donné un système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$ et une séquence d'exécution $(s_i, t_i) \xrightarrow{a_i, q_i} (s_{i+1}, t_{i+1})$ les composantes de s_i sont uniquement déterminés par a (cf. chapitre 2).

3.1 Diagramme des Vitesses

La technique de contrôle énumérative précédente introduit beaucoup de données difficiles à interpréter. En effet beaucoup d'éléments nouveaux s'ajoutent au problème. Nous pouvons citer la marge de sécurité (cf. définition 2.35), tout comme la combinaison de cette marge avec la durée d'exécution moyenne. Plusieurs questions viennent donc perturber la compréhension de l'utilisateur :

- À quoi correspond la marge de sécurité introduite précédemment ?
- Quel est le comportement d'une application contrôlée avec cette politique de gestion de qualité mixte ?
- Quelle interprétation peut on donner de la politique de gestion de qualité mixte ?
- L'utilisation d'une marge de sécurité permet elle de respecter la propriété d'optimalité ?

Pour répondre à ces questions, nous avons décidé de donner un outil nous permettant de faire une analyse de cette politique. Nous proposons donc un **diagramme de vitesses** qui est une représentation graphique de l'espace d'état de l'application contrôlée. Cette représentation graphique va nous permettre de modéliser les choix du Gestionnaire de Qualité pour la politique mixte. Pour cela nous allons dans cette section présenter la manière dont est construit ce diagramme. Premièrement nous définirons la représentation graphique, puis nous introduirons la notion de vitesses. Ensuite nous donnerons une interprétation des choix du Gestionnaire de Qualité.

3.1.1 Définition du diagramme des vitesses

Les diagrammes de vitesses représentent dans un espace à deux dimensions l'évolution d'un système paramétré $SPI(C) = (G, Q, C^{wc}, D, C)$ et son Gestionnaire de Qualité dirigé par la politique *mixte*. L'axe des abscisses correspond au temps d'exécution réel, c'est-à-dire à l'évolution du temps au cours de l'exécution de l'application. L'axe des ordonnées représente le temps virtuel calculé à partir du temps d'exécution moyen et des échéances. Ce temps virtuel correspond à l'évolution à l'intérieur de l'application.

Dans ce qui suit nous allons définir formellement les diagrammes de vitesses mais

aussi donner quelques résultats sur l'interprétation de la politique *mixte* en termes de vecteurs de vitesse.

3.1.1.1 Représentation de l'état du système

Soient $SPI(C) = (G, Q, C^{wc}, D, C)$ un système paramétré incertain, et Γ^{mx} un contrôleur de $SPI(C)$ basé sur la politique de contrôle mixte. Soit (s_i, t_i) un état de $SPI(C)||\Gamma^{mx}$. Soient $q \in Q$ un niveau de qualité, et k l'indice de l'échéance critique $D(a_k)$ d'une action dans la séquence d'actions restantes $a_{i+1}, \ldots, a_k, \ldots, a_n$.

Nous allons tracer le diagramme des vitesses correspondant à l'échéance $D(a_k)$ et au niveau de qualité q. Il s'agit d'une représentation graphique en deux dimensions des états possibles du système, et des choix du contrôleur en fonction de ces états. L'abscisse du diagramme correspond à l'avancée réelle du temps, ou temps-réel t. Nous avons forcément $D(a_k) \geq t$. L'ordonnée du diagramme correspond à un temps théorique, basé sur le modèle des durées d'exécution moyennes. Ce temps théorique permet de mesurer l'avancée de l'application dans les calculs. Nous effectuons une normalisation par rapport à l'échéance $D(a_k)$, de telle sorte que le diagramme ait une forme de carré de côté $D(a_k)$. Plus précisément, l'état (s_i, t_i) correspond à la position $(t_i, y_i(q))$ dans le diagramme des vitesses, où $y_i(q)$ est donné par :

$$y_i(q) = \frac{C^{av}(a_1 \,..\, a_i, q)}{C^{av}(a_1 \,..\, a_k, q)} \cdot D(a_k) \tag{3.1}$$

Intuitivement, $y_i(q)$ est le ratio de temps consommé à l'état s_i par rapport au budget de temps disponible $D(a_k)$. Notons que la normalisation par rapport à l'échéance implique que $y_k(q) = D(a_k)$ (voir la figure 3.1).

Ainsi, le diagramme des vitesses relie l'avancée réelle du temps à l'avancée théorique du temps : nous pouvons notamment dire que la diagonale du graphe correspond au comportement optimal ; mais aussi que les points $(t_i, y_i(q))$ sous la diagonale correspondent aux états où l'exécution est jugée en retard par rapport au temps virtuel. Et inversement, pour les points au dessus de la diagonale, nous considérons que l'application est en avance.

Définition 3.1:
Soient (s_i, t_i) et (s_j, t_j) deux états de $SPI(C) = (G, Q, C^{wc}, D, C)$ tel que $j > i$. Considérons leurs positions correspondantes $(t_i, y_i(q))$ et $(t_j, y_j(q))$ sur le diagramme des vitesses pour une qualité q et une contrainte de temps $D(a_k)$, $k \geq j$. Le ***vecteur de vitesse*** *$v_{i,j}(q)$ entre $(t_i, y_i(q))$ et $(t_j, y_j(q))$ est donné par la relation :*

$$v_{i,j}(q) = \frac{y_j(q) - y_i(q)}{t_j - t_i} \,.$$

Exemple 3.1: *La figure 3.1 nous montre la manière dont on peut représenter l'évolution de l'application. Nous pouvons comprendre sur cet exemple la technique utilisée pour contrôler*

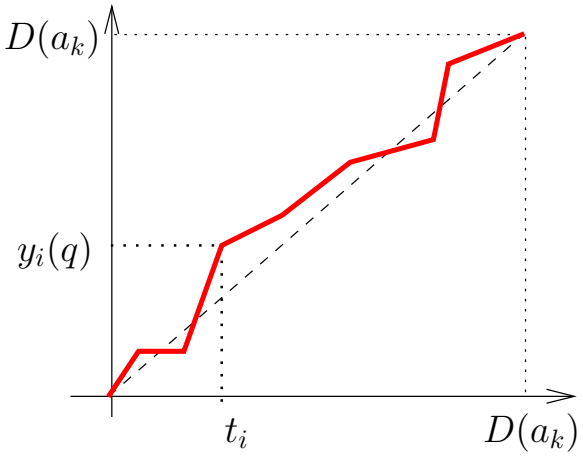


Figure 3.1 — Représentation du diagramme de vitesses pour la qualité q et pour l'échéance $D(a_k)$

une application dans le cadre général. En effet nous avons tracé la diagonale, et intuitivement nous voulons obtenir un comportement qui se rapproche le plus possible de celle-ci. Prenons l'exemple du point $(t_i, y_i(q))$, le contrôleur va observer la position de ce point et adapter la future qualité en fonction de sa position par rapport à la diagonale. Dans ce cas précis, il est au dessus de la diagonale donc en avance, le contrôleur va alors essayer de ralentir l'application.

Cet exemple nous permet de comprendre intuitivement ce que fait un contrôleur d'application dans le cas général. Nous allons, dans les deux sections suivantes, donner deux définitions de vitesses permettant d'expliquer plus en détails le fonctionnement de la politique de gestion de qualité mixte.

3.1.1.2 Vecteur de vitesse idéale

Pour un niveau de qualité q et une position $(t_i, y_i(q))$ dans le diagramme des vitesses, le vecteur de vitesse idéale $v^{idl}(q)$ correspond à la vitesse du système dans le cas où les durées d'exécution réelles sont exactement égales aux durées d'exécution moyennes C^{av}. Pour un niveau de qualité q donné, la vitesse idéale $v^{idl}(q)$ ne dépend pas de la position courante $(t_i, y_i(q))$, mais uniquement de l'échéance critique considérée. Ainsi, tant que l'échéance critique ne change pas, les vitesses idéales restent les mêmes. Elles correspondent à des valeurs intrinsèques du modèle de l'application.

Définition 3.2:
Soit $SPI(C) = (G, Q, C^{wc}, D, C)$, un système paramétré incertain. Supposons que $C = C^{av}$ et $q_{i+1} = \ldots = q_j = q$. Nous avons $t_j - t_i = C(a_1 \,..\, a_j, q) - C(a_1 \,..\, a_i, q) = C(a_{i+1} \,..\, a_j, q) = C^{av}(a_{i+1} \,..\, a_j, q) = C^{av}(a_1 \,..\, a_j, q) - C^{av}(a_1 \,..\, a_i, q)$. Soit k l'indice de l'échéance critique des actions $k \geq j \geq i$. Soit k l'indice de l'échéance critique des actions $k \geq j \geq i$.

Alors le ***vecteur de vitesse idéale*** $v^{idl}(q)$ *entre* $(t_i, y_i(q))$ *et* $(t_j, y_j(q))$ *est égal à :*

$$\begin{aligned} v^{idl}(q) &= \frac{y_j(q) - y_i(q)}{t_j - t_i} \\ &= \frac{D(a_k)}{C^{av}(a_1 \,..\, a_k, q)} \cdot \frac{C^{av}(a_1 \,..\, a_j, q) - C^{av}(a_1 \,..\, a_i, q)}{C^{av}(a_1 \,..\, a_j, q) - C^{av}(a_1 \,..\, a_i, q)} \text{ d'après 3.1} \\ &= \frac{D(a_k)}{C^{av}(a_1 \,..\, a_k, q)} . \end{aligned}$$

Le vecteur de vitesse idéale $v^{idl}(q)$ est indépendant du choix de i et j. Il dépend seulement de l'échéance cible $D(a_k)$ et du niveau de qualité q. Cela signifie que pour une affectation de qualité constante, la trajectoire du système sur le diagramme est linéaire dans un comportement idéal, c'est-à-dire $C = C^{av}$.

3.1.1.3 Vecteur de vitesse optimale

Pour un niveau de qualité q et une position $(t_i, y_i(q))$ dans le diagramme des vitesses, le vecteur de vitesse optimale $v^{opt}(q)$ est obtenu comme le vecteur de vitesse entre la position courante $(t_i, y_i(q))$ et le point de coordonnées $(D(a_k) - \delta^{max}(a_{i+1} \,..\, a_k, q), D(a_k))$. En ayant pour cible le point $(D(a_k) - \delta^{max}(a_{i+1} \,..\, a_k, q), D(a_k))$ au lieu de $(D(a_k), D(a_k))$ le Gestionnaire de Qualité respecte la marge de sécurité $\delta^{max}(a_{i+1} \,..\, a_k, q)$ qui est suffisante pour assurer la terminaison de l'application avant l'échéance $D(a_k)$. La valeur $\delta^{max}(a_{i+1} \,..\, a_k, q)$ est une marge de sécurité caractérisée par le compromis entre la faisabilité et l'optimalité pour la politique de gestion de qualité mixte.

La figure 3.2 donne un exemple de diagramme des vitesses, dans lequel les vitesses idéales et optimales ont été représentées pour le niveau de qualité q. Il s'agit donc d'un comportement optimal dans la mesure où il correspond à une qualité constante et une utilisation maximale des ressources, c'est-à-dire que l'action $\alpha_q(k)$ termine exactement à la date $D(\alpha_q(k))$ qui est son échéance.

Définition 3.3:
Soit $SPI(C) = (G, Q, C^{wc}, D, C)$*, un système paramétré incertain. Nous appelons* ***vitesses optimales sûres*** *à l'état* (s_i, t_i) *de* $SPI(C)$*, par rapport à l'échéance* $D(a_k)$ *pour* $k \geq i$*, les vitesses* $v^{opt}(q)$ *définies pour tout niveau de qualité* q *par :*

$$\begin{aligned} v^{opt}(q) &= \frac{D(a_k)}{C^{av}(a_1 \,..\, a_k, q)} \cdot \frac{C^{av}(a_{i+1} \,..\, a_k, q)}{D(a_k) - \delta^{max}(a_{i+1} \,..\, a_k, q) - t_i} \\ &= v^{idl}(q) \cdot \frac{C^{av}(a_{i+1} \,..\, a_k, q)}{D(a_k) - \delta^{max}(a_{i+1} \,..\, a_k, q) - t_i} \end{aligned}$$

Les définitions du diagramme de vitesses et des vecteurs de vitesses optimales et idéales, vont nous permettre de donner une interprétation de la politique de gestion de qualité mixte. Nous allons dans la partie suivante donner une proposition permettant de

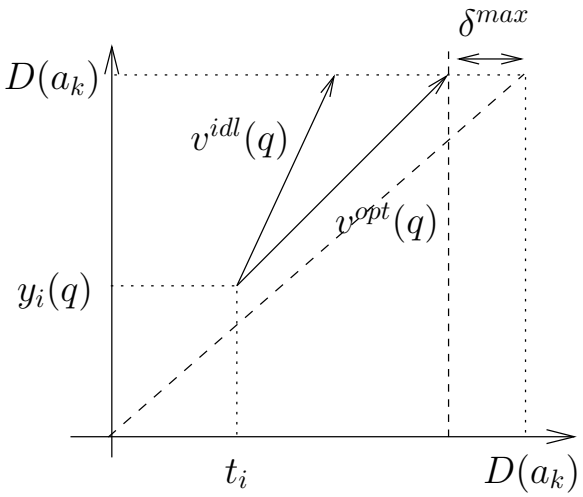


Figure 3.2 — Représentation des vecteurs de vitesses idéales et optimales

comparer les vecteurs de vitesses optimales et idéales. Cette comparaison nous aidera à comprendre le mécanisme de gestion de qualité pour la politique mixte.

3.1.2 Interprétation de la politique *mixte*

L'intérêt de cette représentation graphique, nous allons le voir, réside dans la compréhension du comportement du contrôleur. En effet, les définitions précédentes vont nous permettre d'expliquer les choix de qualité visuellement. La proposition suivante donne une interprétation de la politique mixte en termes de comparaison entre les vitesses idéales et les vitesses optimales.

La condition d'admissibilité $t_p^{mx} \geq t$ associée à la politique de gestion de qualité mixte peut s'exprimer simplement en termes de comparaison entre la vitesse idéale $v^{idl}(q)$ et la vitesse optimale $v^{opt}(q)$ associée. Plus précisément, la condition d'admissibilité est vraie si et seulement si la vitesse idéale $v^{idl}(q)$ est supérieure ou égale à la vitesse optimale associée au niveau $v^{opt}(q)$. Ce résultat est intéressant dans la mesure où il permet de traduire les choix de niveau de qualité obtenus avec la politique de gestion de qualité mixte, directement dans le diagramme des vitesses. Par exemple, dans la figure 3.2, l'état du système correspondant à la position $(t, y(q))$ est tel que le niveau de qualité q est admissible. En effet, sur cette figure, la vitesse idéale $v^{idl}(q)$ est supérieure à la vitesse optimale $v^{opt}(q)$.

Proposition 3.4:
Soient un système paramétré $SPI(C) = (G, Q, C^{wc}, D, C)$, un état (s_i, t_i) de $SPI(C)$, un niveau de qualité q et une échéance cible $D(a_k)$, avec $k > i$, nous avons :

$$v^{idl}(q) \geq v^{opt}(q) \iff D(a_k) - C^{mx}(a_{i+1} \,..\, a_k, q) \geq t_i \,.$$

Preuve (de la proposition 3.4):
Nous avons :

$$
\begin{aligned}
& v^{idl}(q) \geq v^{opt}(q) \\
\iff\ & v^{idl}(q) \geq v^{idl}(q) \cdot \frac{C^{av}(a_{i+1} .. a_k, q)}{D(a_k) - \delta^{max}(a_{i+1} .. a_k, q) - t_i} \\
\iff\ & 1 \geq \frac{C^{av}(a_{i+1} .. a_k, q)}{D(a_k) - \delta^{max}(a_{i+1} .. a_k, q) - t_i} \\
\iff\ & D(a_k) - \delta^{max}(a_{i+1} .. a_k, q) - t_i \geq C^{av}(a_{i+1} .. a_k, q) \\
\iff\ & D(a_k) - \delta^{max}(a_{i+1} .. a_k, q) - C^{av}(a_{i+1} .. a_k, q) \geq t_i \\
\iff\ & D(a_k) - C^{mx}(a_{i+1} .. a_k, q) \geq t_i \ . \ \text{d'après la définition 2.36, } C^{mx} = C^{av} + \delta^{max}
\end{aligned}
$$

□

Dans le cas de la politique d'ordonnancement statique, et tant que l'échéance critique ne change pas, la proposition 3.4 nous permet de représenter l'ensemble des positions $(t, y(q))$ tel que le niveau de qualité q soit admissible par rapport à la politique de gestion de qualité mixte. Nous utilisons la vitesse optimale, qui ne correspond pas à une utilisation maximale des ressources, mais de $D(a_k) - \delta^{max}$ unités de temps sur les $D(a_k)$ disponibles. Ainsi, plus la valeur de δ^{max} sera petite, plus l'utilisation des ressources sera proche du maximum. De la sorte nous faisons apparaître le fait que le respect des échéances limite inévitablement la maximisation du budget de temps. Le contrôleur "prévoit" de perdre δ^{max} unités de temps, si les durées d'exécution réelles sont identiques aux durées d'exécution moyennes. Bien entendu, selon les durées d'exécution du système, la perte réelle peut être différente de cette valeur.

Intuitivement, nous voyons dans le diagramme des vitesses que la valeur δ^{max} est caractéristique dans la politique de gestion de qualité mixte. En effet, elle mesure la différence entre le comportement pire-cas et le comportement moyen, et a donc un lien avec l'incertitude qui existe sur les durées d'exécution réelles. Nous allons voir que deux facteurs limitent l'utilisation du budget de temps. Le premier est justement l'incertitude sur les durées d'exécution ; il sera mesuré grâce à la fonction δ^{max}. Le second concerne la granularité de contrôle. En effet, plus la granularité de contrôle est fine et plus le contrôleur sera en mesure de maximiser le budget de temps en s'approchant au plus près des échéances. Dans le diagramme, ceci s'interprète par une meilleure approximation de la vitesse idéale par les niveaux de qualité discrets du système paramétré incertain.

Le diagramme des vitesses nous permet d'avoir une vue globale du comportement de l'application contrôlée. Nous pouvons à partir de cette représentation graphique savoir quel niveau de qualité sera choisi pour les prochaines actions, mais aussi permettre une meilleur compréhension de la politique de gestion de qualité mixte en ce qui concerne le budget de temps. Nous allons dans ce qui suit nous intéresser à la modélisation des

niveaux de qualité dans cette représentation et ainsi proposer une technique de contrôle basée sur celle-ci.

3.2 Définition des Régions de Qualité

Le diagramme des vitesses défini précédemment nous permet de suivre le comportement de l'application contrôlée pour chaque état. Dans cette section nous allons proposer une représentation plus précise des choix du contrôleur. Cette représentation nous conduira à une étude symbolique de la technique de contrôle.

Le diagramme des vitesses doit être construit pour chaque état de $SPI(C)||\Gamma^{av}$, et pour chaque niveau de qualité. Dans le reste du chapitre nous allons considérer un cas particulier dans lequel nous pourrons comparer les différents états du système et les différentes qualités sur un unique diagramme des vitesses.

3.2.1 Découpage du diagramme des vitesses en régions de qualité

La représentation graphique nous a permis de comprendre la manière dont est fait le choix des niveaux de qualité. Ces choix sont réalisés en termes de comparaison entre les vecteurs de vitesses optimales et les vecteurs de vitesses idéales. Dans ce qui suit nous allons montrer qu'il est possible de modéliser des régions de qualité. Ces dernières représentent des zones du diagramme des vitesses où l'on sait que le contrôleur choisit des niveaux de qualité constants.

Définition 3.6:
Soient $SPI(C) = (G, Q, C^{wc}, D, C)$ un système paramétré incertain et Γ^{mx} un contrôleur utilisant la politique d'ordonnancement mixte. Une ***région de qualité*** *$\mathcal{R}_q$ pour un niveau de qualité q est un sous-ensemble d'états (s_i, t_i) de $SPI(C)$ défini par :*

$$\mathcal{R}_q = \left\{ \ (s_i, t_i) \mid \Gamma^{mx}(s_i, t_i) = (a_{i+1}, q) \ \right\} .$$

Intuitivement, sur la figure 3.3, nous voyons que cette région est délimitée par des frontières. Soient (s_i, t_i) un état d'un système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$, et $(t_i, y_i(q))$ le point du diagramme des vitesses correspondant à l'état courant du système pour une échéance $D(a_k)$, $k \geq i$. Il est possible de montrer que t_p^{mx} est une fonction décroissante en fonction de q en se basant sur le fait que C^{mx} est une fonction croissante.

Ce qui implique que :
- pour $q < q_{max}$, $\Gamma^{mx}(s_i, t_i) = (a_{i+1}, q)$ si et seulement si $t_p^{mx}(a_{i+1} \dots a_n, s_i, q) \geq t_i$ et $t_i > t_p^{mx}(a_{i+1} \dots a_n, s_i, q+1)$.

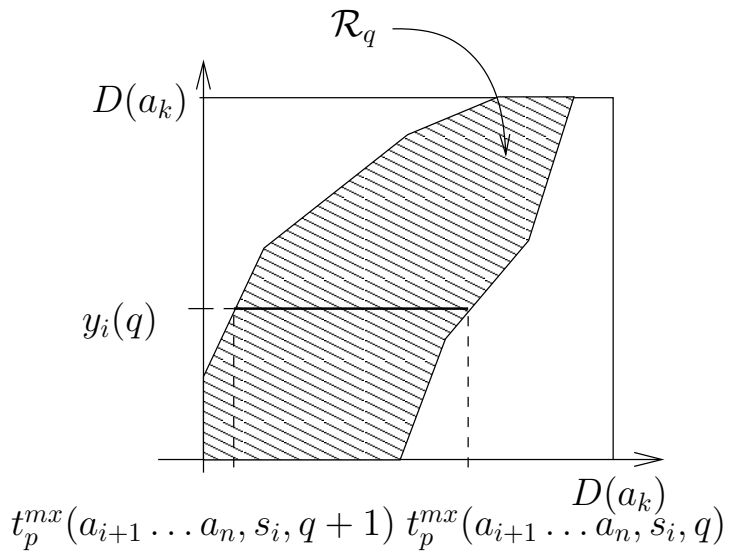


Figure 3.3 — Région de qualité pour un niveau de qualité q.

– pour $q = q_{max}$, $\Gamma^{mx}(s_i, t_i) = (a_{i+1}, q)$ si et seulement si $t_p^{mx}(a_{i+1}\ldots a_n, s_i, q) \geq t_i$ d'après la proposition 2.27.

Ce qui mène à la proposition suivante :

Proposition 3.7:
Pour un niveau de qualité donné et un état (s_i, q_i), $(s_i, t_i) \in \mathcal{R}_q$ si et seulement si :

$$\begin{array}{rcl} t_i & \in & \left]\ t_p^{mx}(a_{i+1}\ldots a_n, s_i, q+1)\ ,\ t_p^{mx}(a_{i+1}\ldots a_n, s_i, q)\ \right] \text{ pour } q < q_{max} \\ t_i & \in & \left]\ -\infty, t_p^{mx}(a_{i+1}\ldots a_n, s_i, q)\right] \text{ pour } q = q_{max}\ . \end{array}$$

Cette proposition permet de calculer des régions de qualité $\mathcal{R}_q$.

3.2.2 Vers une technique de contrôle symbolique

A partir des définitions des régions de qualité, les choix du contrôleur deviennent évident pour chaque état. Prenons un exemple montrant l'exécution entière d'une application contrôlée.

Exemple 3.2: *La figure 3.4 nous présente un diagramme des vitesses auquel a été rajouté les régions de qualité. Dans cette exemple nous considérons un système comportant quatre qualités, $q = 0, 1, 2, 3$. A partir de ce diagramme, nous pouvons directement comprendre la manière dont sera fait le choix du niveau de qualité. En effet prenons l'exécution de l'application jusqu'au point $(y_i(q), t_i)$. La région dans laquelle se situait la trace de l'exécution était celle de la qualité 2. Cependant en utilisant cette qualité l'application a atteint un point de la région de qualité inférieure. A ce moment là, le contrôleur va donc choisir la qualité 1, ce qui va lui permettre de satisfaire les propriétés de sûreté et d'optimalité.*

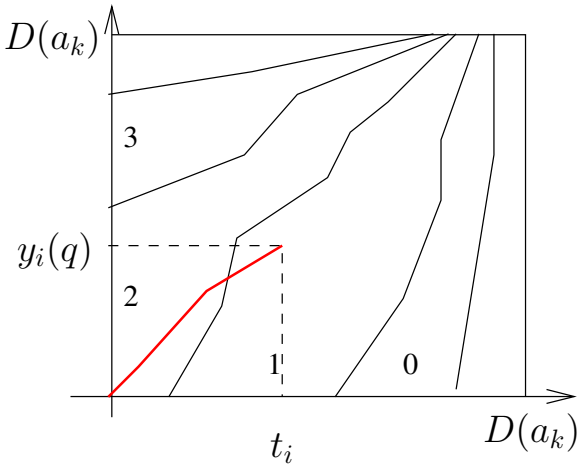


Figure 3.4 — Exemple de diagramme des vitesses muni de ses régions de qualité

L'utilisation du diagramme des vitesses découpé en régions simplifie la compréhension de la technique de contrôle. Elle permet de décrire tous les choix du contrôleur possibles. Les régions de qualité $\mathcal{R}_q$ nous permettent de comprendre exactement le fonctionnement du Gestionnaire de Qualité. En effet selon l'état où l'application se situe, il est assez simple par la construction du diagramme des vitesses de savoir quel niveau de qualité le contrôleur va choisir.

Cependant le diagramme des vitesses n'est pour l'instant qu'une représentation graphique du comportement de l'application contrôlée. A partir des régions que l'on vient de définir, les choix des niveaux de qualité deviennent évidents pour l'utilisateur. Ainsi si l'application se trouve dans la région de qualité $\mathcal{R}_q$, le contrôleur va automatiquement choisir la qualité q. Nous allons voir qu'il est possible d'utiliser cette observation pour modifier la méthode permettant de contrôler l'application.

La technique de contrôle utilisée est dite *énumérative*. Ce qualificatif est dû à son caractère calculatoire pour chaque pas de contrôle. Cette technique est de ce fait très coûteuse. En effet pour chaque état il est nécessaire de recalculer t_p^{mx}, ce qui implique de recalculer plusieurs valeurs notamment δ^{max}. Cela demande donc de parcourir plusieurs fois la séquence d'actions qu'il reste à exécuter. De plus la technique de gestion de qualité suppose que le Gestionnaire de Qualité est appelé avant d'exécuter chaque action de l'application. Puisque le Gestionnaire de Qualité et l'application sont composés ensembles, il y a un surcoût pour exécuter le Gestionnaire de Qualité. Un problème important est de réduire ce surcoût.

Pour réduire ce coût, il peut être intéressant d'utiliser une technique plus automatique mais respectant les mêmes propriétés que la méthode énumérative. En se basant sur les observations réalisées pour les diagrammes de vitesses, nous avons proposé une méthode dite *symbolique*. Cette méthode utilise directement le diagramme des vitesses en modélisant les régions de qualité. Il est possible pour le contrôleur de faire les choix en

observant simplement la région dans laquelle il se trouve.

L'algorithme 3.1 propose un calcul du contrôleur Γ^{mx} basé sur les régions de qualité. Cet algorithme prend en entrée les actions restant à exécuter $a_i \dots a_n$, le temps effectif t et la définition des régions de qualité $\mathcal{R}_\mathcal{Q}$. $\mathcal{R}_\mathcal{Q}$ représente l'ensemble des régions pour chaque qualité. Cet algorithme fonctionne comme celui présenté dans la section 2.1.2.2(algorithme 2.2). En effet il va chercher à quel niveau de qualité l'action a_i doit être exécutée pour que l'application puisse finir dans les temps en utilisant le maximum du budget de temps. La seule différence réside dans la manière de choisir cette qualité. En effet la technique compare le temps effectif à la fonction t_p^{mx} qu'il faut calculer pour chaque état. Alors que pour la méthode symbolique nous vérifions que le temps effectif t est dans la région de qualité $\mathcal{R}_q$.

```
1 Γ^mx(a_i ... a_n,R_Q,t_i)
2 q_m = q_min;
3 forall q ∈ Q do
4 |   if t_i ∈ R_q then
5 |   |   q_m = q ;
6 |   end
7 end
8 return (a, q) = (a_i, q_m)
```

Algorithme 3.1 : Algorithme de calcul du contrôleur Γ^{mx} basé sur les régions de qualité.

Dans le chapitre 5, nous proposerons des expérimentations permettant de comparer les deux techniques de contrôle, la technique énumérative et la technique symbolique. Nous montrerons l'efficacité de la méthode symbolique et l'amélioration au niveau du résultat de l'application contrôlée qu'il est possible d'obtenir.

3.3 Région de Relâchement du contrôleur

Nous avons vu dans la section précédente que l'appel au Gestionnaire de Qualité ajoute un surcoût à l'exécution de l'application. Nous avons donc proposé une méthode symbolique permettant de réduire la complexité des actions que réalise le contrôleur. Dans cette section, nous allons expliquer comment relaxer la granularité de contrôle, c'est-à-dire réduire le nombre d'appels au Gestionnaire de Qualité.

3.3.1 Vers un relâchement du contrôleur

La technique de contrôle symbolique permet d'ores et déjà de réduire le nombres d'opérations réalisées à chaque appel du contrôleur. Ceci est déjà une avancée dans la

technique de contrôle. En effet l'application va pouvoir améliorer la qualité qu'elle choisit, en ayant plus de temps pour s'exécuter. En effet si le temps imparti au contrôleur est réduit, cela augmente à l'inverse le temps que l'application possède pour réaliser ses algorithmes (ou calculs). Et de ce fait, cette technique permet d'augmenter la qualité du résultat de l'application contrôlée. Cependant nous allons voir que dans certaines situations, il serait possible de diminuer les appels au contrôleur sans violer les propriétés de sûreté et d'optimalité.

Exemple 3.3: *L'exemple de la figure 3.5 nous montre la trace d'exécution d'une application contrôlée. Si nous suivons l'exécution sur le diagramme des vitesses, nous pouvons observer des zones où le contrôleur fait un choix sur les niveaux de qualité identique pour des actions consécutives. Par exemple il choisit le niveau de qualité* 2 *pour la séquence d'actions* $a_0 \dots a_i$ *et il choisit le niveau de qualité 1 pour* $a_i \dots a_j$ *(cf. le comportement entouré). Dans cette situation l'appel au contrôleur devient donc redondant. Il suffirait d'affecter le niveau de qualité adéquat pour l'ensemble de cette séquence pour obtenir un résultat identique.*

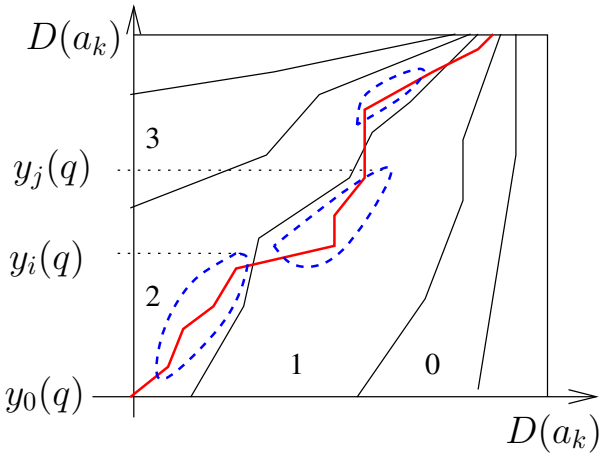


Figure 3.5 — Exemple de relâchement du contrôleur.

Cet exemple met en avant le fait qu'il est possible d'obtenir certaines situations où le contrôle à chaque action n'est pas justifié. En effet pour plusieurs actions consécutives, le contrôleur va choisir la même qualité. Ainsi on peut se demander si ce genre de comportement n'est pas prévisible. Si cela est le cas, alors il est possible de relâcher les appels au contrôleur.

Intuitivement l'on peut se dire qu'il est possible de créer des régions de relâchement à partir des régions de qualité. En effet reprenons l'exemple de la figure 3.5, pour la séquence $a_0 \dots a_i$, il serait possible de faire appel au contrôleur pour l'action a_0 puis de refaire appel au contrôleur seulement à partir de l'action a_i. En effet c'est au moment d'exécuter l'action a_i qu'il est possible de changer de niveau de qualité. Pour le restant de la séquence le contrôleur peut attribuer automatiquement, c'est-à-dire sans comparaison

ni calcul, le niveau de qualité choisi pour l'action a_0. Donc il est possible d'imaginer une zone de relâchement de i pas. C'est-à-dire que pour les i prochaines actions, on est sûr que le comportement de l'application ne peut pas sortir de la région de qualité $\mathcal{R}_q$ dans laquelle il se situe au moment de l'appel au contrôleur.

Nous avons décidé de créer des régions de relâchement du contrôleur à partir des régions de qualité. Ces régions devront nous assurer de répondre aux mêmes propriétés que la technique énumérative. C'est-à-dire qu'il faut qu'elles soient construites de telle sorte que les changements de niveau de qualité interviennent aux mêmes instants que si le contrôle était basé sur la technique énumérative. Pour cela il faut que la technique de relâchement prévoit les cas extrêmes, c'est-à-dire les situations où l'application a la possibilité de sortir de la région de qualité courante. On peut constater que pour sortir de cette région, il faut soit que l'application ait eu un comportement meilleur que celui estimé, soit moins bon. C'est-à-dire que l'on peut dégager deux cas(cf. 3.6) :

- pour l'action courante a_i, nous voulons connaître le nombre de pas avant lequel l'application pourra atteindre la région de qualité supérieure à celle de la qualité courante, c'est-à-dire la région de qualité $\mathcal{R}_3$ dans l'exemple. Pour cela il faut avoir une estimation du temps d'exécution meilleur-cas. Cela nous permettrait d'avoir une borne inférieure des durées d'exécution de l'action. Pour calculer le nombre de pas de relâchement on comptera le nombre d'actions que l'on peut exécuter à partir de l'action courante en prenant comme estimation des valeurs les durées d'exécution meilleur-cas. Cela correspond au tracé du comportement BC de la figure 3.6.
- pour l'action courante a_i, nous voulons connaître le nombre de pas avant lequel l'application pourra atteindre la région de qualité inférieure à celle de la qualité courante, c'est-à-dire la région de qualité $\mathcal{R}_1$ de l'exemple. Nous connaissons pour chaque action sa durée d'exécution pire-cas. Elle va nous permettre de calculer le nombre d'actions que l'on peut relâcher avant d'atteindre la région de qualité inférieure. Cela correspond au tracé du comportement WC de la figure 3.6.

A partir de ces deux valeurs, il est possible de calculer le nombre d'actions que nous pourrons relâcher en étant sûr de respecter le même comportement que le contrôleur énumératif. En effet les deux situations précédentes représentent les situations encadrant le comportement possible de l'application. Donc si on borne l'exécution en prenant en compte ces deux comportements nous sommes sûrs de respecter les mêmes propriétés que le contrôleur énumératif. Pour prendre en compte ces deux comportements, nous prendrons pour nombre de pas de relâchement le minimum de pas obtenus dans les deux cas.

3.3.2 Les régions de relâchement du contrôleur

Nous avons vu ci-dessus la méthode intuitive de relâchement du contrôleur. Nous allons proposer dans cette section, une formalisation des régions de relâchement. Puis nous présenterons un algorithme de relâchement du contrôleur basé sur ces régions.

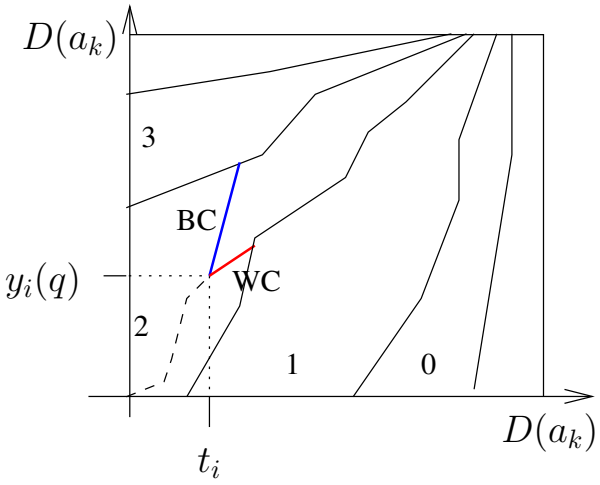


Figure 3.6 — Intuition sur le relâchement du contrôleur.

3.3.2.1 Définitions

Nous définissons les *régions de relâchement du contrôleur*, qui sont l'ensemble des états dans lesquels le Gestionnaire de Qualité peut être relâché sans dégrader la qualité du contrôle.

Soit (s_i, t_i) un état du système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$. Supposons que le Gestionnaire de Qualité Γ^{mx} choisisse le niveau de qualité q pour l'état (s_i, t_i), cela correspond à $(s_i, t_i) \in \mathcal{R}_q$. Nous considérons une technique conservative de relâchement du contrôleur : le Gestionnaire de Qualité peut être relâché pour $r \geq 1$ pas de calcul si nous assurons que le niveau de qualité que l'on choisit pour toutes les prochaines r actions $a_{i+1}, a_{i+2}, \ldots, a_{i+r}$ est q.

Définition 3.8:
Soient $SPI(C) = (G, Q, C^{wc}, D, C)$ un système paramétré incertain et Γ^{mx} un contrôleur utilisant la politique d'ordonnancement mixte. Une ***région de relâchement du contrôleur*** *$\mathcal{R}_q^r$ pour un niveau de qualité q et un entier $r \geq 1$ est définie par :*

$$(s_i, t_i) \in \mathcal{R}_q^r \Leftrightarrow (s_i, t_i) \in \mathcal{R}_q \wedge \forall (s_j, t_j) \in post_q^{r-1}(s_i, t_i),\ (s_j, t_j) \in \mathcal{R}_q$$

où $post_q^r(s_i, t_i) = \{\forall p \in [1, r](s_{i+p}, t_{i+p}) \mid t_{i+p} \in [t_i, t_i + C^{wc}(a_{i+1} \ldots a_{i+p}, q)]\}$ est l'ensemble des successeurs possibles de (s_i, t_i) après r actions.

Nous considérons les états (s_j, t_j), $j \in \{i, i+1, \ldots, i+r-1\}$ de $SPI(C) = (G, Q, C^{wc}, D, C)||\Gamma^{mx}$, et trouvons une condition pour ces états d'être dans $\mathcal{R}_q$ (voir Figure 3.7). Pour l'instant la figure 3.7 montre le cas où la propriété n'est pas satisfaite et le Gestionnaire de Qualité ne peut pas relâcher le contrôleur à partir de l'état (s_i, t_i) pour r pas de calcul.

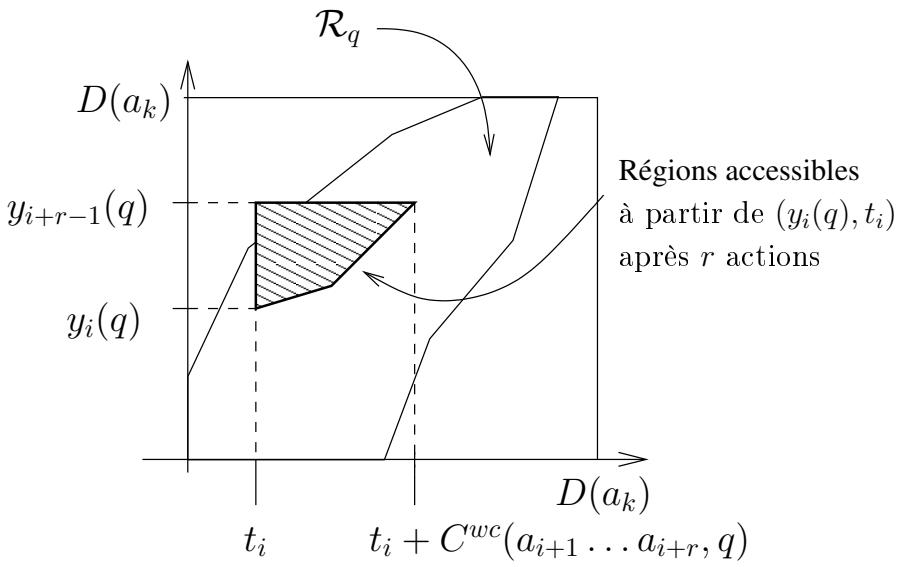


Figure 3.7 — Relâchement du contrôle : Le principe.

Proposition 3.9:
Pour un niveau de qualité q, un entier $r \geq 1$ et un état (s_i, q_i), $(s_i, t_i) \in \mathcal{R}_q^r$ si et seulement si

$$\begin{array}{lll} t_i & \in & \left] \; t_p^{mx}(a_{i+r} \ldots a_n, s_{i+r-1}, q+1) \; , \; t_p^{mx,r}(a_{i+1} \ldots a_n, s_i, q) \; \right] \quad \text{pour } q < q_{max} \\ t_i & \in & \left] \; -\infty \; , \; t_p^{mx,r}(a_{i+1} \ldots a_n, s_i, q) \; \right] \quad \text{pour } q = q_{max}, \end{array}$$

où $t_p^{mx,r}(a_{i+1} \ldots a_n, s_i, q) = \min\{t_p^{mx}(a_{i+1} \ldots a_n, s_i, q),\ \min_{i+1 \leq j \leq i+r-1} t_p^{mx}(a_{j+1} \ldots a_n, s_j, q) - C^{wc}(a_{i+1} \,..\, a_j, q)\}$

Preuve (de la proposition 3.9):
On a, par définition :

$$(s_i, t_i) \in \mathcal{R}_q^r \Leftrightarrow (s_i, t_i) \in \mathcal{R}_q \wedge \forall (s_j, t_j) \in post_q^{r-1}(s_i, t_i),\ (s_j, t_j) \in \mathcal{R}_q$$

Nous allons distinguer deux cas :

• $q = q_{max}$
D'après la proposition 3.7, nous avons :

$$\begin{array}{rcl} (s_i, t_i) \in \mathcal{R}_q^r & \Leftrightarrow & t_i \in \left] -\infty, t_p^{mx}(a_{i+1} \ldots a_n, s_i, q) \right] \wedge \\ & & \forall (s_j, t_j) \in post_q^{r-1}(s_i, t_i), \quad t_j \in \left] -\infty, t_p^{mx}(a_{j+1} \ldots a_n, s_j, q) \right] \\ & \Leftrightarrow & t_i \in \left] -\infty, t_p^{mx}(a_{i+1} \ldots a_n, s_i, q) \right] \wedge \\ & & \forall j \in \left[i+1, i+r-1 \right] \; \forall t_j \in \left[t_i, t_i + C^{wc}(a_{i+1} \ldots a_j, q) \right], \\ & & \quad t_j \in \left] -\infty, t_p^{mx}(a_{j+1} \ldots a_n, s_j, q) \right] \end{array}$$

Etant donné que $t_j \in \big[t_i, t_i + C^{wc}(a_{i+1} \dots a_j, q)\big]$, *on peut écrire :*

$$\begin{aligned}
(s_i, t_i) \in \mathcal{R}_q^r \quad \Leftrightarrow \quad & t_i \in \big]-\infty, t_p^{mx}(a_{i+1} \dots a_n, s_i, q)\big] \wedge \\
& \forall j \in \big[i+1, i+r-1\big], \\
& \quad \big[t_i, t_i + C^{wc}(a_{i+1} \dots a_j, q)\big] \subseteq \ \big]-\infty, t_p^{mx}(a_{j+1} \dots a_n, s_j, q)\big] \\
\Leftrightarrow \quad & t_i \in \big]-\infty, t_p^{mx}(a_{i+1} \dots a_n, s_i, q)\big] \wedge \\
& \forall j \in \big[i+1, i+r-1\big], \\
& \quad t_i + C^{wc}(a_{i+1} \dots a_j, q) \leq t_p^{mx}(a_{j+1} \dots a_n, s_j, q) \\
\Leftrightarrow \quad & t_i \leq t_p^{mx}(a_{i+1} \dots a_n, s_i, q) \wedge \\
& \forall j \in \big[i+1, i+r-1\big], \\
& \quad t_i \leq t_p^{mx}(a_{j+1} \dots a_n, s_j, q) - C^{wc}(a_{i+1} \dots a_j, q) \\
\Leftrightarrow \quad & t_i \leq \min\{t_p^{mx}(a_{i+1} \dots a_n, s_i, q), \\
& \qquad \min_{\mathbf{i+1 \leq j \leq i+r-1}} t_p^{mx}(a_{j+1} \dots a_n, s_j, q) - C^{wc}(a_{i+1} \,..\, a_j, q)\}
\end{aligned}$$

Donc par définiton de $t_p^{mx,r}$, *nous avons pour* $q = q_{max}$ *:*

$$(s_i, t_i) \in \mathcal{R}_q^r \Leftrightarrow t_i \in \big]-\infty, t_p^{mx,r}(a_{i+1} \dots a_n, s_i, q)\big]$$

• $q < q_{max}$
D'après la proposition 3.7, nous avons :

$$\begin{aligned}
(s_i, t_i) \in \mathcal{R}_q^r \quad \Leftrightarrow \quad & t_i \in \big]t_p^{mx}(a_{i+1} \dots a_n, s_i, q+1), t_p^{mx}(a_{i+1} \dots a_n, s_i, q)\big] \wedge \\
& \forall (s_j, t_j) \in post_q^{r-1}(s_i, t_i), \\
& \quad t_j \in \big]t_p^{mx}(a_{j+1} \dots a_n, s_j, q+1), t_p^{mx}(a_{j+1} \dots a_n, s_j, q)\big] \\
\Leftrightarrow \quad & t_i \in \big]t_p^{mx}(a_{i+1} \dots a_n, s_i, q+1), t_p^{mx}(a_{i+1} \dots a_n, s_i, q)\big] \wedge \\
& \forall j \in \big[i+1, i+r-1\big] \ \forall t_j \in \big[t_i, t_i + C^{wc}(a_{i+1} \dots a_j, q)\big], \\
& \quad t_j \in \big]t_p^{mx}(a_{j+1} \dots a_n, s_j, q+1), t_p^{mx}(a_{j+1} \dots a_n, s_j, q)\big]
\end{aligned}$$

Etant donné que $t_j \in \big[t_i, t_i + C^{wc}(a_{i+1} \dots a_j, q)\big]$, *on peut écrire :*

$$\begin{aligned}
(s_i, t_i) \in \mathcal{R}_q^r \quad \Leftrightarrow \quad & t_i \in \big]t_p^{mx}(a_{i+1} \dots a_n, s_i, q+1), t_p^{mx}(a_{i+1} \dots a_n, s_i, q)\big] \wedge \\
& \forall j \in \big[i+1, i+r-1\big], \\
& \quad \big[t_i, t_i + C^{wc}(a_{i+1} \dots a_j, q)\big] \subseteq \\
& \qquad \big]t_p^{mx}(a_{j+1} \dots a_n, s_j, q+1), t_p^{mx}(a_{j+1} \dots a_n, s_j, q)\big] \\
\Leftrightarrow \quad & t_p^{mx}(a_{i+1} \dots a_n, s_i, q+1) < t_i \leq t_p^{mx}(a_{i+1} \dots a_n, s_i, q) \wedge \\
& \forall j \in \big[i+1, i+r-1\big], \\
& \quad t_p^{mx}(a_{j+1} \dots a_n, s_j, q+1) < t_i \leq t_p^{mx}(a_{j+1} \dots a_n, s_j, q) - C^{wc}(a_{i+1} \dots a_j, q)
\end{aligned}$$

En particulier, nous avons :

$$t_p^{mx}(a_{i+r} \dots a_n, s_{i+r-1}, q+1) < t_i$$

Ainsi par définiton de $t_p^{mx,r}$*, nous avons pour* $q < q_{max}$ *:*

$$(s_i, t_i) \in \mathcal{R}_q^r \Leftrightarrow t_i \in \left]t_p^{mx}(a_{i+r} \dots a_n, s_{i+r-1}, q+1), t_p^{mx,r}(a_{i+1} \dots a_n, s_i, q)\right]$$

Donc on peut conclure que :

$$\begin{array}{rcl} (s_i, t_i) \in \mathcal{R}_q^r & \Leftrightarrow & \\ t_i & \in & \left] \ t_p^{mx}(a_{i+r} \dots a_n, s_{i+r-1}, q+1) \ , \ t_p^{mx,r}(a_{i+1} \dots a_n, s_i, q) \ \right] \ \text{pour } q < q_{max} \\ t_i & \in & \left] \ -\infty \ , \ t_p^{mx,r}(a_{i+1} \dots a_n, s_i, q) \ \right] \ \text{pour } q = q_{max}, \end{array}$$

□

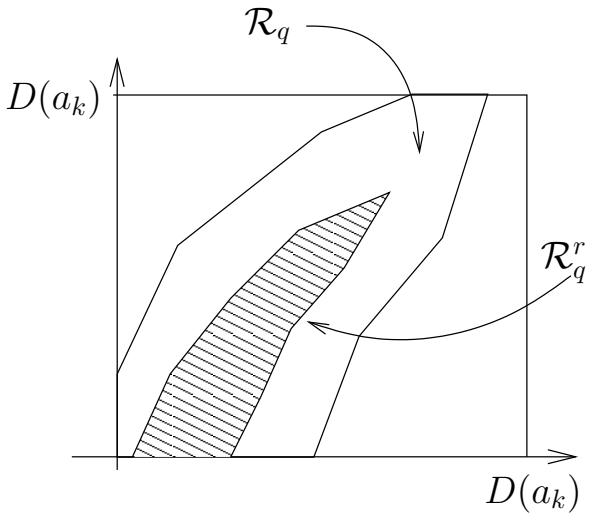


Figure 3.8 — Région de relâchement du contrôleur.

3.3.2.2 Algorithme du contrôleur relâché

La méthode de calcul des régions de relâchement utilise les opérations de la technique de contrôle énumérative. Pour cette raison il est donc nécessaire de précalculer un échantillon des régions $\mathcal{R}_q^r$ en prenant en compte des valeurs de relâchement pertinentes. Les régions de relâchement vont permettre la construction d'un contrôleur Γ^{mx} relâché. Pour bien comprendre la méthode que l'on va utiliser, nous allons prendre un exemple permettant d'observer intuitivement le comportement du contrôleur relâché.

Exemple 3.4: *Pour cet exemple, nous allons prendre comme représentation la figure 3.9. Celle-ci nous présente un diagramme des vitesses sur lequel a été représenté une région*

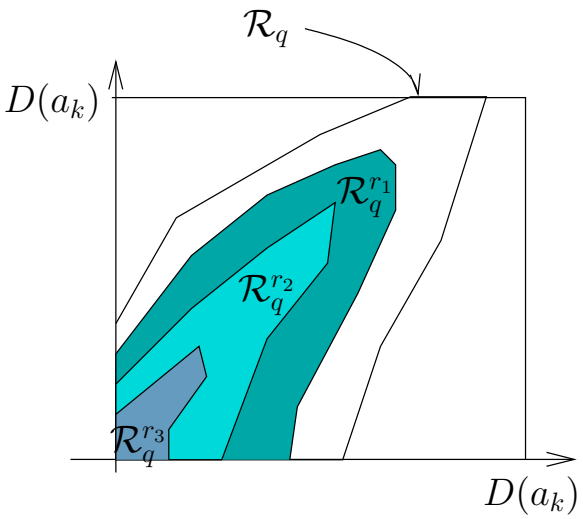


Figure 3.9 — Construction des régions de relâchement pour une région de qualité.

de qualité $\mathcal{R}_q$. Nous avons construit trois zones de relâchement $\mathcal{R}_q^{r_1}$, $\mathcal{R}_q^{r_2}$, $\mathcal{R}_q^{r_3}$. Pour être plus précis, si on applique le contrôleur sur un état du système étant dans la région de qualité $\mathcal{R}_q^{r_1}$, on pourra relâcher les appels au contrôleur pendant r_1 pas, c'est-à-dire pour les r_1 prochaines actions.

Prenons un exemple d'exécution, l'application a atteint l'état $(t_i, y_i(q))$. A cet instant le contrôleur connaît la région de qualité où il se situe, $\mathcal{R}_q$. Intuitivement, il est possible d'observer deux situations pour le contrôleur relâché :

- *dans la première, le nombre d'actions que l'on peut relâcher est égal à zéro. A ce moment là, le contrôleur doit donc recalculer la qualité en observant la région de qualité où il se trouve. Mais aussi savoir combien d'actions à partir de cet état, il peut relâcher. Cette opération est faite en observant la région de relâchement où l'état courant $(t_i, y_i(q))$ se situe.*
- *le deuxième cas est celui où il reste des actions à relâcher. Dans ce cas précis, le contrôleur, sans vérification, va affecter à l'action suivante la qualité de l'action courante q ainsi qu'un nombre de pas de relâchement $r - 1$.*

A partir de cet exemple, il est assez simple de déduire la manière dont le contrôleur fonctionnera. Nous proposons donc un algorithme de calcul du contrôleur Γ^{mx} basé sur les régions de relaxation.

L'algorithme 3.2 nous montre la méthode de relâchement du contrôleur. Il prend en entrée la séquence d'action restant à exécuter, ainsi que le temps t_i que l'application a atteint. De plus on lui fournit un ensemble de régions de relâchement $\mathcal{R}_{\mathcal{Q}}^R$. Celui-ci correspond au régions de relâchement pour toutes les qualités ainsi que pour tous les pas de relâchement r que l'on a choisis. On peut le définir par :

$$\mathcal{R}_{\mathcal{Q}}^R = \{\mathcal{R}_q^r | r \in R, q \in \mathcal{Q}\}$$

```
1  Γ^mx(a_i ... a_n, R^R_Q, t, relax, quality)
2  q_m = quality;
3  r_m = relax − 1;
4  if relax = 0 then
5  |  forall q ∈ Q do
6  |  |  forall r ∈ R do
7  |  |  |  if t_i ∈ R^r_q then
8  |  |  |  |  q_m = q_i ;
9  |  |  |  |  r_m = r;
10 |  |  |  end
11 |  |  end
12 |  end
13 end
14 return (a, q, r) = (a_i, q_m, r_m)
```

Algorithme 3.2 : Algorithme de calcul du contrôleur Γ^{mx} basé sur les régions de relaxation.

Il est aussi nécessaire de lui fournir en entrée deux autres valeurs représentant le nombre de pas de relâchement, *relax*, qu'il lui reste à effectuer, ainsi que la *qualité* à laquelle l'action précédente a été exécutée. L'algorithme retourne la qualité de l'action précédente et le nombre de pas de relâchement diminué d'un pas dans le cas où le paramètre *relax* est différent de 0. Sinon il récupère les informations nécessaires, à savoir la qualité à choisir et le nombre de pas de relâchement en comparant l'état courant avec les régions de relâchement $\mathcal{R}^R_{\mathcal{Q}}$.

3.4 Conclusion

Le diagramme des vitesses est une représentation graphique permettant de modéliser le comportement de l'application contrôlée. Ce diagramme permet notamment de suivre l'exécution de l'application, mais il donne aussi la possibilité de comprendre les politiques de gestion de qualité. Dans cette section, nous avons expliqué plus spécifiquement la politique de gestion de qualité *mixte*. Nous avons pu expliquer son fonctionnement au niveau des choix de qualité et montrer sur un diagramme ce que représentait la marge de sécurité δ^{max}.

De plus cette représentation graphique a servi de point de départ à l'élaboration d'une technique de contrôle réduisant le surcoût lié aux appels du contrôleur. Cette technique est basée sur la construction de régions de qualité. C'est-à-dire des régions où le contrôleur choisira une qualité constante. L'algorithme issue de cette technique de contrôle, du fait de l'utilisation de ses régions, correspond à une étude symbolique de l'état du système. Cela revient à observer dans quelle région de qualité se situe l'application contrôlée. Il

n'y a donc plus aucun calcul d'effectué ce qui réduit le temps de contrôle.

À partir de cette technique de contrôle symbolique nous avons pu observer que le contrôleur avait dans certaines situations la possibilité de réutiliser le niveau de qualité qu'il avait déjà choisi auparavant. C'est la raison pour laquelle nous avons proposé une technique de contrôle symbolique avec relâchement. Ceci dans l'optique d'optimiser la politique de gestion de qualité *mixte*, en réduisant son surcoût. Nous avons proposé une représentation de régions de relâchement toujours en basant notre approche sur le diagramme des vitesses. Cette technique permet d'accroître le niveau de qualité que choisit le contrôleur, grâce à un surcoût fortement diminué tout en respectant les mêmes propriétés que la technique de contrôle .

Dans le chapitre 5, nous allons donner une comparaison des trois techniques de contrôle. Ainsi nous pourrons mettre en évidence sur les améliorations que propose la technique de contrôle symbolique.

CHAPITRE 4

Évolution vers une approche stochastique

La politique de contrôle mixte présentée dans le chapitre 2 est basée sur un mélange entre les durées d'exécution pire-cas et moyennes des actions de l'application cible. Ces deux types de comportements sont nécessaires pour garantir le respect des contraintes de sûreté et d'optimalité. D'une manière générale, les durées d'exécution pire-cas permettent de restreindre l'éventail des comportements possibles, mais elles ne permettent pas de connaître le comportement le plus probable. Les durées d'exécution moyennes, quant à elles, permettent de rapprocher les estimations du contrôleur du comportement réel de l'application. Cependant les durées d'exécution pire-cas sont généralement surévaluées (cf. section 1.1.1.2), et leurs utilisations conduisent à une double approximation :

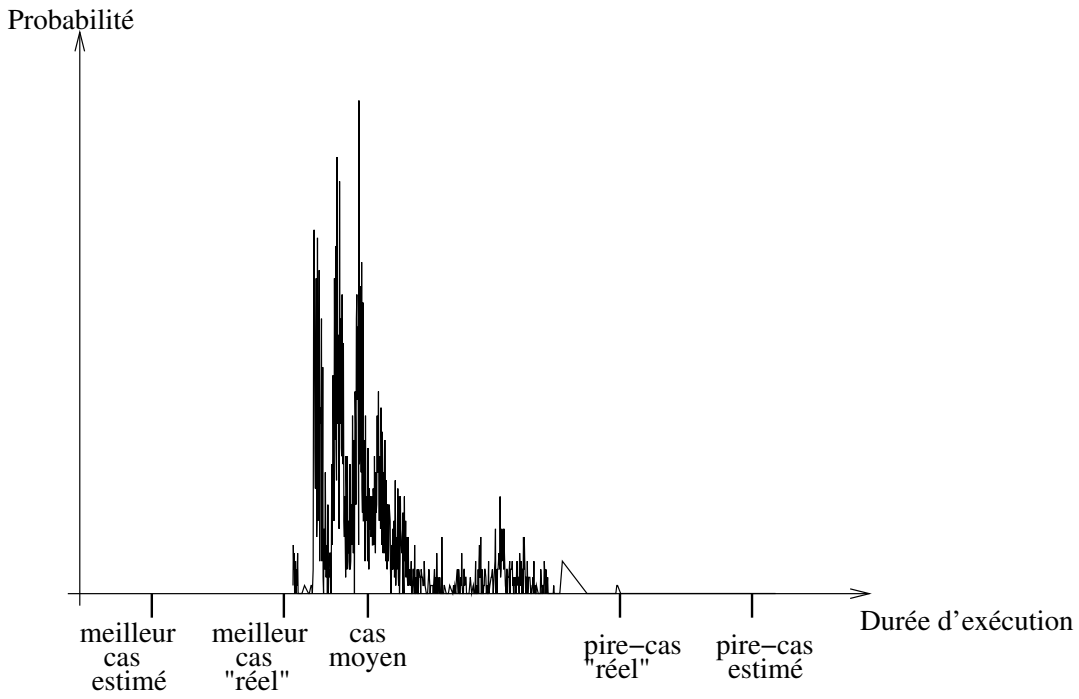


Figure 4.1 — Distribution typique de la durée d'exécution d'une action.

- D'une part, les durées d'exécution réelles sont souvent très inférieures aux durées d'exécution pire-cas ;
- D'autre part, il est très difficile de calculer statiquement la vraie durée d'exécution pire-cas. Ainsi les durées fournies ne sont que des bornes supérieures des durées d'exécution pire-cas "réelles". La figure 4.1 illustre ce phénomène en représentant

une distribution typique des durées d'exécution d'une action pour un niveau de qualité donné.

Cette double approximation peut se révéler très pénalisante du point de vue des performances du contrôleur. Nous ne pouvons pas nous passer des durées d'exécution pire-cas lorsqu'il s'agit de garantir le respect des échéances. Cependant pour les applications multimédia embarquées ne possédant pas de caractère critique, si les échéances sont violées, il y aura uniquement une dégradation de la qualité de service. On peut donc se demander si garantir toutes les propriétés de sûreté est pertinent. De ce fait l'utilisation des durées d'exécution pire-cas autant surévaluées n'est pas forcément adéquate. Néanmoins il est tout de même possible d'utiliser des modèles de temps plus proches du comportement réel de l'application afin de mieux renseigner le contrôleur.

Dans ce chapitre nous allons proposer une technique de contrôle de qualité de service qui sera basée sur un modèle de temps représenté par des distributions des durées d'exécution des actions. A partir de ces distributions, il est possible de définir un problème de qualité de service légèrement différent de celui qui a été présenté dans la section 2.1.2. En effet il est nécessaire d'apporter des modifications car nous n'aurons plus de durées d'exécution pire-cas surévaluées et donc plus la possibilité de faire des estimations sur le comportement de l'application en garantissant la sûreté. C'est pourquoi nous allons proposer à l'utilisateur de fixer à partir des distributions des *pire-cas stochastiques*. Ceci sera réalisé à partir d'un *seuil de tolérance* qui représente le pourcentage de valeurs sur la distribution comprise après la durée d'exécution pire-cas. A partir de cette dernière il est possible d'utiliser la politique mixte mais en considérant que certaines échéances peuvent être manquées.

Le seuil de tolérance va permettre de générer des durées d'exécution pire-cas stochastiques pour chaque action. Cependant nous verrons qu'il n'est pas évident de connaître à partir de cette valeur une probabilité de dépasser les échéances. C'est la raison pour laquelle nous allons construire la distribution de l'application pour sa date de terminaison, en composant les distributions de chaque action. Cette construction va nous permettre de faire une étude de performances en fonction de la valeur qui aura été donnée pour le seuil de tolérance. En effet nous pourrons fournir à la fois une estimation de la consommation du budget de temps mais aussi une probabilité en ce qui concerne le nombre d'échéances qui pourront être violées. Ces résultats vont permettre à l'utilisateur de maîtriser totalement le fonctionnement de l'application et pourra en fonction de ceux-ci modifier le seuil de tolérance. On pourra alors parler de *temps-réel "souple" mais maîtrisé*.

4.1 Motivations

Cette section présente les motivations de la démarche. Nous donnerons en premier lieu les limitations de la méthode présentée dans le chapitre 2. En particulier nous expliquerons pourquoi les durées d'exécution pire-cas ne répondent pas toujours au domaine des

applications multimédia temps-réel. Nous expliquerons ensuite l'intérêt d'une nouvelle approche basée sur un modèle de temps stochastique.

4.1.1 Calcul du temps d'exécution pire-cas

La connaissance des durées d'exécution pire-cas est, comme nous l'avons vu précédemment, nécessaire pour garantir les contraintes de temps. Il est cependant très difficile d'obtenir ces pire-cas pour des applications dont le logiciel et le matériel sont suffisamment complexes (cf. section 1.1.1.2). Et nous utilisons donc des approximations de ces pire-cas, qui sont souvent surévaluées. Une question se pose donc par rapport à l'influence que peut avoir un modèle de temps construit à partir de ces surévaluations sur les choix que fait le contrôleur au niveau de la qualité. En d'autres termes, comment le degré de précision de l'estimation du pire-cas influence la technique de contrôle. L'exemple suivant montre les niveaux de qualité obtenus pour différentes durées d'exécution pire-cas.

Exemple 4.1: *Nous reprenons ici les notations vues dans le chapitre 2 et nous considérons un système paramétré SPI(C) avec trois actions {QUANT, IQUANT, IDCT}, un ensemble Q de quatre qualités {1,2,3,4} et une échéance unique* $D = 9$. *Nous supposons que SPI(C) possède un ordonnancement possible "QUANT • IQUANT • IDCT" tel que le temps-réel et le temps pire-cas sont donnés dans le tableau de la figure 4.2. Nous considérons aussi que le temps moyen est égal au temps-réel. Nous introduisons pour cet exemple un temps pire-cas que nous appellerons "temps pire-cas observé", il correspond au temps maximal que nous avons pu récupérer lors des exécutions de l'application.* α *représente la séquence QUANT • IQUANT • IDCT.*

q	$C^{wc}(a,q)$ surestimées	$C^{wc}(a,q)$ observées	$C(a,q) = C^{av}(a,q)$
1	1	1	1
2	5	3	2
3	6	3	3
4	7	6	5

Figure 4.2 — Durées d'exécution pire-cas surestimées, pire-cas observées et moyennes pour les actions QUANT, IQUANT et DCT

Si nous appliquons la politique de gestion de qualité mixte *décrite à la section 2.2.3,*

avec le pire-cas surestimé, le contrôleur fera les calculs suivants :

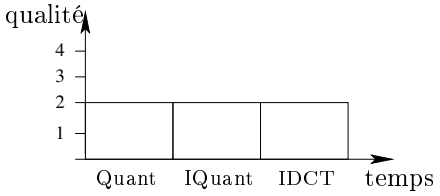
$$
\begin{aligned}
\textit{pour } q=4 \ : t_p^{mx}(\alpha, s_0, 4) &= D - (C^{av}(\alpha,4) + \delta^{max}(\alpha,4)) \\
&= 9 - (15+3) = -9 \\
t_p^{mx}(\alpha, s_0, 4) &< 0 \\
\textit{pour } q=3 \ : t_p^{mx}(\alpha, s_0, 3) &= D - (C^{av}(\alpha,3) + \delta^{max}(\alpha,3)) \\
&= 9 - (9+3) = -3 \\
t_p^{mx}(\alpha, s_0, 3) &< 0 \\
\textit{pour } q=2 \ : t_p^{mx}(\alpha, s_0, 2) &= D - (C^{av}(\alpha,2) + \delta^{max}(\alpha,2)) \\
&= 9 - (6+3) = 0 \\
t_p^{mx}(\alpha, s_0, 2) &\geq 0
\end{aligned}
$$

Le respect de la date d'échéance n'est obtenu que pour le choix d'un niveau de qualité égal ou inférieur à 2. Ainsi il choisira le niveau de qualité 2 pour l'action QUANT, et en déroulant le calcul pour chaque action, il assigne cette même qualité pour chaque action.

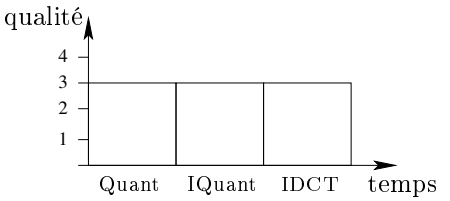
Par contre en appliquant le pire-cas réel, le choix serait autre :

$$
\begin{aligned}
\textit{pour } q=4 \ : t_p^{mx}(\alpha, s_0, 4) &= D - (C^{av}(\alpha,4) + \delta^{max}(\alpha,4)) \\
&= 9 - (15+1) = -7 \\
t_p^{mx}(\alpha, s_0, 4) &< 0 \\
\textit{pour } q=3 \ : t_p^{mx}(\alpha, s_0, 3) &= D - (C^{av}(\alpha,3) + \delta^{max}(\alpha,3)) \\
&= 9 - (9+0) = 0 \\
t_p^{mx}(\alpha, s_0, 3) &\geq 0
\end{aligned}
$$

Il aurait donc choisi une meilleure qualité(cf. figure 4.3).



Choix de qualité pour le pire-cas surestimé



Choix de qualité pour le pire-cas réel

Figure 4.3 — Comparaison entre un pire-cas surestimé et un pire-cas réel.

Cette exemple met en évidence que pour des applications multimédia, il serait nécessaire de se rapprocher des durées d'exécution réelles et non de baser les calculs sur des pire-cas trop élevés.

4.1.2 Entre temps-réel dur et temps-réel mou

Le contrôleur d'application tel qu'il a été réalisé permet de respecter les propriétés des deux mondes du temps-réel, le *temps-réel mou* (ou souple) et le *temps-réel dur* (voir section 1.1.1.2). Premièrement le fait d'avoir des durées d'exécution pire-cas et une échéance à ne pas dépasser va répondre aux attentes de la propriété de sûreté qui est de ne pas violer les échéances, propriété du temps-réel dur. Dans un deuxième temps, le contrôleur a pour objectif d'atteindre une utilisation maximum de ce budget de temps, ce qui correspond aux attentes du critère d'optimalité qui fait parti des propriétés du temps-réel mou.

Une application multimédia est plus souvent considérée comme relevant du temps-réel mou. En effet il est rare de voir des aspects critiques sur des visioconférences ou des jeux en réseau. Ce type d'application est doté de mécanismes permettant de rattraper les dépassements d'échéances. De plus le non respect des contraintes de temps influe uniquement sur la qualité de service, et n'a aucune conséquence "dramatique". Ces applications s'accommodent du non-respect des contraintes temporelles dans certaines limites au-delà desquelles le système devient inutilisable. Prenons l'exemple d'une application de visioconférence : l'encodage d'une image dure plus longtemps que prévu lors d'une discussion. Pour rattraper l'erreur, le système a la possibilité de ne pas encoder l'image suivante, ce qui implique une diminution du taux d'images par seconde et donc une séquence vidéo saccadée. Ce phénomène est acceptable s'il est peu fréquent. Laisser la possibilité au système de violer les contraintes temps-réel nécessite des mécanismes additionnels, soit pour réduire le taux de contraintes violées, soit pour éviter que le système ne plante lorsque ces contraintes ne sont pas respectées. Un système temps-réel mou doit respecter les contraintes pour une moyenne de ses exécutions. Un dépassement exceptionnel est toléré. En effet il sera peut-être récupéré à l'exécution suivante. Pour des applications multimédia, l'intérêt de garantir les propriétés du temps-réel mou est d'améliorer la qualité de service. Mais en contrepartie, il n'y a aucun moyen pour l'utilisateur de prévoir leurs comportements, c'est-à-dire de connaître à priori le nombre d'échéances violées par exemple. En général le taux de contraintes temps-réel violées n'est pas prévisible. Pour cette raison, nous avons dans l'approche initiale choisi de ne pas dépasser les échéances, et ainsi de contrôler exactement ce taux. Cela correspond à traiter le problème avec des approches temps-réel dur.

Les approches temps-réel dur ne tolèrent aucun dépassement des contraintes. Elles sont souvent utilisées lorsque de tels dépassements peuvent conduire à des situations critiques. Elles cherchent à s'assurer que le système fonctionnera correctement dans tous les cas de figure. Elles sont le plus souvent utilisées dans des systèmes dits critiques, par exemple pour les applications embarquées dans les avions, les trains (cf. section 1.2). Pour des applications multimédia, le fait de considérer ces approches va permettre de ne pas avoir le phénomène décrit précédemment pour les visioconférences, c'est-à-dire de respecter les contraintes de temps dans toutes les situations. Certes une telle approche

est intéressante étant donné qu'il n'est pas nécessaire de faire appel à des mécanismes de récupération du système comme nous pourrions le faire avec des approches du temps-réel mou. Cependant cela réduit considérablement la qualité de service que le système aurait eu avec ces dernières.

Les applications multimédia embarquées nécessitent donc de répondre aux deux mondes du temps-réel. Nous avons donc présenté dans la section 2.2.3 une politique de contrôle permettant de garantir à la fois les propriétés de sûreté, mais aussi d'optimalité. Ainsi la technique a permis de rapprocher les deux mondes. Cependant elle reste toutefois assez contraignante pour des applications non critiques. Dans ces dernières, on ne veut pas garantir le respect de toutes les contraintes de temps mais seulement qu'elles en respectent suffisamment pour bien fonctionner. Il pourrait donc être intéressant pour l'utilisateur de pouvoir contrôler le taux de violation des échéances. On pourrait ainsi lui permettre de trouver un compromis entre le taux de contraintes temps-réel manquées et le taux d'utilisation de la plateforme.

4.1.3 Vers une maîtrise des temps d'exécution pire-cas

L'aspect temps-réel dur des applications multimédia embarquées peut sembler nécessaire comme nous venons de le voir. Cependant il peut être intéressant pour ces applications de violer certaines contraintes à condition d'en violer le moins possible. En effet cela peut améliorer d'une certaine manière la qualité de service. Les contraintes temporelles sont respectées grâce aux évaluations pire-cas des durées d'exécution des actions. Nous avons vu que celles-ci sont surévaluées et que cela occasionne une dégradation du niveau de qualité de l'application. On peut donc se demander comment améliorer la génération de ces valeurs.

Un deuxième point que nous pouvons mettre en évidence est la difficulté de maîtriser le comportement de l'application, c'est-à-dire de connaître à priori le nombre d'échéances qui seront violées. Ainsi on peut donner à l'utilisateur un moyen de fixer les durées d'exécution pire-cas et de connaître par rapport à ceux-ci le comportement qu'aura l'application contrôlée. Cela lui permettra d'adapter les pire-cas aux performances attendues de son application.

L'idée est ainsi de construire un modèle de temps basé sur des distributions de probabilités des durées d'exécution de chaque action. A partir de celles-ci, l'utilisateur peut fixer lui même les pire-cas qu'il ne veut pas dépasser, c'est à dire les durées d'exécution pire-cas avec lesquelles seront réalisées les calculs de la politique de contrôle. Pour les fixer, on lui propose d'utiliser un seuil de tolérance, ce qui équivaut à une valeur représentant la probabilité que la durée réelle d'une action dépasse le pire-cas. Ensuite la technique de contrôle pourra être la même que précédemment, mais nous pourrons fournir un moyen de maîtriser le comportement de l'application contrôlée, en fournissant des estimations de performances basées sur le calcul de la distribution de la date de terminaison t_n de l'action a_n en fonction du seuil de tolérance.

Dans la suite du chapitre nous allons proposer une méthode permettant de fixer les temps d'exécution pire-cas selon les choix de l'utilisateur. Ceux-ci consisteront soit à respecter des propriétés de sûreté soit à garantir des propriétés d'optimalité. Nous pouvons qualifier cette approche de "temps-réel mou mais maîtrisé". En effet nous allons permettre au système de dépasser les contraintes de temps mais étant donné que nous fixons le pourcentage d'erreur, nous maîtrisons le nombre de ces dépassements.

4.2 Rappel sur la théorie des probabilités

Dans cette section nous allons énoncer quelques rappels sur la théorie des probabilités. Ceux-ci vont nous permettre de fixer les définitions utiles pour la construction de la technique de contrôle stochastique.

4.2.1 Notions de base des probabilités

La théorie des probabilités s'intéresse à la modélisation des phénomènes aléatoires. Une modélisation implique certainement une simplification des phénomènes, mais cette simplification conduit à une quantification, donc à la possibilité de faire des calculs et à prédire. La modélisation et le calcul des probabilités sont faits à partir de 3 objets mathématiques $(\Omega, \mathcal{A}, P)$ que nous allons décrire dans ce qui suit.

Définition 4.1:
L'ensemble Ω contenant tous les résultats possibles des observations d'un phénomène est appelé ensemble des observables. *Ses éléments $\omega \in \Omega$ sont appelés* observables.

Nous conviendrons qu'effectuer une expérience correspond à sélectionner par un procédé quelconque un élément ω dans l'ensemble Ω.

Exemple 4.2: *Jeter un dé revient à sélectionner un élément de $\Omega = \{1, 2, 3, 4, 5, 6\}$. Observer la durée d'exécution d'une action pour une qualité revient à sélectionner un élément de $\Omega = [0, +\infty[$.*

Notons $\mathcal{P}(\Omega) = \{A \subseteq \Omega\}$ l'ensemble des parties de Ω.

Définition 4.2:
Soient Ω un espace d'observables et $\mathcal{A}$ un sous-ensemble de $\mathcal{P}(\Omega)$. $\mathcal{A}$ est une tribu sur Ω si

1. *$\Omega \in \mathcal{A}$*
2. *$\mathcal{A}$ est stable par passage au complémentaire (c'est-à-dire $A \in \mathcal{A} \Rightarrow \Omega \setminus A \in \mathcal{A}$),*
3. *$\mathcal{A}$ est stable par réunion dénombrable (c'est-à-dire $\forall i \in \mathbb{N}, A_i \in \mathcal{A} \Rightarrow \bigcup_{i \geq 1} A_i \in \mathcal{A}$),*

Le couple $(\Omega, \mathcal{A})$ formé d'un ensemble Ω et d'une tribu $\mathcal{A}$ sur Ω sera appelé un espace mesurable. Les éléments de $\mathcal{A}$ sont appelés ensembles mesurables.

Définition 4.3:
Tout élément de $\mathcal{P}(\Omega)$ peut être considéré comme la représentation d'un évènement.

Exemple 4.3: *Pour l'exemple du lancer de dé, "Faire un nombre pair" est un évènement. De même, $A = \{1, 3, 5\}$ est l'évènement correspondant à un résultat impair.*

La calcul des probabilités consiste à associer aux évènements leur probabilité. Dans le cas général, on n'assigne pas une probabilité à tous les évènements possibles mais à une collection d'évènements, notée $\mathcal{A}$.

Définition 4.4:
Soient $(\Omega, \mathcal{A})$ et $(E, \mathcal{B})$, deux espaces mesurables. Soit f une fonction définie par $f : \Omega \to E$. On dit que f est une fonction mesurable (pour $\mathcal{A}$ et $\mathcal{B}$) si $f^{-1}(B) \in \mathcal{A}$ $\forall B \in \mathcal{B}$.

Définition 4.5:
Etant donné un espace mesurable $(\Omega, \mathcal{A})$, une mesure de probabilité P est une application de $\mathcal{A}$ dans $[0, 1]$, donc un procédé qui associe à tout évènement mesurable A un nombre $P(A)$ compris entre 0 et 1 appelé probabilité de A, et qui satisfait aux axiomes suivants :

- *L' évènement certain est de probabilité 1 : $P(\Omega) = 1$.*
- *Axiome d'additivité dénombrable : pour toute suite $A_1, A_2, \ldots, A_n \ldots$ d'évènements de $\mathcal{A}$ qui sont de plus deux à deux disjoints, c'est-à-dire tels que $A_k \cap A_j = \emptyset$ si $k \neq j$, alors la série $\sum_{k=1}^{\infty} P(A_k)$ converge et a pour somme $P(\bigcup_{k\geq 1} A_k)$.*

Le triplet $(\Omega, \mathcal{A}, P)$ est alors appelé un espace de probabilité. On dit aussi que P est une loi de probabilité.

Exemple 4.4: *Reprenons l'exemple du lancé de dé, on a pour tout ensemble $A \subset \Omega = \{1, 2, 3, 4, 5, 6\}$, la fonction*

$$P(A) = \frac{|A|}{|\Omega|} = \frac{|A|}{6}$$

est une mesure de probabilité sur $(\Omega, P(\Omega))$

Propriété 4.6:
Soit l'espace de probabilité $(\Omega, \mathcal{A}, P)$. Si A et B sont mesurables et si pour tout $n \in \mathbb{N}$, les A_n sont mesurables, alors :

- *$A \subset B \Rightarrow P(A) \leq P(B)$.*
- *$P(A \cup B) = P(A) + P(B) - P(A \cap B)$.*
- *$P(\bigcup_{n\in\mathbb{N}} A_n) \leq \sum_{n\in\mathbb{N}} P(A_n)$.*

Exemple 4.5: *Dans le lancer de dé, si on note*

1. *A l'événement "Obtenir 6", c'est-à-dire $A = \{6\}$*
2. *B l'événement "Obtenir un résultat supérieur ou égal à 2", c'est-à-dire $B =$*

$\{2, 3, 4, 5, 6\}$

On a

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) = \frac{1}{6} + \frac{5}{6} - \frac{1}{6} = \frac{5}{6}$$

On constate que la formule tient compte du fait que A est déjà compté dans B et permet de ne pas le compter deux fois.

Définition 4.7:
Si $(\Omega, \mathcal{A}, P)$ est un espace de probabilité, soit $B \in \mathcal{A}$ un évènement tel que $P(B) > 0$. On définit alors la nouvelle probabilité P_B sur $\mathcal{A}$ par

$$P_B(A) = \frac{P(A \cap B)}{P(B)},$$

qu'on note aussi $P_B(A) = P(A|B)$, et qui se lit "probabilité de A conditionnée par B", ou "sachant B", ou "sachant que B est réalisé".

Définition 4.8:
Sur un espace probabilisé $(\Omega, \mathcal{A}, P)$, deux évènements A, B sont dit indépendants si

$$P(A \cap B) = P(A) \cdot P(B)$$

.

Deux évènements sont dits *indépendants* si la probabilité que A soit vrai ou non n'est pas influencée par le fait que B soit vrai. On a alors $P(A \cap B) = P(A)P(B)$ et $P(A|B) = P(A)$.

Exemple 4.6: *Toujours pour l'expérience du lancer de dé, considérons les trois évènements suivants :*

A : ω est pair
B : ω est multiple de 3
C : ω est inférieur ou égale à 3

On a, pour la mesure de probabilité P définie à l'exemple 4.4, $P(A) = 3/6$, $P(B) = 2/6$, $P(C) = 3/6$. Si on calcule les probabilités conditionnelles associées, on a :

$$P(A|B) = \frac{1/6}{2/6} = P(A), \quad P(A|C) = \frac{1/6}{3/6} \neq P(A), \quad P(B|C) = \frac{1/6}{3/6} = P(B)\ .$$

On constate donc que les évènements A et B sont indépendants, de même que les évènements B et C, mais pas les évènements A et C.

De plus, B est indépendant des évènements A et C pris séparément, mais clairement pas de l'évènement $A \cap C$, car $P(B|A \cap C) = 0$.

Définition 4.9:
On appelle variable aléatoire toute fonction mesurable d'un espace probabilisé $(\Omega, \mathcal{A}, P)$ à valeur dans $\mathbb{R}$ (muni de sa tribu borélienne définit comme la plus petite tribu sur $\mathbb{R}$ contenant tous les intervalles).

Il est d'usage d'utiliser $X, Y, ...$ pour noter des variables aléatoires. Une variable aléatoire sert à associer aux résultats observables $\omega \in \Omega$ une certaine valeur, fonction de ω, qui va nous intéresser.

Exemple 4.7: *Supposons que l'on s'intéresse toujours à un lancer de dés, mais cette fois dans le cadre d'un jeu, le but étant de faire le plus grand "score" avec les dés sur un total de trois lancés. Soit $\Omega = \{a, b, c\}$, on va alors associer à chaque résultat un score (par exemple la somme des valeurs de dés), et la fonction*

$$\begin{array}{rccl} X: & \Omega & \mapsto & \mathbb{N} \\ & (a,b,c) & \to & a+b+c \end{array}$$

est une variable aléatoire.

4.2.2 Les Lois de probabilité

Soit X une variable aléatoire réelle(c'est-à-dire $X : \Omega \to \mathbb{R}$), définie sur un espace de probabilité $(\Omega, \mathcal{A}, P)$.

Définition 4.10:
On appelle fonction de répartition de X, et on note F^X, la fonction sur $\mathbb{R}$ définie par

$$F^X(x) = P[X \leq x] \quad x \in \mathbb{R}\ .$$

Une loi de probabilité, ou distribution, décrit les répartitions typiques des fréquences d'apparition des résultats d'un phénomène aléatoire. Elle se caractérise de différentes manières. Le plus souvent, on utilise la fonction de répartition pour caractériser une loi. Dans le cas discret, les probabilités élémentaires suffisent à caractériser cette loi.

Exemples de lois discrètes
Les résultats de la variable aléatoire X sont discrets, on peut définir leur probabilité

$$P(X = n)$$

On peut citer la loi de Bernoulli, qui correspond à un lancer de pile ou face :

$$\begin{array}{rcl} P(X=1) & = & p \\ P(X=0) & = & q \end{array}$$

Il existe aussi la loi binomiale qui correspond à N épreuves de Bernoulli identiques :

$P(X = k) \quad = \quad C_n^k p^k (1-p)^{n-k}$ pour tout k de 0 à n, p étant un réel compris entre 0 et 1

Nous pouvons aussi citer comme règle connu la loi de Poisson (cf. fig4.4).

$$P(X = n) \quad = \quad \frac{\lambda^n}{n!} \exp(-\lambda) \qquad n \in \mathbb{N},\ \lambda \in \mathbb{R}_+^*$$

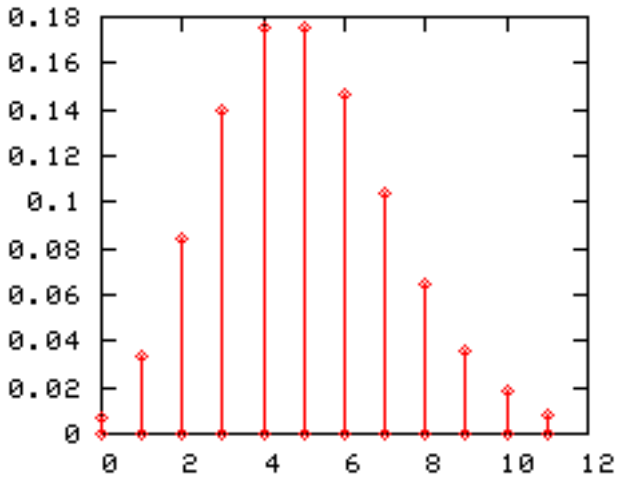


Figure 4.4 — Exemple de distribution discrète (loi de poisson)

A partir de ces distributions, il est possible de définir les moyennes et de trouver les probabilités des évènements. Cependant dans le cas des probabilités, il est souvent nécessaire de pouvoir calculer la probabilité de deux évènements. Or les distributions, qu'elles soient discrètes ou continues, ne représentent qu'un seul évènement. Il est donc utile de définir un outil permettant d'obtenir une distribution à partir d'autres distributions. Cet outil se nomme le produit de convolution.

4.2.3 Produit de convolution

En mathématiques et, en particulier, dans l'analyse de fonction, un produit de convolution est un opérateur qui prend 2 fonctions f et g, que l'on va considérer discrètes et produit une troisième fonction qui, en un sens, représente la quantité de chevauchement entre f et une version inversée et translatée de g.

Définition 4.11:
Le produit de convolution de deux fonctions discrètes f et g s'écrit :

$$(f * g)(x) = \sum_{k=-\infty}^{\infty} f(k)g(x-k)$$

On peut considérer cette formule comme une généralisation de l'idée de moyenne mobile. C'est un type de moyenne statistique utilisé pour analyser des séries ordonnées de données, le plus souvent des séries temporelles, en supprimant les fluctuations transitoires de façon à en souligner les tendances à plus long terme. Cette moyenne est dite mobile parce qu'elle est recalculée de façon continue, en utilisant à chaque calcul un sous-ensemble d'éléments dans lequel un nouvel élément remplace le plus ancien ou s'ajoute au sous-ensemble.

Le produit de convolution permet, dans notre cas, de construire la distribution de deux actions consécutives, et ainsi d'obtenir la distribution d'une application entière.

Ces définitions générales vont nous permettre de créer un modèle de temps basé sur les probabilités.

4.3 Politique de gestion de qualité *stochastique*

La politique de gestion de qualité proposée dans cette partie est une adaptation stochastique de la politique de gestion de qualité mixte. Elle est obtenue de cette dernière en remplaçant les durées d'exécutions pire-cas, par des estimations pire-cas stochastiques.

Dans cette section, nous allons présenter le nouveau problème de gestion de qualité, en introduisant des *distributions* définies pour chaque action et chaque qualité et un *seuil de tolérance* qui nous permettra de calculer les durées d'exécution pire-cas stochastiques.

4.3.1 Définition du problème

Nous allons, comme pour le chapitre 3, ne prendre en compte que le Gestionnaire de Qualité. En effet dans cette partie nous ne parlerons pas des principes d'ordonnancement.

Tout d'abord nous définissons l'espace de probabilité $(\Omega, \mathcal{A}, P)$ que nous allons traiter tout au long de cette étude :

- L'ensemble des observables représente l'ensemble des durées d'exécutions possibles associées à une action, c'est-à-dire $\Omega = \mathbb{R}^+$,
- $\mathcal{A}$ représente les intervalles des durées d'exécution des actions,
- soit $p : \Omega \rightarrow [0,\ 1]$ satisfaisant $\sum_{\omega \in \Omega} p(\omega) = 1$, on a la mesure de probabilité $P : \mathcal{A} \rightarrow [0,\ 1]$ définit par : $P(A) = \sum_{\omega \in A} p(\omega)$.

La technique de contrôle développée dans la section 2.2.3 nécessite de connaître les durées d'exécution moyennes et pire-cas. Nous ne modifierons pas la politique mixte, cependant nous adapterons cette technique en prenant un modèle de temps différent basé sur des distributions.

4.3.1.1 Modèle de temps stochastique

Dans ce chapitre, il s'agit de donner à l'utilisateur une méthode permettant de maîtriser le comportement de son application. Pour cela nous lui proposons de gérer le modèle de temps fourni au contrôleur par le biai des probabilités sur les durées d'exécution pire-cas. Nous allons donc construire un *modèle de temps stochastique*.

Définition 4.12 (Fonction de distribution de probabilité):
*Pour chaque action $a_i \in A$ et chaque niveau de qualité $q \in Q$, nous supposons que $d^q_{a_i} : \mathbb{N} \rightarrow [0,1]$ est la **fonction de distribution de probabilité** de $C(a_i, q)$, c'est-à-*

dire :

$$d_{a_i}^q(k) = \mathrm{P}\big[C(a_i, q) = k\big] \qquad (\text{avec } d_{a_i}^q = 0 \text{ presque partout})$$

où $\mathrm{P}\big[C(a_i, q) = k\big]$ *représente la probabilité de* $C(a_i, q) = k$*, et satisfait :*

$$\sum_{k=0}^{+\infty} d_{a_i}^q(k) = 1 \ .$$

Soit une fonction de distribution de probabilité $d : \mathbb{N} \to [0, 1]$, nous représentons par $len(d) \in \mathbb{N} \cup \{ +\infty \}$ l'index maximal pour lequel d n'est pas nul, c'est-à-dire, $len(d) = \mathbf{max}_{k \in \mathbb{N}} d(k) \neq 0$. Nous pouvons donc dire qu'une distribution d est *finie* si $len(d) < +\infty$. Dans la suite, nous considérerons que les distributions de probabilité $d_{a_i}^q$ sont finies.

Pour obtenir ces distributions, il faut avoir un échantillon des durées d'exécution représentatif de notre application. A partir de là il est simple d'obtenir une distribution représentant la répartition des fréquences d'apparition des durées d'exécution.

Nous allons, à partir de ces distributions, définir un système paramétré incertain stochastique $SPIS(C) = (G, Q, d, D, C)$.

Définition 4.13 (système paramétré incertain stochastique):
Un ***système paramétré incertain stochastique*** *est un n-uplet* $SPIS(C) = (G, Q, d, D, C)$ *où :*

- G *est un* ***graphe de précédence****.*
- $Q = [q_{min}, q_{max}]$ *est un intervalle fini d'entiers correspondant aux* ***niveaux de qualité****.*
- *Pour chaque action* $a_i \in A$ *et chaque niveau de qualité* $q \in Q$*, nous supposons que* $d_{a_i}^q : \mathbb{N} \to [0, 1]$ *est la* ***fonction de distribution de probabilité*** *de* $C(a_i, q)$*.*
- $D : A \to \mathbb{R}^+ \cup \{ +\infty \}$ *est une fonction qui pour une action* a *donne son* ***échéance*** $D(a)$*.*
- *Le paramètre* $C : A \times Q \to \mathbb{R}^+$ *est une fonction qui pour une action* a *et un niveau de qualité* q *donne sa durée d'exécution* ***réelle*** $C(a, q)$*. Nous supposons que, pour tout* $a \in A$*,* $q \mapsto C(a, q)$ *est une fonction croissante telle que* $C \leq C^{wc}$*.*

Le contrôleur nécessite de connaître la durée d'exécution moyenne C^{av}. Pour une distribution elle est représentée par la moyenne pondérée ou l'espérance.

Soit une action a_i et un niveau de qualité q, nous définissons la durée d'exécution moyenne par la valeur moyenne $\mathrm{E}(d_{a_i}^q)$ de la fonction de distribution de probabilité $d_{a_i}^q$. Ceci est conforme à l'intuition de la valeur attendue pour une fonction de distribution discrète.

Définition 4.14 (fonction de durée d'exécution moyenne):
Soit un système paramétré incertain $SPIS(C)$*, nous définissons la* ***fonction de durée***

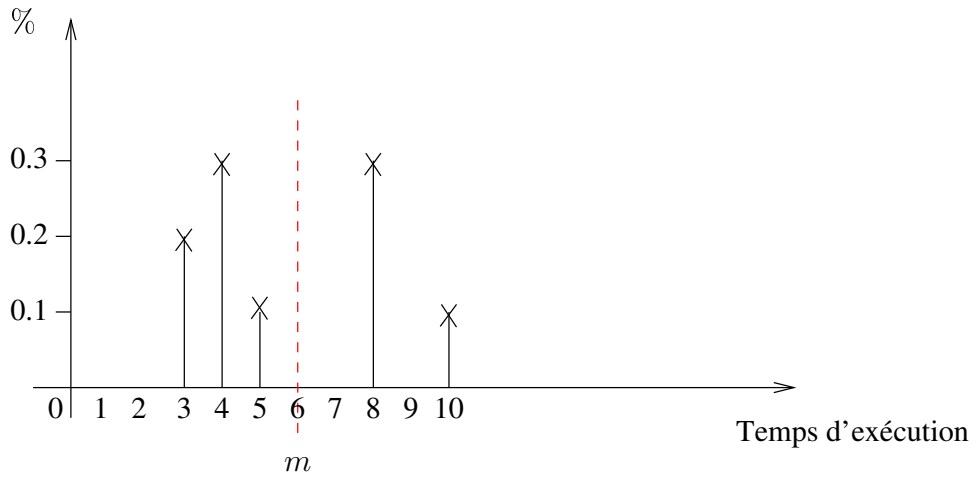


Figure 4.5 — Exemple de moyenne pondérée

d'exécution moyenne $C^{av} : A \times Q \to \mathbb{N}$ *par :*

$$C^{av}(a_i, q) = \mathrm{E}(d^q_{a_i}) = \sum_{k=0}^{+\infty} k d^q_{a_i}(k) \ .$$

Exemple 4.8: *Pour expliquer ce calcul de la moyenne, prenons un exemple de distribution discrète (cf. fig4.5). Cette figure nous montre une distribution de cinq durées d'exécution. Par exemple nous avons une probabilité de 0,2 que la durée d'exécution de l'action soit égale à 3 unités de temps. A partir de la nous voulons calculer la moyenne attendue* C^{av} *de cette distribution.*

$$\begin{aligned} C^{av}(a_i, q) &= \mathrm{E}(d^q_{a_i}) \\ &= \sum_{k=0}^{10} k d^q_{a_i}(k) \\ &= 0 * d^q_{a_i}(0) + \ldots + 10 * d^q_{a_i}(10) \\ &= 3 * 0\,.\,2 + 4 * 0\,.\,3 + 5 * 0\,.\,1 + 8 * 0\,.\,3 + 10 * 0\,.\,1 \\ C^{av}(a_i, q) &= 5\,.\,7\,. \end{aligned}$$

Nous allons maintenant définir la durée d'exécution pire-cas stochastique. Son calcul est paramétré par un *seuil de tolérance* $\tau \in [0, 1]$. Cette estimation pire-cas est calculée de telle sorte que la probabilité de les dépasser est inférieure à τ. Cela conduit à la définition suivante.

Définition 4.15 (Fonction de durée d'exécution pire-cas stochastique):
Soient A l'ensemble des actions d'un graphe de précédence G, Q un ensemble de niveaux de qualité et un seuil de tolérance τ*. Nous définissons la* ***fonction de durée d'exécution***

pire-cas stochastique $C_\tau^{wc} : A \times Q \to \mathbb{N}$ *telle que* $\mathrm{P}\big[C(a_i, q) > C_\tau^{wc}(a_i, q)\big] \leq \tau$, *c'est-à-dire :*

$$C_\tau^{wc}(a_i, q) = \min\{k \geq 0 \mid \big[\sum_{l=k+1}^{+\infty} d_{a_i}^q(l) \leq \tau\big]\} .$$

4.3.1.2 Problème de gestion de qualité stochastique

Ces nouvelles données vont permettre de définir un nouveau problème de gestion de qualité. En effet, du fait de l'utilisation de l'utilisation de durées d'exécution pire-cas stochastiques, la propriété de sûreté ne peut plus être garantie. Ainsi nous devons laisser la possibilité à l'application contrôlée de violer des échéances, mais en s'assurant de ne pas dépasser un certain taux de violation.

Définition 4.16 (taux de violation des échéances):
Soit un système paramétré incertain stochastique $SPIS(C)$. Nous représentons par $\mu(\Gamma_\tau^{mx})$ le ***taux de violation des échéances*** *pour le Gestionnaire de Qualité Γ_τ^{mx}. Il est défini par la probabilité de violer l'échéance critique de $SPIS(C)||\Gamma_\tau^{mx}$, c'est-à-dire $\mu(\Gamma_\tau^{mx}) = \mathrm{P}\big[t_n \geq D\big]$ où t_n est la date de terminaison de la dernière action de $SPIS(C)||\Gamma_\tau^{mx}$.*

Le problème de gestion de qualité pour un système paramétré incertain stochastique $SPIS(C)$ consiste à trouver un Gestionnaire de Qualité Γ_τ^{mx} satisfaisant les propriétés de qualité de service. C'est-à-dire que les échéances peuvent être violées avec une probabilité minimale et une qualité optimale, et en ayant une affectation de niveau de qualité régulière.

Définition 4.17 (Problème de gestion de qualité stochastique):
Soient un système paramétré incertain stochastique $SPIS(C)$ et un taux ciblé de violation de l'échéance critique $\lambda \in [0, 1]$. Trouver un Gestionnaire de Qualité Γ_τ^{mx} tel que :

- *Γ_τ^{mx} possède un taux de violation des échéances qui est au maximum égal à λ, c'est-à-dire, $\mu(\Gamma_\tau^{mx}) \leq \lambda$.*
- *Le temps d'exécution global est maximal, c'est-à-dire que pour chaque Gestionnaire de Qualité $\Gamma_\tau'^{mx}$ avec un taux de violation des échéances maximal de λ, nous avons $t_n \geq t'_n$, où t_n (resp. t'_n) est la date de terminaison de la dernière action de $SPIS||\Gamma_\tau^{mx}$ (resp. $SPIS||\Gamma_\tau'^{mx}$).*
- *L'affectation du niveau de qualité est régulière. Par exemple, nous chercherons à ce que l'écart type ou la variance de l'affectation soit minimal.*

Il s'agit maintenant de trouver une politique de gestion de qualité permettant de résoudre ce problème. Nous allons donc dans la section suivante proposer une politique de contrôle mixte stochastique qui vérifie que le taux de violation des échéances est au maximum égale à λ.

4.3.2 Politique de gestion de qualité stochastique

Dans cette section nous proposons une politique de gestion de qualité mixte stochastique C^{mx}_{τ} répondant au problème ci-dessus. Cette politique est une adaptation de la politique de gestion de qualité mixte C^{mx} présentée dans la section 2.2.3. Elle est basée sur un paramètre τ, que l'on nomme seuil de tolérance. Celui-ci va permettre par son adaptation de générer un contrôleur Γ^{mx}_{τ} tel que le taux de violation des échéances est au maximum égal à λ. Il existe donc un lien entre le seuil de tolérance et le taux de violation des échéances. Nous étudierons ce lien dans la section 4.4.

Définition 4.18 (politique de gestion de qualité stochastique):
*La **politique de gestion de qualité stochastique** C^{mx}_{τ} considérée est une adaptation de la politique de gestion de qualité mixte C^{mx}. Elle est obtenue de cette dernière en remplaçant les durées d'exécution pire-cas par les durées d'exécution pire-cas stochastiques, c'est-à-dire,*

$$C^{mx}_{\tau} = C^{av} + \delta^{max}_{\tau}$$

où $\delta^{max}_{\tau}(a_i \,..\, a_n, q) = \mathbf{max}\ \{\ 0\ \} \cup \{\ C^{sf}_{\tau}(a_j \,..\, a_n, q) - C^{av}(a_j \,..\, a_n, q) \mid i \leq j \leq n\ \}$ *et* $C^{sf}_{\tau}(a_j \,..\, a_n, q) = C^{wc}_{\tau}(a_i, q) + C^{wc}_{\tau}(a_{i+1}, q_{min}) + \,..\, + C^{wc}_{\tau}(a_n, q_{min})$.

Puisque la durée d'exécution réelle d'une action peut dépasser les estimations pire-cas stochastiques, l'utilisation de la politique de gestion de qualité stochastique peut mener à des échéances violées. Nous allons montrer le fonctionnement du seuil de tolérance et son influence quant au respect des échéances.

En fixant le τ à 0, cela implique que la probabilité de dépasser les durées d'exécution pire-cas C^{wc}_0 est nulle. En d'autres termes, C^{wc}_0 correspond aux durées d'exécution pire-cas exactes, c'est-à-dire que le temps réel ne les dépassera jamais, sous l'hypothèse que la fonction de distribution de probabilité $d^q_{a_i}$ est un modèle fidèle de durées d'exécution des actions. Pour $\tau = 0$ la politique de gestion de qualité stochastique est donc équivalente à la politique mixte et ainsi aucune échéance n'est violée.

Au contraire, la fonction de durée d'exécution pire-cas stochastique C^{wc}_{τ} tend vers 0 si τ tend vers 1. Ainsi, l'utilisation d'un $\tau = 1$, c'est-à-dire $C^{wc}_1 = 0$, correspond à la politique de gestion de qualité moyenne. C'est-à-dire qu'elle ne cherche pas à réduire le taux des échéances violées mais conduit simplement à une utilisation maximale du budget de temps.

La politique de gestion de qualité stochastique permet une flexibilité dans la combinaison entre les comportements moyens et les comportements pire-cas. Le seuil de tolérance déterminera la manière dont le comportement pire-cas est pris en compte par la politique stochastique. Cela peut varier entre ne pas tenir compte des estimations pire-cas, ce qui correspond à des comportements de type temps-réel mou ($\tau = 1$), et au contraire prendre en compte le pire-cas réel, c'est-à-dire répondre aux attentes du temps réel dur, aucune échéance ne sera manquée ($\tau = 0$). En un sens, le paramètre τ est utilisé pour exprimer

la souplesse de l'échéance D.(voir la figure 4.6).

Figure 4.6 — Influence du seuil de tolérance τ

Notre méthode permet aux programmeurs d'obtenir un compromis entre le taux de violation des échéances et l'utilisation des ressources, en choisissant le seuil de tolérance.

Nous allons prendre un exemple d'utilisation de cette technique permettant de comprendre plus explicitement l'utilisation du seuil de tolérance. L'utilisateur décide de permettre à son application de dépasser les échéances dans 10% des situations. Pour cela il fixera le seuil de tolérance à 0.1. C'est-à-dire, l'ensemble des valeurs supérieur au C_τ^{wc} calculé sur la distribution correspond à 10% des valeurs possibles que peut prendre l'action a_i pour la qualité q. Il s'aperçoit lors de l'exécution de son application que le résultat obtenu n'a pas la qualité qu'il aurait voulue, et que le taux de violation des échéances reste faible. Il peut alors décider de réajuster les seuil de tolérance pour palier aux phénomènes qu'il a rencontrés. Ainsi on peut dire que $\lambda(\Gamma_\tau^{mx})$ est une fonction croissante du seuil de tolérance τ.

Nous voyons bien sur cet exemple qu'il serait nécessaire de connaître à l'avance le taux d'utilisation des ressources attendues et le taux de violation des échéances attendu en utilisant ce seuil de tolérance. Ainsi dans la section suivante, nous allons définir une relation entre le paramètre τ et le taux de dépassement des échéances.

4.4 Analyse de performances

Cette section va présenter deux sous parties. La première va donner une relation entre le seuil de tolérance et le taux de violation des échéances. La deuxième va proposer une technique d'analyse de performances permettant au programmeur de maîtriser l'application contrôlée.

4.4.1 Calcul du taux de violation des échéances

Dans cette partie nous présentons la politique de gestion de qualité stochastique. Celle-ci est une extension de la politique mixte présentée à la section 2.2.3. Dans un deuxième temps nous essaierons de calculer la probabilité de dépasser l'échéance critique, en donnant tout d'abord une condition nécessaire à ce dépassement, puis en proposant un calcul de cette probabilité.

4.4.1.1 Condition de violation des échéances

L'objectif de l'approche stochastique est de fournir à l'utilisateur un moyen de contrôler le taux d'échéances violées grâce au seuil de probabilité τ. Pour connaître cette probabilité, il faut savoir dans quels cas les contraintes de temps sont dépassées. L'étude du comportement de l'application va permettre d'entreprendre le calcul de la probabilité d'atteindre ces conditions de dépassement. Et ainsi nous pourrons calculer un taux de violation des contraintes de temps par rapport au seuil de tolérance.

Nous savons par construction de la politique mixte que $t_p^{mx} \leq t_p^{sf}$. De plus la politique mixte utilise la construction de la politique *sûre* pour récupérer le système dans les cas d'exécution "mauvaises". Nous pouvons donc en déduire que pour dépasser les échéances, il faut que la politique *sûre* ne puisse être satisfaite. A partir de là, nous pouvons distinguer deux situations de non-respect des échéances. Nous définissons la politique de gestion de qualité sûre stochastique par t_τ^{sf} basée sur la fonction de durée d'exécution pire-cas stochastique C_τ^{sf} et sur la politique de gestion de qualité sûre t_p^{sf}.

Nous allons présenter deux cas illustrés respectivement par les figures 4.7 et 4.8. Ces deux cas posent les bases d'une bonne compréhension des conditions de dépassement.

Ces deux figures représentent l'exécution d'un ensemble d'actions $a_1 \ldots a_n$. Toutes ces actions doivent être réalisées avant d'atteindre une échéance D. Celle-ci est la même pour chaque action, c'est à dire $D = D(a_1) = \ldots = D(a_n)$. Pour chaque action nous notons la qualité qui a été choisie par le contrôleur, et au dessus des actions nous représentons l'ensemble $\mathcal{Q}_i$ des qualités satisfaisant la relation :

$$\mathcal{Q}_i = \{q | t_\tau^{mx}(a_i \ldots a_n, s_{i-1}, q) \geq t_{i-1}\} .$$

Maintenant que la représentation de l'exécution a été introduite, nous allons montrer les deux situations de "mauvaises" exécutions. C'est-à-dire les deux situations où l'application ne respecte pas les échéances

Premier cas : $t_\tau^{sf}(a_n, s_{n-1}, q_{min}) \geq t_{n-1}$, c'est à dire entre l'exécution de a_{n-1} et celle de a_n

Lors du dernier point de contrôle, le contrôleur choisit la qualité q_{min} mais en exécutant l'action a_n, l'échéance est dépassée(cf.4.7), c'est-à-dire $t_n > D$. Nous allons nommer cette évènement Δ et le définir par :

$$\Delta = t_n > D \quad \wedge \quad t_\tau^{sf}(a_n, s_{n-1}, q_{min}) \geq t_{n-1}$$

Cet évènement est possible du fait de l'utilisation des durées d'exécution pire-cas stochastiques. En effet cela laisse la possibilité à l'application de dépasser la durée d'exécution pire-cas que le contrôleur avait prévu.

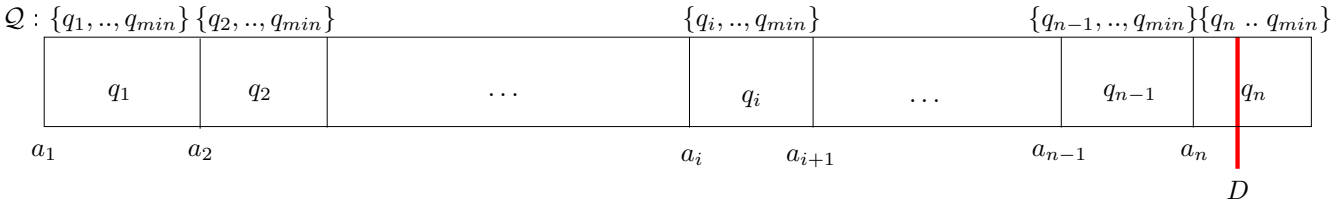


Figure 4.7 — échec lors de la dernière action

Dans ce cas de figure, l'application contrôlée atteint un état du système (s_n, t_n) vérifiant :

$$\begin{aligned} t_n &= t_{n-1} + C(a_n, q_{min}) > D \\ D &\geq t_{n-1} + C^{sf}_{\tau}(a_n, q_{min}) \end{aligned}$$

car q_n vérifie :

$$\begin{aligned} & t^{mx}_{\tau}(a_n, s_{n-1}, q_{min}) \geq t_{n-1} \\ \Rightarrow\ & t^{sf}_{\tau}(a_n, s_{n-1}, q_{min}) \geq t_{n-1} \text{ par construction de } t^{mx}_{p} \\ \Rightarrow\ & D - C^{sf}_{\tau}(a_n, q_{min}) \geq t_{n-1} \\ \Rightarrow\ & D - C^{wc}_{\tau}(a_n, q_{min}) \geq t_{n-1} \\ \Rightarrow\ & D \geq t_{n-1} + C^{wc}_{\tau}(a_n, q_{min}) \ . \end{aligned}$$

On peut écrire :

$$\begin{aligned} t_{n-1} + C(a_n, q_{min}) &> D \\ t_{n-1} + C(a_n, q_{min}) &> t_{n-1} + C^{wc}_{\tau}(a_n, q_{min}) \end{aligned}$$

Donc on peut en conclure que dans ce cas, l'application va dépasser l'échéance, si :

$$C(a_n, q_{min}) > C^{sf}_{\tau}(a_n, q_{min}) = C^{wc}_{\tau}(a_n, q_{min})$$

Autrement dit, si l'échéance n'est pas respectée la durée d'exécution réelle de la dernière action est plus grande que la durée d'exécution pire-cas stochastique. Ainsi la probabilité de dépasser l'échéance mais d'avoir $t^{sf}(a_n, s_{n-1}, q) \geq t$ est :

$$P[\Delta] = P[C(a_n, q_{min}) > C^{wc}_{\tau}(a_n, q_{min})] = \tau$$

Deuxième cas : $t^{sf}_{\tau}(a_n, s_{n-1}, q_{min}) < t_{n-1}$

Cette fois-ci, le contrôleur peut choisir uniquement la qualité minimale par défaut.

C'est-à-dire qu'à partir d'un certain état i, on a la relation $t_\tau^{sf}(a_i \dots a_n, s_{i-1}, q_{min}) < t$ pour tout un ensemble de valeur i. Nous allons noter cet évènement Δ_i, et le décrire par :

$$\Delta_i = t_n > D \quad \wedge \quad t_\tau^{sf}(a_n, s_{n-1}, q_{min}) < t_{n-1} \wedge i = j | \forall k > j \ \mathcal{Q}_k = \emptyset$$

Du fait de cet évènement le contrôleur décide d'utiliser le niveau de qualité q_{min} pour tous les i, c'est-à-dire $t_\tau^{sf}(a_i \dots a_n, s_{i-1}, q_{min}) < t$. Cependant même en utilisant ce niveau de qualité, l'exécution dépasse l'échéance (cf. figure 4.8).

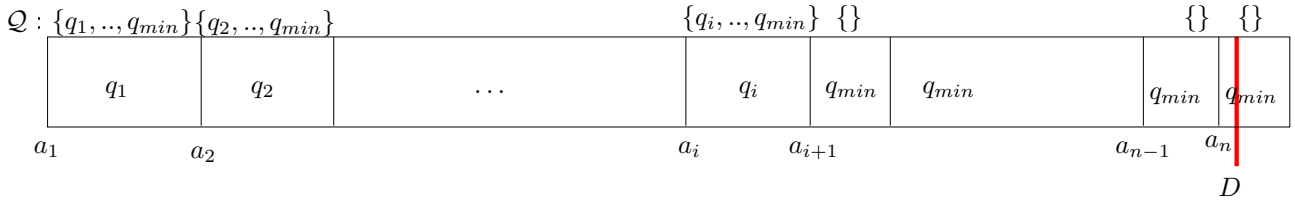


Figure 4.8 — échec lors d'un ensemble d'actions

La figure 4.8 nous montre les choix faits par le contrôleur au niveau de la qualité. Jusqu'à l'action i, le déroulement est normal, le contrôleur peut faire des choix de qualité satisfaisants. A partir de l'état $i+1$, nous pouvons observer qu'aucune qualité ne vérifie $t_\tau^{sf} \geq t$. Par conséquent le contrôleur décide d'utiliser la qualité minimale pour les actions suivantes jusqu'à ce qu'il trouve un état où il peut choisir une qualité convenable. Prenons le pire des cas, celui où l'évolution de l'application ne permet pas de revenir dans un état proposant un choix de qualité au contrôleur. Donc à partir de l'état i, le système n'a pu "être récupéré".
Plus formellement nous avons pour l'action i la relation :

$$t_{i-1} + C_\tau^{sf}(a_i \dots a_n, q_i) \leq D$$

Mais la réalité ne correspond pas aux prévisions du contrôleur, et nous obtenons pour l'action suivante la relation :

$$t_{i-1} + C(a_i, q_i) + C_\tau^{wc}(a_{i+1} \dots a_n, q_{min}) > D$$

Par définition de la politique de gestion de qualité sûre nous avons donc $C(a_i, q_i) > C_\tau^{wc}(a_i, q_i)$. Cependant pour que l'application n'arrive pas à rattraper le temps perdu, il faut aussi que $C(a_{i+1} \dots a_n, q_{min}) \geq C_\tau^{wc}(a_{i+1} \dots a_n, q_{min})$.

Ainsi nous pouvons calculer la probabilité de l'évènement Δ_i comme suit :

$$P[\Delta_i] = P[C(a_i, q_i) > C_\tau^{wc}(a_i, q_i) \wedge C(a_{i+1} \dots a_n, q_{min}) \geq C_\tau^{wc}(a_{i+1} \dots a_n, q_{min})]$$

Du fait de l'indépendance des actions on a :

$$P[\Delta_i] \leq P[C(a_i, q) > C_\tau^{wc}(a_i, q)] \ . \ P[C(a_{i+1} \dots a_n, q_{min}) > C_\tau^{wc}(a_{i+1} \dots a_n, q_{min})]$$

$P[\Delta_i] \leq \tau \,.\, P[C(a_{i+1} \ldots a_n, q_{min}) > C_\tau^{wc}(a_{i+1} \ldots a_n, q_{min})]$ d'après définition 4.15

Pour que $C(a_{i+1} \ldots a_n, q_{min}) > C_\tau^{wc}(a_{i+1} \ldots a_n, q_{min})$, il faut qu'il existe au moins une action dont la durée d'exécution pour q_{min} soit supérieur à la durée d'exécution pire-cas. Ainsi nous avons donc :

$$P[\Delta_i] \leq \tau^2$$

Les deux situations précédentes nous présentent les conditions de dépassement des échéances pour la politique de gestion de qualité *mixte*. A partir de là, nous allons proposer un calcul de la probabilité de dépasser les échéances en fonction des deux évènements que nous avons cités.

4.4.1.2 Probabilité de dépasser les échéances

La manière de fixer les durées d'exécution pire-cas ne permet pas exactement de maîtriser le comportement de l'application dans le sens où les probabilités sont calculées pour une action et une qualité, et non sur la probabilité de dépasser les contraintes de temps à la fin de l'exécution. Ceci induit donc une maîtrise de l'application locale et non globale. De ce fait il est intéressant de proposer une relation entre le seuil de tolérance et le taux de dépassement des échéances.

La probabilité de dépasser les échéances correspond à l'union des deux évènements décrits précédemment. Ainsi nous pouvons donner la relation formelle de $\lambda(\Gamma_\tau^{mx})$ suivante :

$$\lambda(\Gamma_\tau^{mx}) \;=\; P[\Delta \vee \Delta_1 \vee \ldots \vee \Delta_{n-1}]$$

Du fait de l'indépendance des évènements, nous avons :

$$\begin{aligned}
\lambda(\Gamma_\tau^{mx}) &\leq P[\Delta] + \sum_{i=1}^{n-1} P[\Delta_i] \\
\lambda(\Gamma_\tau^{mx}) &\leq \tau + \sum_{i=1}^{n-1} \tau^2 \\
\lambda(\Gamma_\tau^{mx}) &\leq \tau + (n-1)\tau^2
\end{aligned}$$

Ce calcul est issu de nombreuses approximations. La première réside dans la majoration de la fonction de durée d'exécution mixte par celle de la politique sûre. D'ores et déjà nous n'avions pas la possibilité de donner un résultat exact mais s'en approchant. Une deuxième approximation a été faite lors du calcul de la probabilité de Δ_i. Nous avions majoré cette probabilité en définissant une situation où toutes les durées d'exécution réelles étaient soit égales soit supérieures aux durées d'exécution pire-cas stochastiques. De ce fait nous avons pris un cas d'exécution extrême de l'application. A partir de là, le taux d'échéances violées a été surévalué.

Pour cette raison, nous allons proposer, une méthode permettant de calculer directement le taux de violation des échéances exact. Cette méthode ne sera pas issue d'une simplification du calcul de la probabilité, mais en faisant une construction de la distribution de la date de terminaison t_n de la dernière action a_n. Cette méthode nous permettra aussi de déterminer le taux d'utilisation des ressources attendu.

4.4.2 Analyse de performances

Dans cette section nous proposons une méthode permettant la prédiction du comportement de l'application contrôlée. Nous avons développé un algorithme qui calcule la distribution de probabilité pour la date de terminaison des actions. Ensuite, la fonction de distribution de probabilité de la date de terminaison de la dernière action est utilisée pour calculer le taux de violation de l'échéance attendu. On pourra aussi calculer le taux d'utilisation des ressources en termes de budget de temps.

4.4.2.1 Résultats préliminaires

Soit un système paramétré incertain stochastique $SPIS(C)$ avec un ordonnancement statique $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$. Soient une échéance D et un Gestionnaire de Qualité Γ_τ^{mx}, la date de terminaison t_i de l'action a_i du système contrôlé $SPIS(C)||\Gamma_\tau^{mx}$ est définie récursivement par :

$$\begin{cases} t_0 = 0 \text{ et} \\ t_i = t_{i-1} + C(a_i, q_i), \end{cases}$$

où q_i est seulement déterminé par la valeur de t_{i-1}, c'est-à-dire, $(a_i, q_i) = \Gamma_\tau'^{mx}(s_{i-1}, t_{i-1})$. Donc, t_i est une variable aléatoire discrète. Nous représentons par $d_i : \mathbb{N} \to [0,1]$ la fonction de distribution de probabilité de t_i. Elle est définie par $d_i(k) = \mathrm{P}\big[t_i = k\big]$ où $\mathrm{P}\big[t_i = k\big]$ est la probabilité de $t_i = k$.

Calcul de $\mathrm{P}\big[t_i = k\big]$

Soit k un entier. Nous supposons que $\{\ t_i = k \wedge t_{i-1} = l\ \}_{l\in\mathbb{N}}$ sont des unions disjointes d'évènements. Alors nous obtenons :

$$\begin{aligned} \mathrm{P}\big[t_i = k\big] &= \mathrm{P}\Big[\bigvee_{l=0}^{+\infty} t_i = k \wedge t_{i-1} = l\Big] \\ \mathrm{P}\big[t_i = k\big] &= \sum_{l=0}^{+\infty} \mathrm{P}\big[t_i = k \wedge t_{i-1} = l\big]\ . \end{aligned} \tag{4.1}$$

Comme $t_i = t_{i-1} + C(a_i, q_i)$, nous avons :

$$\mathrm{P}\big[t_i = k \wedge t_{i-1} = l\big] = \mathrm{P}\big[t_{i-1} = l \wedge C(a_i, q_i) = k - l\big] \ .$$

Puisque $t_i = C(a_1, q_1) + \ldots + C(a_{i-1}, q_{i-1})$, nous obtenons :

$$\mathrm{P}\big[t_i = k \wedge t_{i-1} = l\big] = \mathrm{P}\big[t_{i-1} = l\big]\mathrm{P}\big[C(a_i, q_i) = k - l\big] \ .$$

L'équation ci-dessus et (4.1) mènent à la relation suivante entre les distributions :

$$d_i(k) = \sum_{l=0}^{+\infty} d_{i-1}(l) d_{a_i}^{q_i}(k - l) \ . \tag{4.2}$$

4.4.2.2 Algorithme de Convolution paramétré

L'algorithme 4.1 est une implémentation du principe de calcul décrit précédemment, c'est-à-dire de l'équation (4.2). Nous supposons que les valeurs non initialisées sont égales à 0. L'ordre de calcul dans l'algorithme est légèrement différent de celui de (4.2).

```
1  convolution_paramétré({ d_{a_i}^{q_i} }_{1≤i≤n}, Γ_τ^{mx})
2  i = 1
3  d_0(0) = 1
4  for i ← 1 to n do
5  |  foreach l tel que d_{i-1}(l) ≠ 0 do
6  |  |  (a_i, q_i) = Γ_τ^{mx}(s_{i-1}, l)
7  |  |  foreach t tel que d_{a_i}^{q_i}(t) ≠ 0 do
8  |  |  |  d_i(l + t) ← d_i(l + t) + d_{i-1}(l) * d_{a_i}^{q_i}(t)
9  |  |  end
10 |  end
11 end
```

Algorithme 4.1 : Convolution paramétrée

Soit deux entiers l et t, nous incrémentons la valeur de $d_i(l + t)$ par $d_{i-1}(l) d_{a_i}^{q_i}(t)$, ce qui est équivalent à (4.2).

A chaque pas du calcul et pour chaque distribution d_i, le nombre de valeurs $d_i(k)$ différent de 0 est fini. En fait, il est facile de voir que

$$len(d_i) = len(d_{i-1}) + \mathbf{max}_{q_i \in Q} len(d_{a_i}^{q_i}) \tag{4.3}$$

au pire-cas.

Donc calculer d_i à partir de d_{i-1} et $d_{a_i}^{q_i}$ nécessite seulement un nombre fini d'additions au lieu d'une infinité comme précédemment.

L'algorithme de convolution paramétré (cf. algorithme 4.1) prend en entrée les distributions de l'application $\{d_{a_i}^{q_i}\}_{1 \leq i \leq n}$ et le Gestionnaire de Qualité Γ_τ^{mx}. A partir de ces

données, il va en partant d'une distribution d_0 initialisée à 1 calculer itérativement la distribution de l'application pour la date de terminaison. Pour cela il parcourt toutes les actions (ligne 4). Puis pour chaque action, le gestionnaire de qualité calcule le niveau de qualité que le contrôleur assignera à l'action suivante pour la valeur de la durée d'exécution l de l'action courante (ligne 6). Ensuite pour cette valeur l, il réalise le produit de convolution avec chaque valeur de la distribution suivante pour la qualité choisie (ligne 5,7-8).

Grâce à cet algorithme, nous pouvons construire la distribution de probabilité correspondant à la date de terminaison t_n de la dernière action a_n. Nous allons étudier dans la partie suivante la complexité de cet algorithme de convolution. Puis nous donnerons un exemple l'illustrant.

Complexité de l'algorithme

Le nombre d'additions de l'algorithme 4.1 pour le calcul de d_i est dans le pire des cas égal à :

$$\big(1 + len(d_{i-1})\big)\big(1 + \mathbf{max}_{q_i \in Q} len(d_{a_i}^{q_i})\big)$$

Cela signifie que le nombre d'additions maximal pour le calcul de d_n est :

$$\sum_{i=1}^{n} \big(1 + len(d_{i-1})\big)\big(1 + \mathbf{max}_{q_i \in Q} len(d_{a_i}^{q_i})\big) \ .$$

Le comportement pire-cas est donné par :

$$(N+1) \sum_{i=1}^{n} \big(1 + len(d_{i-1})\big),$$

où $N = \mathbf{max}_{a_i \in A, q_i \in Q} len(d_{a_i}^{q_i})$ est la longueur maximale des distributions de durées d'exécution $C(a_i, q_i)$. Cela correspond à une dimension caractéristique des entrées du problème. En utilisant (4.3) et $len(d_0) = 0$, nous obtenons dans le pire des cas $len(d_i) = iN$. Ainsi le nombre maximal d'additions est de :

$$(N+1) \sum_{i=1}^{n} \big((i-1)N + 1\big) = \frac{1}{2} N(N+1)n(n-1) = \mathcal{O}(N^2 n^2) \ .$$

Cela signifie que l'algorithme proposé est polynomial par rapport au nombre d'actions et à la longueur des distributions des durées d'exécution. Cependant, cela peut être accéléré avec l'utilisation des transformées de Fourier pour le calcul des convolutions.

Exemple d'exécution de l'algorithme

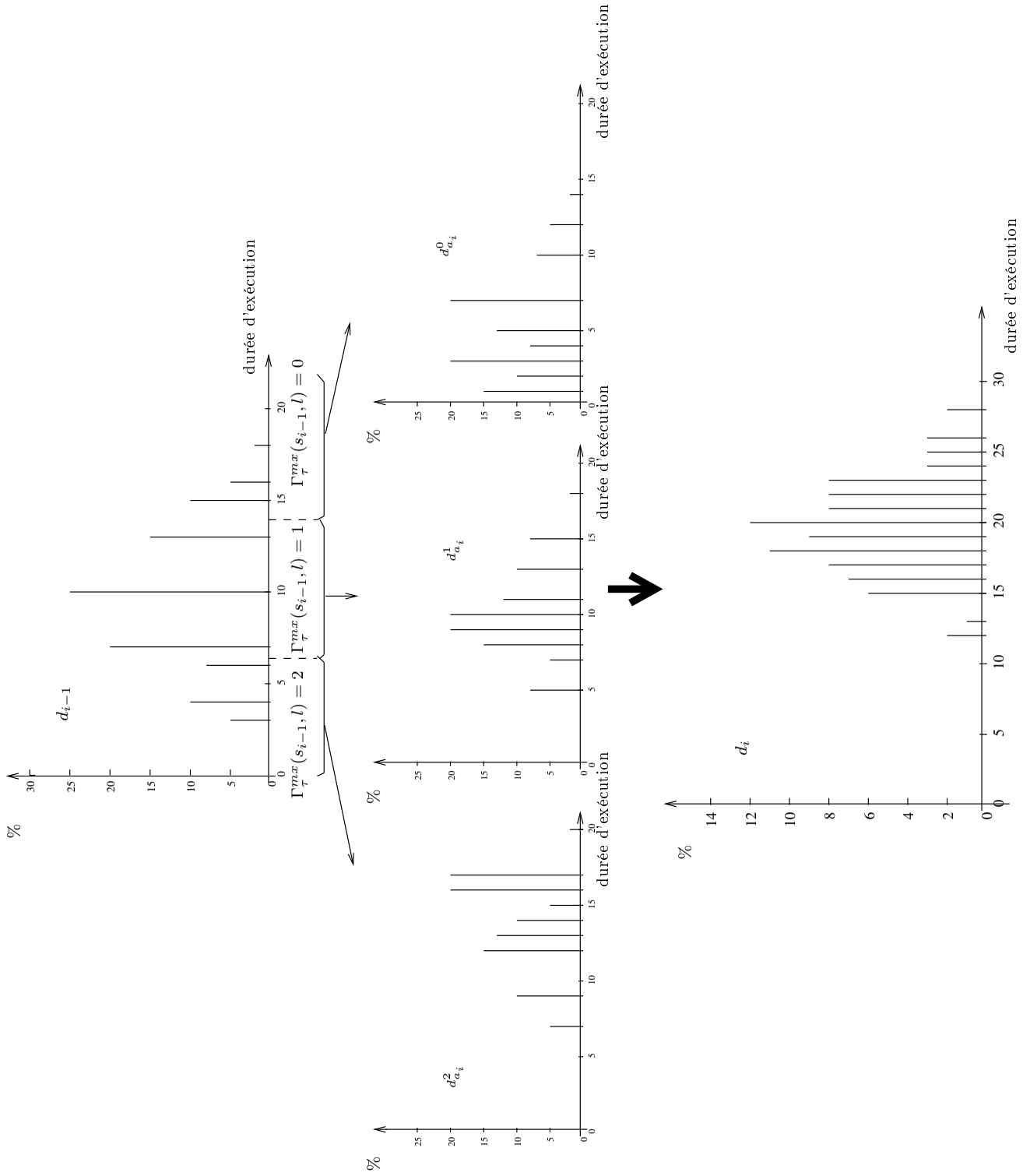


Figure 4.9 — Exemple de convolution paramétrée

L'exemple de la figure 4.9 va nous permettre de mieux comprendre l'intuition de la convolution paramétrée.

Exemple 4.9: *La figure 4.9 nous présente l'exécution de l'algorithme pour l'action a_{i-1}, c'est-à-dire que la distribution d_{i-1} a déjà été calculée. Nous considérons dans cet exemple une application ayant uniquement trois qualités. Nous avons représenté le choix de qualité du contrôleur Γ_τ^{mx} sur la distribution d_{i-1}. A partir de là, l'algorithme fait la convolution de toutes les valeurs où le Gestionnaire de Qualité choisit la qualité 2 avec la distribution de l'action suivante a_i pour la qualité 2. Il fait de même pour les qualités 1 et 0. Ainsi il obtient la distribution d_i, correspondant à la date de terminaison t_i de l'action a_i.*

Cet algorithme nous permet de construire la distribution des durées d'exécution pour la date de terminaison t_n de l'application. Elle dépend du seuil de tolérance qui conditionne les durées d'exécution pire-cas utilisées par le Gestionnaire de Qualité. Cette distribution permet de réaliser des analyses à priori sur le comportement de l'application contrôlée.

4.4.2.3 Étude de performances

Dans cette section nous allons nous intéresser aux performances de l'application contrôlée. Nous proposons deux types de résultats, le premier concerne le taux de violation de l'échéance, le deuxième nous donnera des indications sur le budget de temps que devrait utiliser l'application.

Dans la suite, nous considérons une échéance critique D et un Gestionnaire de Qualité Γ_τ^{mx} qui applique la politique de gestion de qualité stochastique C_τ^{mx}. Le Gestionnaire de Qualité proposé permet de répondre à deux caractéristiques bien différentes :

- la première est de respecter les contraintes temporelles, c'est-à-dire que le temps d'exécution total t_n est inférieur à l'échéance D avec une probabilité minimale de λ.
- la deuxième est d'avoir une utilisation optimale des ressources, c'est-à-dire maximiser le niveau de qualité choisi par le Gestionnaire de Qualité. Cela peut être formalisé par la maximisation du temps d'exécution total t_n (cf. section 2).

Le programmeur d'application multimédia peut grâce à cette méthode choisir un compromis en les deux caractéristiques. Ainsi l'analyse de performances va être faite pour calculer le taux de violation des échéances, mais aussi sur l'utilisation du budget de temps attendu.

Taux de violation de l'échéance

Le taux de violation de l'échéance $\mu(\Gamma_\tau^{mx})$ pour le Gestionnaire de Qualité Γ_τ^{mx} est

défini par

$$\mu(\Gamma_\tau^{mx}) = \mathrm{P}\left[t_n \geq D\right] .$$

Puisque d_n est la fonction de distribution des probabilités pour la variable aléatoire t_n, le taux de violation de l'échéance pour l'application contrôlée est défini par :

$$\mu(\Gamma_\tau^{mx}) = \sum_{k=D+1}^{+\infty} d_n(k) .$$

Notons que du fait que la distribution d_n est finie, la somme infinie ci-dessus peut être réduite à la somme suivante :

$$\mu(\Gamma_\tau^{mx}) = \sum_{k=D+1}^{len(d_n)} d_n(k) .$$

Taux d'utilisation du budget de temps attendu

Le taux d'utilisation du budget de temps attendu est calculé comme l'espérance de la distribution d_n, c'est-à-dire :

$$\mathrm{E}(t_n) = \sum_{k=0}^{len(d_n)} k d_n(k) .$$

4.5 Évolution de la technique symbolique

Dans cette section, nous allons présenter l'approche stochastique du point de vu de la méthode symbolique. Dans un premier temps, nous montrerons l'influence du modèle de temps "stochastique" sur la construction de la fonction $t_p^{mx,r}$. Ensuite une deuxième partie montrera comment obtenir une plus grande précision au niveau du calcul des régions de relâchement.

4.5.1 Influence de l'approche stochastique

Dans le partie 3.3 nous avons présenté une méthode permettant de respecter exactement le comportement du contrôleur non relâché tout en ne faisant pas appel à celui-ci à chaque point de contrôle. Nous savons que le modèle de temps "stochastique" influence le contrôleur sans relâchement. En effet il lui permet de rendre ses estimations plus proche de la réalité, mais en contrepartie il permet de violer quelques contraintes de temps. A partir de là, nous pouvons nous poser les mêmes questions pour l'approche symbolique. C'est-à-dire quels sont les apports du modèle de temps "stochastique" sur le relâchement du contrôleur. Et à l'inverse à quel comportement doit on s'attendre si les durées d'exécution pire-cas sont dépassées par les durées d'exécution réelles.

Comme pour le contrôleur initial, l'influence du modèle stochastique sur la méthode symbolique réside dans le fait d'avoir des durées d'exécution pire-cas plus proches du comportement. Ces durées d'exécution vont directement impacter sur les constructions des zones de relâchement.

Reprenons la définition des zones de relâchement précédemment décrite dans le chapitre 3.3 en utilisant la politique de gestion de qualité stochastique :

$$t_i \in]t_\tau^{mx}(s_{i+r-1}, q+1), t_\tau^{mx,r}(s_i, q)] \qquad pour\ q < q_{max} \tag{4.4}$$

$$t_i \in]-\infty, t_\tau^{mx,r}(s_i, q)] \qquad pour\ q = q_{max} \tag{4.5}$$

avec $t_\tau^{mx,r}(s_i, q) = min_{i \leq j \leq i+r-1} t_\tau^{mx}(s_j, q) - C_\tau^{wc}(a_{i+1} \ldots a_j, q)$. La fonction $t_\tau^{mx,r}$ correspond à la borne supérieure de la zone de relâchement, c'est-à-dire au comportement pire-cas.Dans sa définition, nous remarquons l'influence des durées d'exécution pire-cas. Si les durées d'exécution sont plus proches du comportement de l'application, il est évident que la zone de relâchement va s'affiner et va pouvoir réellement correspondre à des estimations plus proches que pour le modèle de temps précédent.

Cependant tout comme pour le contrôleur initial, dans lequel nous laissons la possibilité de violer quelques échéances, nous avons pour cette approche symbolique des états où le comportement ne sera pas forcément idéal. En effet le dépassement des durées d'exécution pire-cas va aussi permettre de mal évaluer le fait de sortir des zones de relâchement. Ainsi l'utilisation de régions de relâchement "stochastiques" va permettre d'avoir un comportement différent du contrôleur initial mais avec une probabilité fixée par le seuil de tolérance τ.

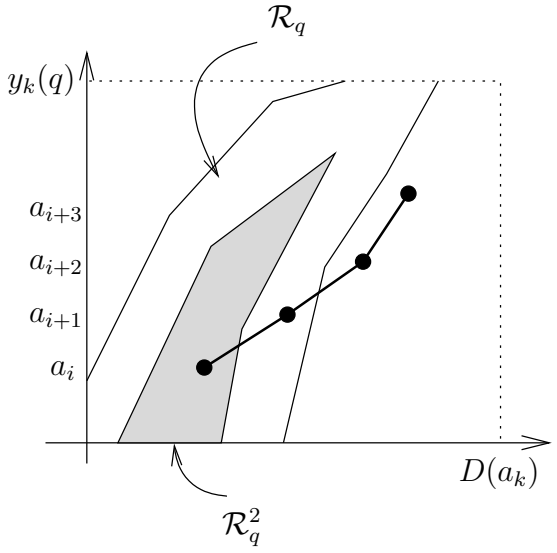


Figure 4.10 — Exemple d'exécution de l'application relâchée avec un modèle de temps stochastique

Exemple 4.10: *La figure 4.10 nous montre un exemple d'exécution de l'application contrôlée*

avec la technique de relâchement. Nous avons calculé une zone $\mathcal{R}_q^2$ correspondant à 2 pas de relâchement pour la qualité q. Pour l'action a_i, le contrôleur observe qu'il peut relâcher le contrôleur pour les deux prochains points de contrôle. Il exécute a_i, a_{i+1} puis a_{i+2}. A ce moment là il observe son état et il s'aperçoit qu'il est sorti de la région de qualité $\mathcal{R}_q$. Ceci est un comportement qu'il n'avait pas prévu. En effet, lors des actions a_i, a_{i+1} et a_{i+2}, le contrôleur avait prévu l'utilisation obligatoire de la qualité q. Or sur la figure nous voyons qu'il aurait été préférable d'utiliser la qualité $q-1$ pour l'action a_{i+2}. Cette mauvaise prédiction est due au fait que lors d'au moins une de ces trois actions, la durée d'exécution a été supérieure à la durée d'exécution pire-cas prévue.

Cet exemple nous montre bien que la technique de contrôle symbolique peut être imprécise si l'on utilise l'approche stochastique. Mais ce type d'exécution reste quand même très rare et, tout comme le phénomène de violation des échéances, est maîtrisé par l'utilisateur.

4.5.2 Durée d'exécution meilleur-cas

L'approche stochastique permet de générer simplement un modèle de temps basé sur les durées d'exécution pire-cas et moyennes. Nous savons que les zones de relâchement prennent en compte le comportement des prochaines actions pour des durées d'exécution pire-cas, mais aussi pour un meilleur-cas que nous avions préalablement défini à 0 unité de temps (cf. section 3.3). Cela simplifiait le problème étant donné que nous n'avions pas défini dans le modèle de temps précédent cette borne.

A partir du modèle de temps stochastique, il est assez simple de définir une durée d'exécution meilleur-cas pour chaque action et chaque qualité. En effet nous pouvons utiliser la même méthode que dans la définition de la durée d'exécution pire-cas (cf.section 3.3). Nous définissons les durées d'exécution meilleur-cas, C_τ^{bc}, par :

$$\forall i \in |A| \quad P[C(a_i, q) < C_\tau^{bc}(a_i, q)] = \tau$$

Le seuil de tolérance τ reste le même que pour les durées d'exécution pire-cas.

Il est maintenant possible de reconsidérer la construction des zones de relâchement en sachant que les durées d'exécution sont comprises entre C_τ^{bc} et C_τ^{wc}. Donc nous pouvons donner les bornes pour t_j :

$$t_i + C^{wc}(a_{i+1} \dots a_j, q) \geq t_j \geq t_i + C^{bc}(a_{i+1} \dots a_j, q)$$

Définition 4.19:
Soit $\widehat{t_p^{mx,r}} : A* \times \Theta \rightarrow \mathbb{R}^+$ la borne inférieure de la zone de relâchement. Nous la définissons par :

$$\widehat{t_p^{mx,r}}(a_{i+1} \dots a_n, s_i, q) = min_{i \leq j \leq i+r-1} t_p^{mx}(a_{i+1} \dots a_n, s_j, q) - C^{bc}(a_{i+1} \dots a_j, q)$$

Cette fonction permet de décrire le comportement meilleur-cas de l'application. Cela conduit à la proposition suivante :

Proposition 4.20:
Pour un niveau de qualité donné, un entier $r \geq 1$ et un état du système (s_i, q_i), $(s_i, q_i) \in R_q^r$ si et seulement si :

$$t_i \in]\widehat{t_\tau^{mx,r}}(s_i, q), t_\tau^{mx,r}(s_i, q)]$$

Cette nouvelle construction des zones de relâchement va permettre une plus grande flexibilité dans les choix du contrôleur. En effet nous ne considérerons pas des cas extrêmes se produisant presque jamais. Et ainsi nous aurons une meilleure connaissance du comportement de l'application. Ce qui permettra de faire moins d'appels au contrôleur, tout en sachant que l'application ne changera pas de région de qualité.

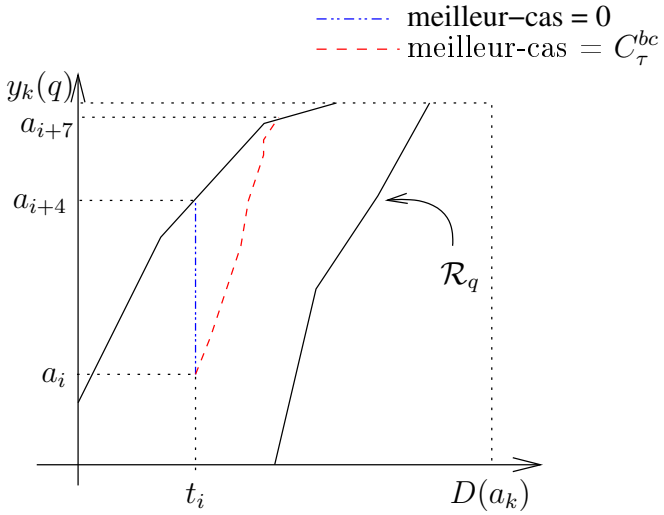


Figure 4.11 — Comparaison de l'estimation pour les temps d'exécution meilleur-cas

Exemple 4.11: *La figure 4.11 nous montre une comparaison au niveau des estimations du nombre de pas de relâchement que l'on peut faire dans les comportements meilleur-cas. Un tracé correspond à la construction que l'on a pour une durée meilleur-cas évaluée à 0. Ce modèle de temps accepte seulement 4 pas de relâchement. Par contre en prenant une durée d'exécution calculée à partir de l'approche stochastique, nous pouvons voir que le contrôleur peut relâcher ses appels pour les 7 pas suivants.*

L'exemple précédent suffit pour montrer l'intérêt d'avoir une borne meilleur-cas pour le relâchement du contrôleur.

4.6 Conclusion

Dans ce chapitre, nous avons présenté une politique de gestion de qualité stochastique. C'est une approche pertinente dans le cadre des applications multimédia qui font partie de monde du temps-réel mou. En effet cette politique laisse la possibilité de manquer des échéances mais en bornant le nombre de "ratés" par un seuil de tolérance. Cette technique est basée sur une politique de gestion de qualité générée à partir de distributions de probabilités. Nous avons proposé de calculer les durées d'exécution pire-cas à partir de ces dernières et du seuil de tolérance. Le modèle de temps stochastique permet à l'application d'avoir un comportement plus proche de ce que recherche l'utilisateur. En effet nous lui donnons la possibilité de gérer le compromis qu'il veut faire entre avoir un comportement sûr ou avoir une meilleur utilisation des ressources.

A partir de cette politique nous avons fourni des outils au programmeur lui permettant de connaître à priori les performances de son application par rapport au seuil de tolérance fixé. Nous donnons notamment des résultats par rapport aux taux de violation des échéances, mais aussi un résultat par rapport au budget de temps qui devrait être utilisé. A partir de ces résultats, le programmeur peut donc adapter le seuil de tolérance afin d'obtenir le comportement qu'il souhaite pour l'application.

En résumé cette méthode permet de réaliser une sorte de contrôle des applications temps-réel mou mais maîtrisé.

Pour finir nous avons montré que l'approche stochastique pouvait s'appliquer aussi à la technique de contrôle symbolique. Cela permet comme nous l'avons vu, de raffiner les zones de relâchement, non seulement avec des pire-cas plus réalistes, mais aussi avec des durées d'exécution meilleur-cas.

Dans le chapitre 5, nous allons montrer, sur des expérimentations, les apports de cette méthode.

CHAPITRE

Implémentation et Expérimentations

5

Nous présentons dans ce chapitre des résultats expérimentaux réalisés à partir des améliorations théoriques proposées dans les différents chapitres précédents. Ce travail de thèse a été motivé par une application cible réelle, une application d'encodage vidéo. Nous présenterons dans une première partie cette application ainsi que la plateforme d'exécution utilisée. Dans une deuxième partie nous présenterons l'intérêt du contrôle d'application en comparant les résultats donnés par une application multimédia contrôlée et une utilisation industrielle de celle-ci. Ensuite, nous présenterons les résultats expérimentaux des contributions de cette thèse. Nous commencerons par montrer les différentes politiques ainsi que leurs différents comportements sur un diagramme de vitesses. Nous donnerons aussi les résultats obtenus pour la technique de contrôle symbolique. Pour finir nous présenterons les résultats expérimentaux de l'approche stochastique.

5.1 Environnement d'expérimentation

5.1.1 La plateforme

Pour ces résultats expérimentaux, la plateforme cible est une *board* STm8010 fournie par STMicroelectronics. Cette plateforme est composée de trois processeurs ST231 cadencés à 400MHz. Etant donné que nous nous focalisons uniquement sur des plateformes mono-processeur, nous utilisons un seul processeur ST231.

Cette plateforme est dite "machine nue". En effet, elle n'a pas de système d'exploitation, et nous permet donc d'exécuter uniquement l'application contrôlée que nous souhaitons sans interférence avec d'autres applications. Cependant elle possède une bibliothèque de fonctions permettant d'exécuter tout le code possible mais aussi de pouvoir récupérer les résultats et des données tels qu'un compteur de cycles.

Un registre comptant le nombre de cycles du processeur va nous donner la possibilité d'obtenir les durées d'exécution réelles avec un surcoût minimum.

5.1.2 L'encodeur vidéo MPEG4

5.1.2.1 Présentation de l'application

L'application sur laquelle on se base est un encodeur vidéo présent dans un visiophone. La visiophonie est une technique associant téléphonie et télévision et permettant aux utilisateurs de se voir lors d'une conversation téléphonique. Plusieurs protocoles sont utiles au bon déroulement de cette technique. Dans ce qui suit nous parlerons uniquement de la partie encodage vidéo temps-réel. Cet encodeur vidéo est un *system-on-chip* capable de réaliser l'encodage en temps réel d'un flux vidéo, dans un contexte embarqué tel que la téléphonie mobile.

L'architecture de l'encodeur vidéo est décrite par la figure 5.1. Le principe est simple, une caméra capture une séquence vidéo, la transmet, pour finir par afficher des images à l'écran. A partir d'une image capturée, l'encodeur vidéo produit un *bitstream* correspondant. Ce dernier est transmis à un décodeur vidéo qui le décode et affiche ce *bitstream* décodé à l'écran. Cette architecture utilise des *buffers* d'entrée et de sortie de la même taille K. Cela permet d'éviter autant que possible les *frameskipping*, ce phénomène survient quand le *buffer* d'entrée est plein en sautant l'encodage de la frame suivante, nous l'expliquerons dans la suite (p.115).

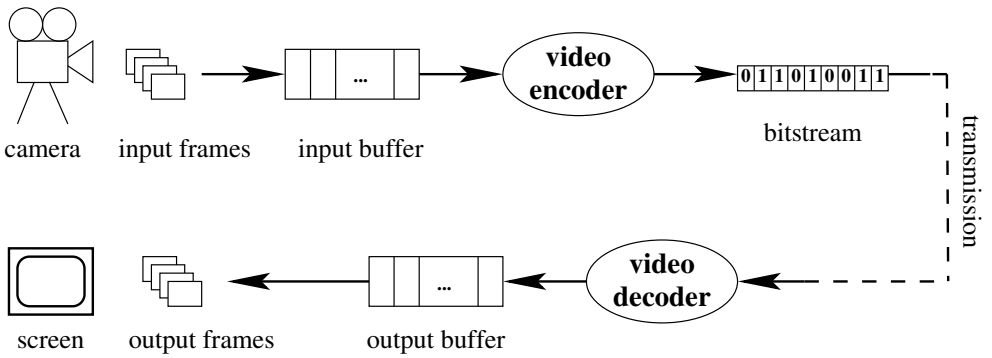


Figure 5.1 — Video encoder architecture.

5.1.2.2 Modélisation de l'application

L'encodage vidéo MPEG4 est un algorithme de compression vidéo. L'objectif de cet encodage est de générer un flux de bits à partir d'un flux vidéo, ou flux de *frames*, issu par exemple d'une caméra vidéo.

Nous donnons ici le modèle de l'encodeur que nous avons utilisé pour les résultats expérimentaux présentés dans ce chapitre. La figure 5.3 nous montre une représentation de l'application sous forme de graphe. Celui-ci est composé de plusieurs actions qui traitent une image.

La première étape de l'algorithme d'encodage, **GrabPicture**, est un pré-traitement

sur la *frame* courante qui permet, en particulier, de déterminer si c'est une *frame* INTER ou INTRA. En effet, nous pouvons distinguer deux manières principales d'encoder une *frame*. Le codage de type INTER permet de faire référence à une ou plusieurs *frames* autres que la *frame* courante. Les *frames* auxquelles l'encodage fait référence peuvent tout aussi bien être situées temporellement avant ou après la *frame* courante. Le codage de type INTRA sera quant à lui utilisé pour encoder les *frames* clés c'est-à-dire qui correspondent à des changements de plan dans la vidéo. Dans ce type d'encodage, on ne fait pas référence à une autre *frame* que celle traitée.

Pendant cette première étape de l'algorithme, d'autres traitements sont effectués, comme par exemple la transformation d'un codage des couleurs de type RVB, issu de la caméra vidéo, en un codage de type YUV.

La suite de l'algorithme est constituée par des traitements qui ne se situent pas au niveau de l'intégralité *frame* traitée, mais au niveau *macro-block*. Un *macro-bloc* est une portion d'une *frame*, un carré de 16×16 pixels dans l'algorithme considéré ici (cf. figure 5.2). Tout d'abord, le *macro-bloc* est récupéré dans la *frame* à l'aide de **GrabMacroBlock**.

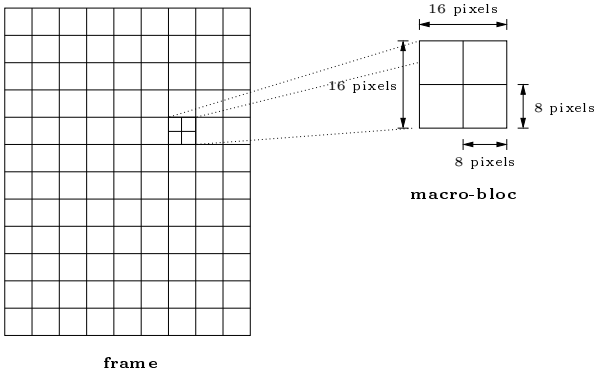


Figure 5.2 — Différents niveaux d'une image

Le traitement suivant est appelé estimation de compensation du mouvement (**MotionEstimation**). Cette action permet de trouver les corrélations spatiales qui existent entre les frames. Pour la réaliser, on se base sur des standards de compression comme H264 [42], [43] ou MPEG4 [44], [45]. Ces standards vont permettre d'augmenter le taux de compression et à partir d'algorithme de complexité croissante, d'obtenir une meilleure qualité d'image. C'est donc à cette étape que l'algorithme d'encodage vidéo fluctue en fonction de la qualité. **MotionEstiomation** va regarder les corrélations qui existent entre les *frames*, qu'elles soient consécutives, distantes ou entre une et plusieurs *frames*. Cette étape ne sera pas faite si la *frame* traitée est de type INTRA. Cependant si elle est de type INTER, l'algorithme détermine si le *macro-bloc* courant est de type INTRA ou de type INTER. Dans ce dernier cas, l'estimation de mouvement calcule aussi son vecteur de mouvement. Celui-ci peut être considéré comme étant le vecteur entre la position du macro-block actuel et celle du macro-block lui ressemblant le plus dans la frame précédente. Le critère de ressemblance le plus utilisé est la somme des valeurs absolues

des différences (ou *SAD*), calculé sur la luminance uniquement. Plutôt que d'utiliser une recherche exhaustive du macro-bloc le plus ressemblant, qui est évidemment très coûteuse même pour une fenêtre de recherche de 16×16 pixels, des heuristiques sont utilisées. Une fois le vecteur de mouvement calculé, la compensation de mouvements consiste à calculer la différence entre le macro-bloc courant et le macro-bloc le plus ressemblant. Dans le cas d'un macro-bloc de type INTER, ce sont les coefficients issus de cette différence qui sont transmis aux étapes de calcul suivantes.

Le traitement suivant est la transformée de Fourier discrète à deux dimensions (**DCT**). Pour des raisons techniques, elle est implémentée pour des blocs d'images de 8×8 pixels. Le macro-bloc courant est donc découpé en quatre blocs de 8×8 (cf. figure 5.2), sur lesquels sont appliqués la transformée de Fourier discrète. A partir des coefficients correspondant au macro-bloc courant, nous obtenons autant de coefficients avec la transformée de Fourier. Il s'agit donc d'une autre représentation des coefficients de départ, qui correspond à un changement de base. Les coefficients obtenus ne correspondent plus au différents pixels qui composent le macro-bloc, mais aux différentes fréquences qui composent le macro-bloc.

Le traitement suivant, appelé *quantization* (**Quant**), est une opération qui consiste à faire diviser les coefficients issus de la transformée de Fourier, de façon à réduire leur variabilité. Les coefficients sont plus ou moins affectés en fonction de la fréquence à laquelle ils correspondent, et en fonction du taux de compression qui est visé.

Le reste des traitements sur le macro-bloc courant correspondent à la production du *bitstream*, et la reconstruction de l'image encodée. Concernant la production du *bitstream*, la prédiction des coefficients INTRA (**IntraPredict**) est utilisée dans le cas des *frames* ou macro-blocs de type INTRA afin d'exploiter les corrélations spatiales de l'image de façon à réduire la variabilité des coefficients. Ces corrélations spatiales correspondent à toutes les corrélations que l'on peut trouver dans une même *frame*.

L'étape finale consiste à appliquer une compression conservative (**Coding**), appelée parfois codage entropique, aux coefficients issus des étapes précédentes. Cela permet d'encoder les coefficients avec le minimum de bits, en tenant compte des redondances que l'on retrouve dans les coefficients. Le *bitstream* final contient également toutes les informations utiles pour décoder la *frame*, comme sa taille, son type, le type de chaque macro-bloc ou encore les vecteurs de mouvements pour les macro-blocs de type INTER.

La reconstruction de l'image se fait en appliquant les transformations inverses à la quantization (**IQUANT**), la transformée de Fourier discrète (**IDCT**) et l'estimation et compensation de mouvement(**Reconstruction**). Ces étapes permettent de construire une *frame* qui correspond à la *frame* courante, mais en tenant compte des dégradations liées à la quantization. Cette *frame* reconstruite est donc identique à celle qui sera décodée par un décodeur.

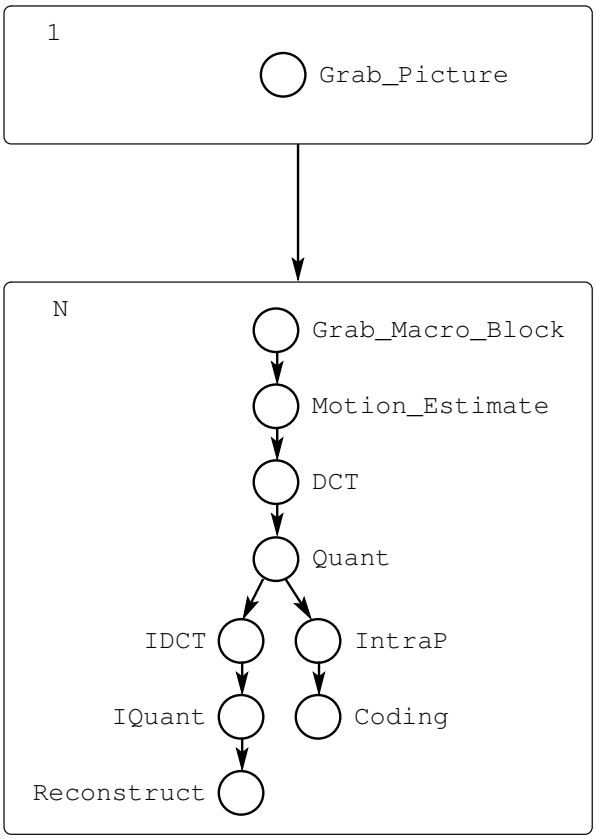


Figure 5.3 — Graphe de précédence pour l'encodeur vidéo

5.1.2.3 Durées d'exécution et ordonnancement

Maintenant que l'application cible a été présentée, à travers les différentes actions mais aussi ses dépendances (voir figure 5.3), il s'agit de donner le modèle de temps utilisé pour réaliser la première série d'expériences. Les durées d'exécution fournies seront utilisées pour développer les résultats théoriques décrits dans les chapitres 2 et 3. Les durées d'exécution moyen et pire-cas sont données par la figure 5.4. Les durées d'exécution de *MotionEstimation*, *DCT*,*Quant* et *Coding* dépendent du niveau de qualité. Les durées d'exécution de toutes les autres actions sont constantes, et sont données par la figure 5.4. Nous utilisons aussi une échéance D correspondant au budget de temps alloué à l'encodage d'une frame. La seule action dont la durée d'exécution a une fluctuation significative en fonction des niveaux de qualité est *MotionEstimation*. Pour réduire les nombres d'appels aux Gestionnaire de Qualité, nous contrôlons le niveau de qualité uniquement avant l'exécution de cette action. Ainsi le nombre d'appels au contrôleur par *frame* est égale à N.

Action	Moyen	pire-cas
Grab_Picture	11000	20000
Grab_Macro_Block	9000	20000
IntraP	8000	20000
IQuant	10000	15000
IDCT	8000	15000
Reconstruct	11000	20000

MotionEstimation		
Qualité	**Moyen**	**pire-cas**
0	3000	10000
1	12000	40000
2	20000	50000
3	30000	100000
4	40000	120000
5	50000	150000
6	70000	200000
7	90000	300000

Action	Moyen $q > 0$	pire-cas $q > 0$	Moyen $q = 0$	pire-cas $q = 0$
DCT	11000	15000	150	400
Quant	12000	20000	1500	4000
Coding	10000	25000	1500	3000

Figure 5.4 — Durées d'exécution moyen et pire-cas.

Dans les résultats suivants, nous prendrons en compte une politique d'ordonnancement statique. C'est-à-dire que l'ordonnancement ne sera pas réalisé dynamiquement mais calculé à priori. L'application sera représentée par l'ordonnancement statique suivant :

`Grab_Picture` $\{$`Grab_Macroblock MotionEstimation DCT Quant IQuant IDCT Intra_Prediction Coding Reconstruction`$\}^n$

5.2 L'outil

Nous avons développé un prototype d'outil de génération automatique de contrôleur (cf. figure 5.5). Dans ce qui suit nous présentons le fonctionnement général de l'outil. Cette génération se déroule en deux phases. La première permet de générer, à partir d'un système paramétré, un ordonnancement et des tables pour le calcul des politiques de gestion de qualité. La deuxième partie est la génération de l'application contrôlée à partir de ces données mais aussi de bibliothèques de fonctions.

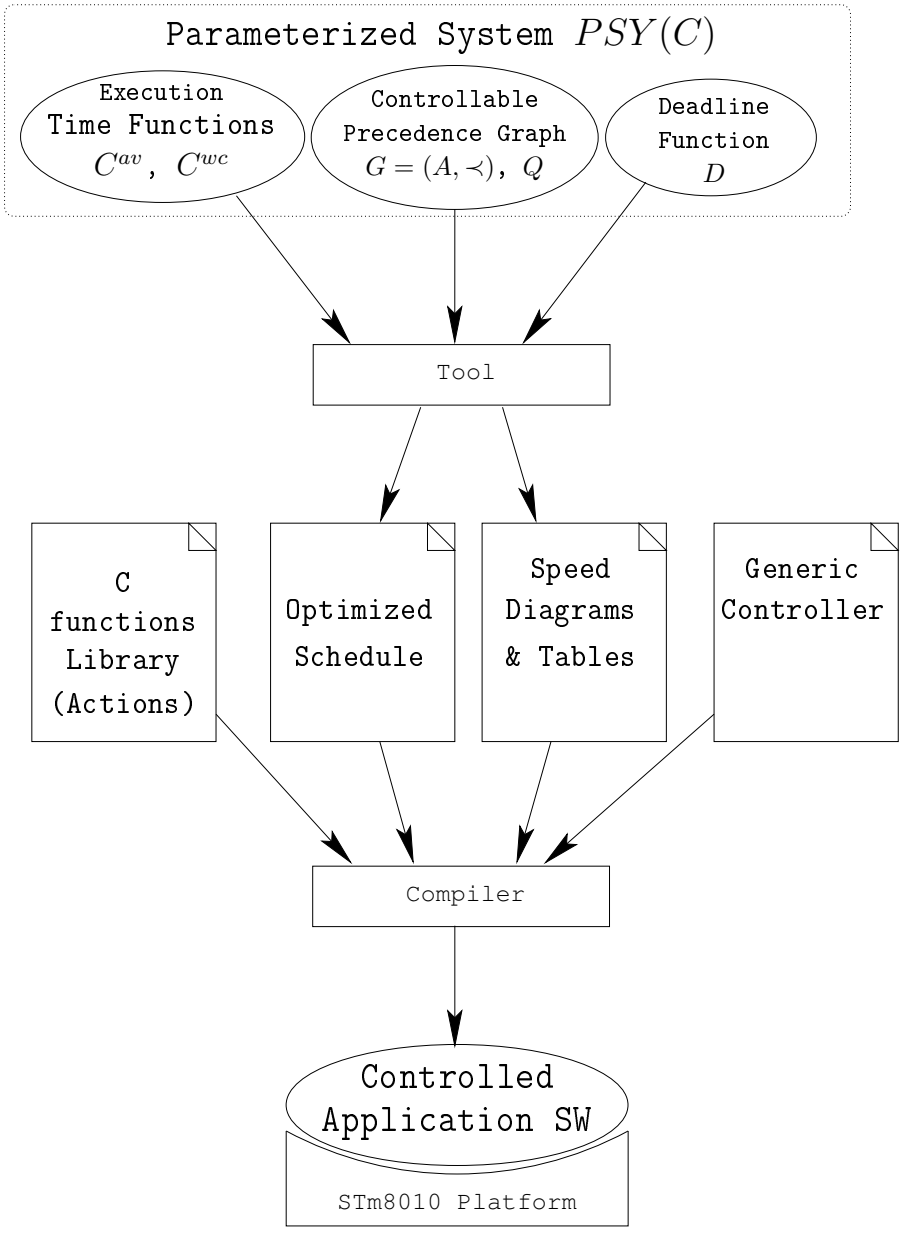


Figure 5.5 — Outils de génération du contrôleur d'application

Modèle d'entrée de l'application

Le modèle de l'application qui se trouve en entrée du flot de génération de contrôle est décrit par un système paramétré incertain $SPI(C) = (G, Q, C^{wc}, D, C)$ et une fonction d'une durée d'exécution moyen C^{av}. Les entrées du flot de génération du contrôleur sont donc représentées par :

- un graphe de précédence $G = (A, \prec)$ qui modélise les actions (fonctions C) et les dépendances entre les actions,
- un ensemble de niveaux de qualité Q correspondant à chaque action contrôlable,
- les durées d'exécution pire-cas C^{wc} et les durées d'exécution moyennes C^{av},
- une échéance critique D de l'application.

Génération de l'ordonnancement et des tables précalculées

À partir de ce modèle d'application, il s'agit maintenant de calculer les informations utiles à la génération de l'application contrôlée finale. Le premier outil de l'application génère donc :

- le diagramme des vitesses permettant l'utilisation de la technique de contrôle symbolique. Celui-ci sera donc non seulement composé des régions de qualité R_q mais aussi des régions de relâchement R_q^R. Le nombre de régions de relâchement sera fixé par le programmeur,
- des tables précalculées permettant d'optimiser le calcul des politiques de contrôle énumératives t_p^{sf}, t_p^{sp}, t_p^{mx},
- un ordonnancement statique de l'application. Celui-ci correspond à un ordonnancement optimisé basé sur les techniques d'ordonnancement E.D.F. (cf. section 5.1.2.3).

Génération de l'application contrôlée

Il s'agit pour finir de générer une application contrôlée. Pour cela il est nécessaire d'avoir les données précédemment décrites, notamment le diagramme des vitesses, les tables et l'ordonnancement statique. Mais il faut aussi avoir une bibliothèque des fonctions écrites en C ; celle-ci fournie une implémentation des actions du système paramétré incertain. Ces fonctions décrivent l'effet de l'exécution des actions. Dans l'implémentation de l'outil, le lien entre une action et sa fonction correspondante peut être assuré par une table fournie en entrée. Le troisième élément nécessaire à la génération de l'application contrôlée correspond au Contrôleur, qui sera principalement constitué d'un Gestionnaire de Qualité. à partir de là, un compilateur génère une application instrumentée par un contrôleur.

5.3 Intérêt du contrôle d'application

5.3.1 Comparaison entre une exécution contrôlée et une exécution à qualité constante

Dans cette première série d'expérimentations, nous allons montrer l'intérêt des techniques de contrôle pour les applications multimédia. Pour cela nous allons comparer l'application contrôlée par rapport à l'application dans laquelle nous avons fixé un niveau de qualité constant. Ceci correspond à la pratique industrielle courante, dans laquelle le seul contrôle qui est fait par rapport aux contraintes de temps est le *frameskip*. C'est-à-dire que si les échéances ont été dépassées alors la *frame* suivante ne sera pas encodée. Le niveau de qualité à été choisi en fonction de l'échéance globale des actions, de façon à obtenir un compromis entre le taux de *frameskip* et la qualité de l'encodage.

Nous mesurons le PSNR (Peak Signal to Noise Ratio) entre les frames d'entrée et les frames de sortie. PSNR caractérise le niveau de qualité d'une *frame* et il est utilisé pour mesurer l'effet du processus d'encodage sur la qualité de la vidéo. Nous avons comparé la performance de l'application contrôlée par la politique mixte et le même encodeur pour un niveau de qualité constant. Nous avons donc représenté l'utilisation du budget de temps qui est le ratio entre le temps d'encodage d'une *frame* et l'échéance D. Nous considérons une étude de cas de 50 *frames*, constituée de trois séquences produites par une caméra toutes les $D = 100\ ms$ ce qui équivaut à un taux de 10 $frame/s$.

La taille du *buffer* de K permet une latence maximale de $D \cdot K$. Le budget de temps alloué à l'encodeur pour le traitement d'une *frame* dépend de l'occupation de ce *buffer* et correspond en moyenne à D. Comme notre méthode garantit la sûreté, nous avons pris $K = 1$ pour un encodeur contrôlé qui se doit de ne manquer aucune échéance.

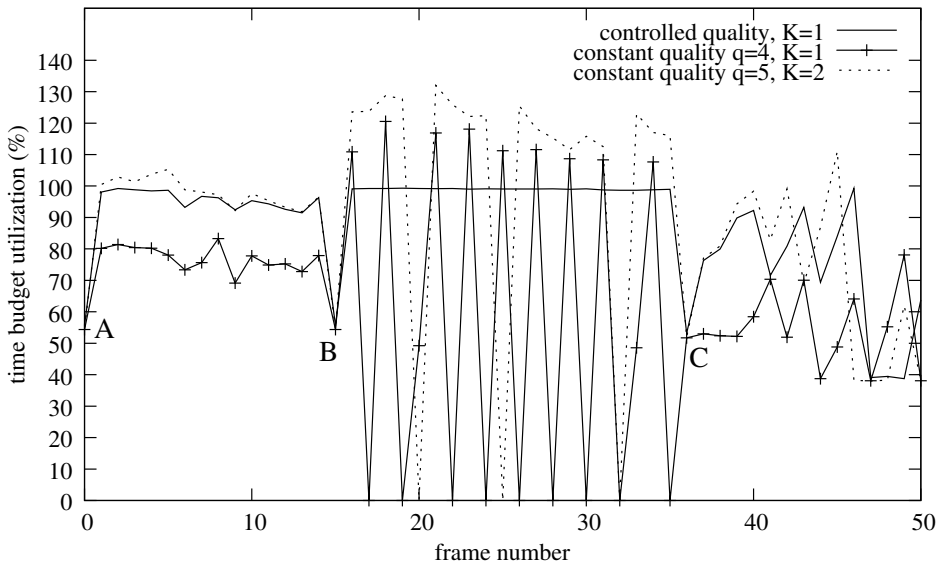


Figure 5.6 — Utilisation du budget de temps

L'utilisation du budget de temps est représentée par la figure 5.6, pour un niveau de qualité contrôlé par la politique mixte et pour un niveau de qualité constant $q = 4, K = 1$, et $q = 5, K = 2$.

Sur ce graphe nous pouvons voir la présence de deux types de pics :
- trois pics correspondant aux changements de séquences vidéo, c'est-à-dire des frames de type INTRA. Ces pics ont lieu aux frames 0, 15, et 35 (respectivement aux points A, B et C du graphique). Nous pouvons observer que ces pics ont aussi bien lieu pour l'encodeur contrôlé que pour l'encodeur sans contrôle ;
- les pics correspondant aux *frameskipping*. Ce phénomène est du au fait que le *buffer* est plein et apparaît seulement pour les encodeurs sans contrôle.

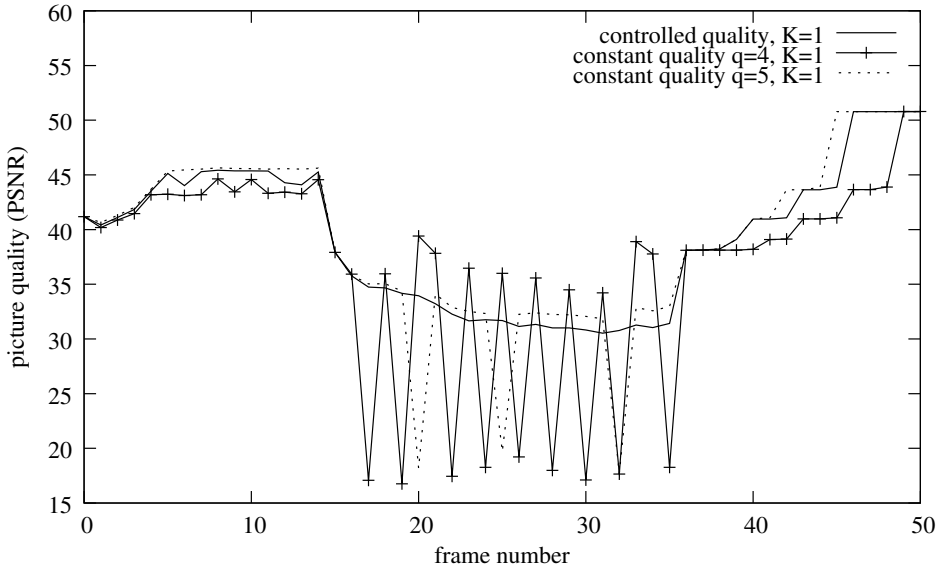


Figure 5.7 — PSNR entre l'entrée et la sortie

Le PSNR est donné par le graphique 5.7. Nous pouvons observer ici aussi deux types de pics qui ont lieu pour les mêmes raisons que précédemment. Quand une *frame* est "skippée", la *frame* qui la précède directement est renvoyée par le décodeur et ainsi la comparaison avec la *frame* d'entrée donne une valeur du PSNR plus petite. Par exemple dans notre cas nous observons des PSNR inférieurs à 25. Le PSNR est plus élevé pour l'encodeur contrôlé que pour un encodage à qualité constante $q = 4$, sauf pour les régions où les *frames* sont "skippées". En effet, les bits correspondant aux frames "skippées" sont utilisés pour améliorer la qualité. Mais bien que le PSNR soit plus élevé pour ces régions, la qualité de la vidéo est affectée étant donné que le taux de *frame* par seconde est divisé par deux. L'utilisation d'une taille de *buffer* plus grande, $K = 2$, permet l'activation d'une qualité supérieure mais le phénomène de frameskipping reste.

Ces résultats expérimentaux montrent que pour un niveau de qualité constant des fluctuations de la charge de travail peuvent diminuer la qualité de la vidéo si la taille du *buffer* n'est pas suffisante. Pour un encodeur contrôlé, il n'y a pas de *frameskipping*.

De plus, l'utilisation d'une taille de *buffer* plus grande n'éradique pas le problème de *frameskipping* mais cela implique des coûts additionnels.

Donc cette première série d'expériences montre que pour un contrôle de qualité, nous obtenons une meilleure qualité de vidéo, et que le phénomène de *frameskipping* est totalement évité.

5.3.2 Comparaison de différentes politiques de gestion de qualité

Dans cette partie, nous allons montrer l'importance du choix de la politique de gestion de qualité. Nous avons comparé les politiques de gestion de qualité sûre, simple et mixte. Nous donnons des résultats pour un ordonnancement statique. La figure 5.8 donne le diagramme des vitesses pour une *frame* particulière de la séquence d'entrée.

Dans le chapitre 2, nous avons vu que les politiques de gestion de qualité sûre, simple et mixte permettent toutes de respecter les échéances, ainsi que d'obtenir une utilisation élevée du budget de temps. Nous voyons en effet sur la figure 5.8 que l'utilisation du budget de temps est similaire pour les trois politiques de gestion de qualité.

Concernant la régularité des niveaux de qualité choisis par le contrôleur, la différence est plus importante. Le diagramme des vitesses est une représentation intuitive pour étudier la régularité des niveaux de qualité. En effet, nous avons vu dans la section 3.1 que les niveaux de qualité correspondent à des vitesses constantes sur le diagramme, lorsqu'on utilise une politique d'ordonnancement statique et que la fonction d'échéance est constante. Ainsi, si les durées d'exécution réelles sont proches des durées d'exécution moyennes, une affectation de qualité régulière devrait correspondre à un tracé presque rectiligne dans le diagramme des vitesses. C'est d'ailleurs ce que nous constatons dans les expériences que nous avons menées, et en particulier dans celle qui est représentée sur le diagramme de la figure 5.8.

En effet, les niveaux de qualité obtenus en utilisant la politique mixte sont presque constants, ce qui se traduit par un tracé presque rectiligne dans le diagramme.

Pour les politiques de gestion de qualité sûre et simple, les discontinuités de vitesses sont observées aux points A et B. Pour la politique safe (resp. simple), la vitesse du système du point A au point B correspond à l'activation du niveau de qualité minimal. Cela réduit considérablement la qualité de la vidéo. Ces deux politiques sont donc trop optimistes au début de l'exécution, ce qui conduit à une réduction de qualité pour assurer le respect de l'échéance.

Les résultats expérimentaux confirment l'intérêt de la politique de contrôle mixte par rapport aux politiques de contrôle simple et sûre. Si les deux dernières assurent le respect des échéances et une bonne utilisation de la plateforme pour une politique d'ordonnancement donnée, elle ne conduit pas à une bonne régularité des niveaux de qualité choisis par le contrôleur. En effet, alors que les politiques simple et sûre conduisent à l'activation du niveau de qualité minimal, la politique de contrôle mixte conduit à une

affectation de qualité presque constante. Ceci améliore sensiblement la qualité d'image finale.

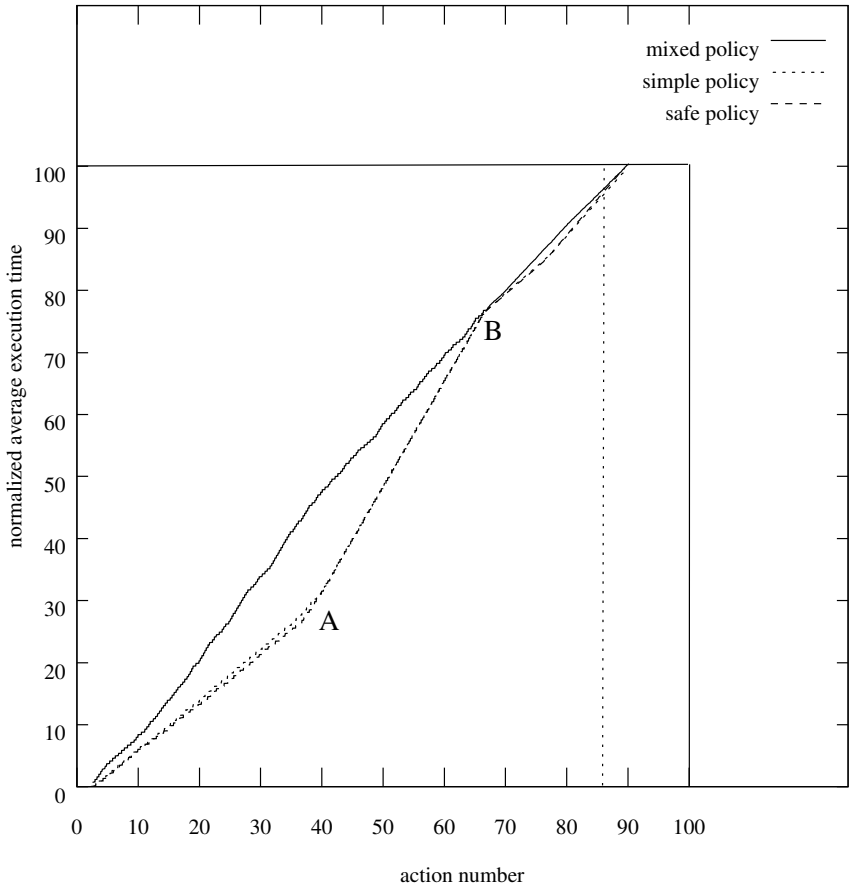


Figure 5.8 — Comparaison des différentes politiques de contrôle

5.4 Les apports de l'approche symbolique

Dans cette partie des résultats expérimentaux, nous allons attirer votre attention sur l'intérêt de l'utilisation de l'approche symbolique présentée dans la section 3.3. Pour cela nous allons comparer quatre techniques de contrôle :

Technique Énumérative

C'est une implémentation directe de la politique de gestion de qualité mixte décrite dans la section 2.2.3. Nous considérerons une version non optimisée qui réalise tous les calculs des fonctions de durées d'exécution et de la marge de sécurité à la volée.

Technique Énumérative Optimisée

C'est une implémentation de la politique de gestion de qualité mixte qui utilise des tables précalculées pour accélérer son exécution.

Technique Symbolique

Nous utiliserons pour cette implémentation la description des régions de qualité $\mathcal{R}_q$ définies dans la section 3.2. Ces régions sont précalculées, et seront utilisées par le Gestionnaire de Qualité pour choisir en ligne la qualité de l'action suivante.

Technique Symbolique Relâchée

Nous utiliserons, pour cette dernière implémentation, un outil qui calculera à priori les régions de relâchement du contrôleur $\mathcal{R}_q^r$ définies dans la section 3.3 pour $r \in \rho = \{1, 3, 6, 9, 16, 32\}$. Ces régions seront utilisées par le Gestionnaire de Qualité pour réduire la granularité de contrôle et calculer le niveau de qualité de l'action suivante.

Nous avons comparé les performances de l'exécution de l'implémentation énumérative et celle de l'implémentation symbolique pour une séquence d'entrée de 29 frames de 320×144 pixels (N = 180 *macroblocks*). Nous pouvons mesurer deux surcoûts, le premier concerne la mémoire, tandis que le second concerne les durées d'exécution.

Le surcoût d'allocation mémoire pour les techniques énumératives peut être considéré comme nul. En effet la taille des tables précalculées n'est pas suffisamment élevée pour être considérée comme étant un surcoût et de ce fait peut être négligée. Cependant il n'en est pas de même pour les approches symboliques, nous avons les surcoûts d'allocation mémoire suivants :

Gestionnaire de Qualité utilisant les régions de qualité

La proposition 3.7 nous montre que les régions de qualité sont construites pour l'ensemble des valeurs $t_p^{mx}(a_{i+1} \dots a_n, s_i, q)$ pour tous les niveaux de qualité q et pour chaque état s_i. Ainsi, étant donné que i est compris entre 0 et $N-1$, cet ensemble de valeurs peut être spécifié par $N \times |Q| = 1440$ entiers. Pour l'encodeur vidéo, nous avons ainsi mesuré un surcoût en termes d'allocation mémoire de 20 KB.

Gestionnaire de Qualité utilisant les régions de relâchement

La proposition 3.9 nous montre que les régions de relâchement sont caractérisées par l'ensemble des valeurs $t_p^{mx}(a_{i+1} \dots a_n, s_{i+r-1}, q+1)$ et $t_p^{mx,r}(a_{i+1} \dots a_n, s_i, q)$ pour tous les niveaux de qualité q, pour tous les indices $i \in \{1, \dots, N-1\}$ et pour tous les pas de relâchement $r \in \rho$. Ainsi cet ensemble de valeurs peut être spécifié par $2N \times |Q||\rho| = 17280$ entiers. Nous avons observé un surcoût en termes d'allocation mémoire égale à 350 KB.

La figure 5.9 nous montre les différents surcoûts en termes de durée d'exécution par *frame* du contrôleur. Ceux-ci sont mesurés avec une horloge qui compte le temps que l'application passe dans le Gestionnaire de Qualité. Il est exprimé en pourcentage du temps de l'encodage d'une *frame* au total. Nous pouvons observer que l'approche énumérative non optimisée atteint un temps moyen de 8 . 5 % qui est réduit à 4 % après optimisation. En revanche l'utilisation de la technique de contrôle symbolique nous donne un surcoût en temps d'exécution avoisinant les 1 . 2% de moyenne, pour atteindre moins de 1% de moyenne pour la technique de relâchement du contrôleur.

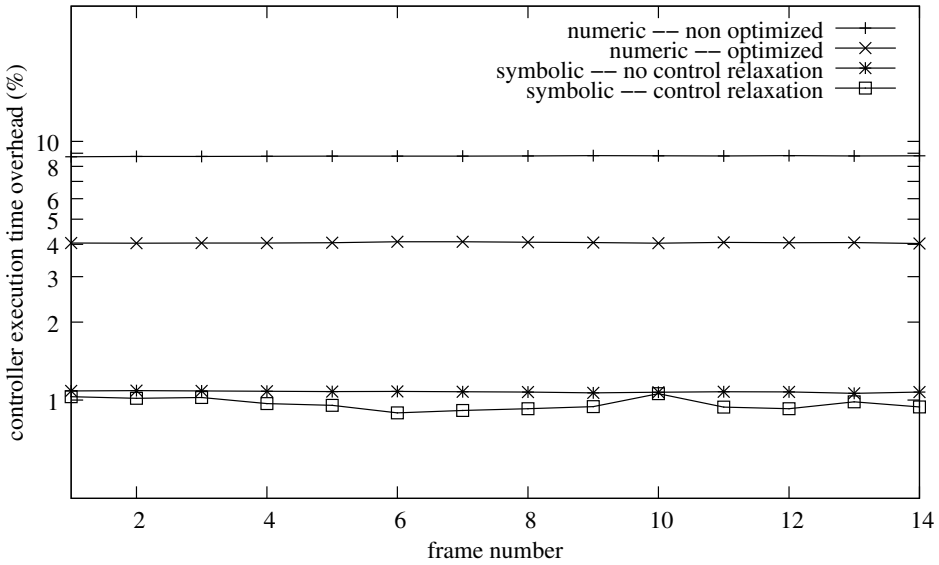


Figure 5.9 — Surcoût en termes de temps d'exécution à cause de la gestion de qualité

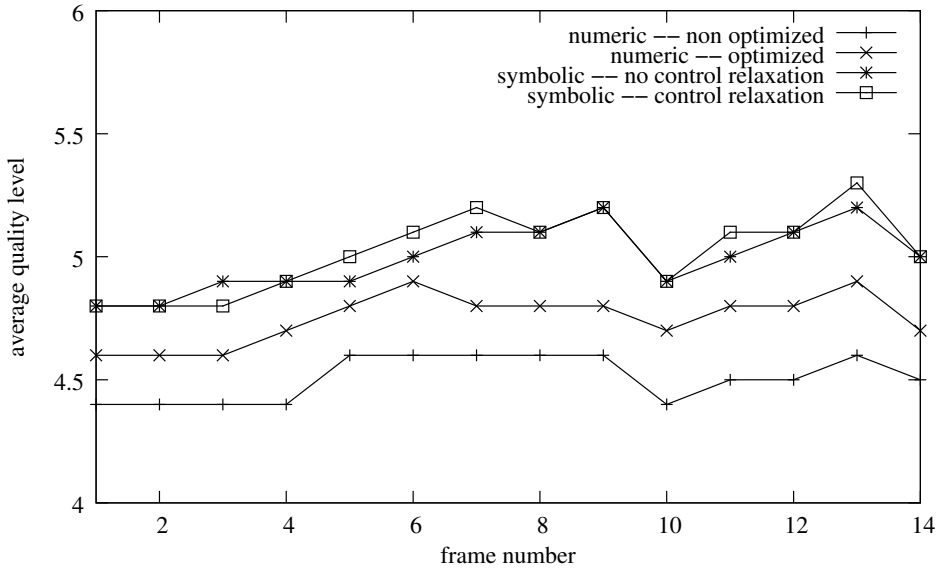


Figure 5.10 — Niveau de qualité moyen

Ainsi l'utilisation de la technique de contrôle symbolique est pertinente, et mène à des résultats intéressants pour l'encodeur vidéo. Il permet notamment une réduction importante du surcoût en termes de temps d'exécution. Ce résultat est non seulement intéressant pour montrer que l'aspect théorique du travail fonctionne, mais il est important de montrer que la réduction de ce surcoût va permettre de gagner du temps pour l'exécution de l'application. Ce gain de temps sera utilisé pour l'encodage des données, et va permettre d'augmenter le niveau de qualité. La figure 5.10 nous montre que le Gestionnaire de Qualité symbolique choisit des niveaux de qualité plus élevés que celui basé sur l'approche énumérative. Ceci va considérablement améliorer le résultat général de l'encodage vidéo.

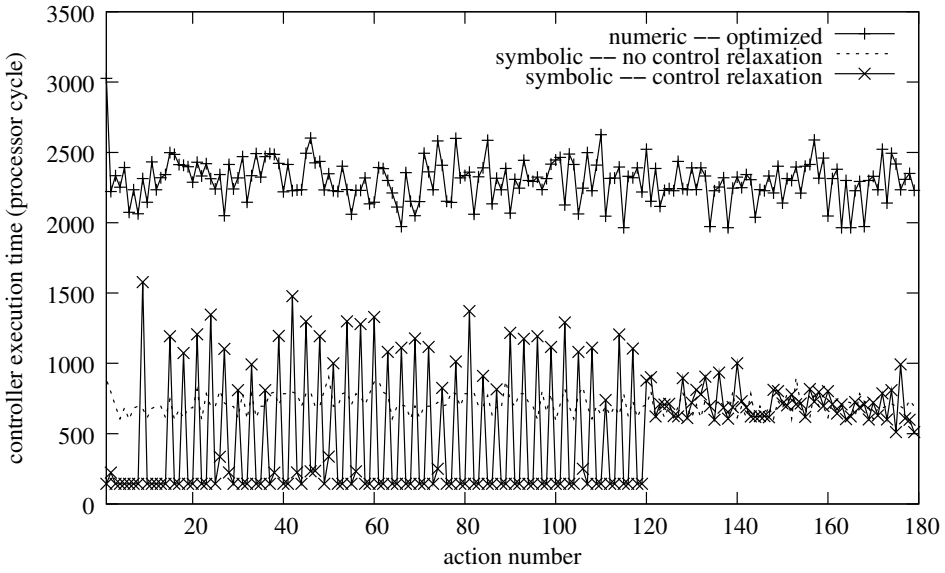


Figure 5.11 — Surcoût en termes de temps d'exécution

Une dernière expérimentation va nous être utile pour montrer le fonctionnement de la technique de relâchement du contrôleur. La figure 5.11 compare pour une séquence de 180 actions d'encodage d'une frame, les surcoûts en temps d'exécution avec et sans le relâchement du contrôleur. Nous pouvons observer que le nombre de pas de relâchement est dynamiquement adapté au cours de l'exécution de l'application. Il varie entre $r = 9$ pour l'action a_0, $r = 6$ pour l'action a_9, $r = 3$ pour les actions de a_{15} à a_{120} et $r = 1$ pour le reste des actions.

5.5 Résultats sur l'approche stochastique

5.5.1 L'outil d'analyse de performances

Pour ces résultats expérimentaux nous avons défini un autre outil (cf. figure 5.12). Il possède la même architecture que le précédent mais il est basé sur un système paramétré

incertain stochastique. Il permet de réaliser la partie analyse de performances mais en introduisant aussi le seuil de tolérance dont nous avons parlé dans le chapitre 4.1. Cet outil peut aussi générer un modèle de temps pouvant être utilisé par le précédent.

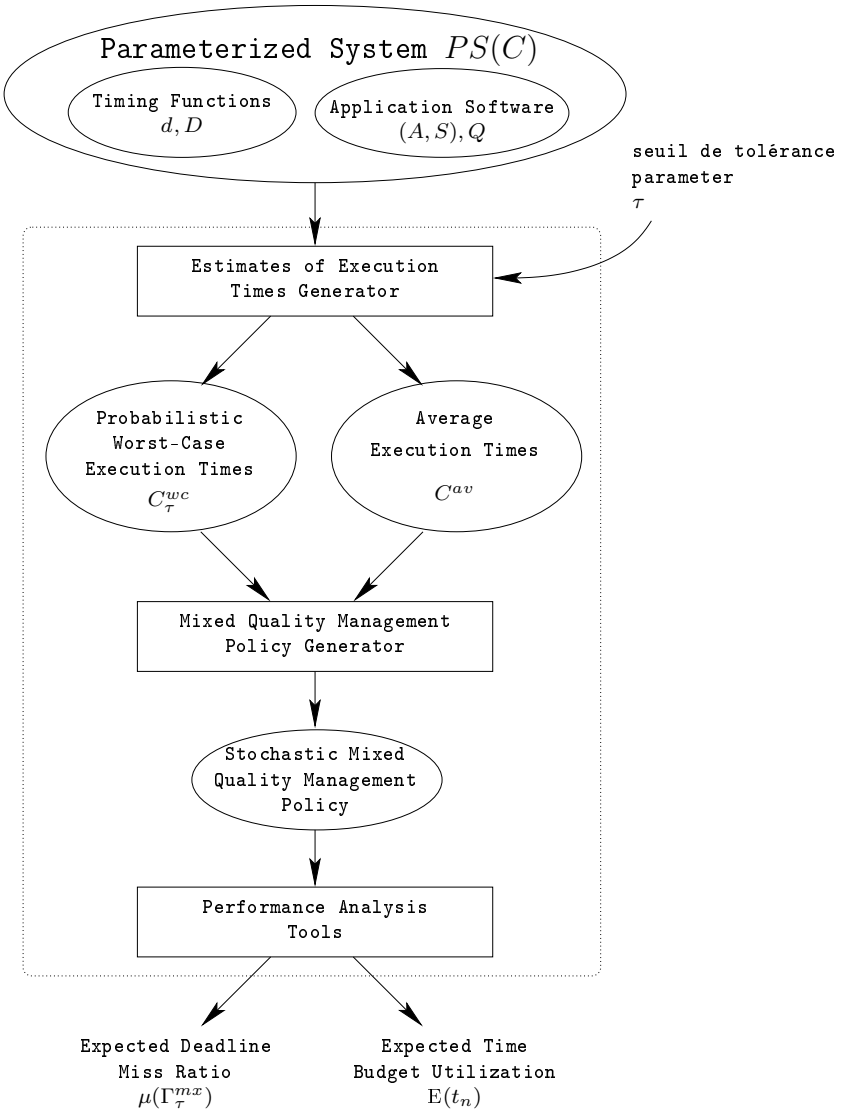


Figure 5.12 — Outils de calcul des performance

L'outil d'analyse de performances prend en entrée une application logicielle (A, S) modélisant les actions et l'ordre d'exécution de ces actions, un ensemble de niveaux de qualité Q, et une échéance D. Mais on lui fournit aussi deux autres données :

- des fonctions de distribution de probabilité discrètes $\{d_{a_i}^q\}_{a_i \in A, q \in Q}$ représentant les variations des durées d'exécution de chaque action,
- un seuil de tolérance τ.

Les fonctions de distribution $d_{a_i}^q$ ont été calculées en utilisant des techniques basées sur l'instrumentation du code. La figure 5.13 nous montre les distributions des actions

`Grab_Macroblock` et `Reconstruction`. Ces deux actions sont dites non contrôlables, c'est-à-dire que le niveau de qualité choisi n'influe pas sur les durées d'exécution. Nous pouvons donc utiliser une seule et même distribution pour ces actions.

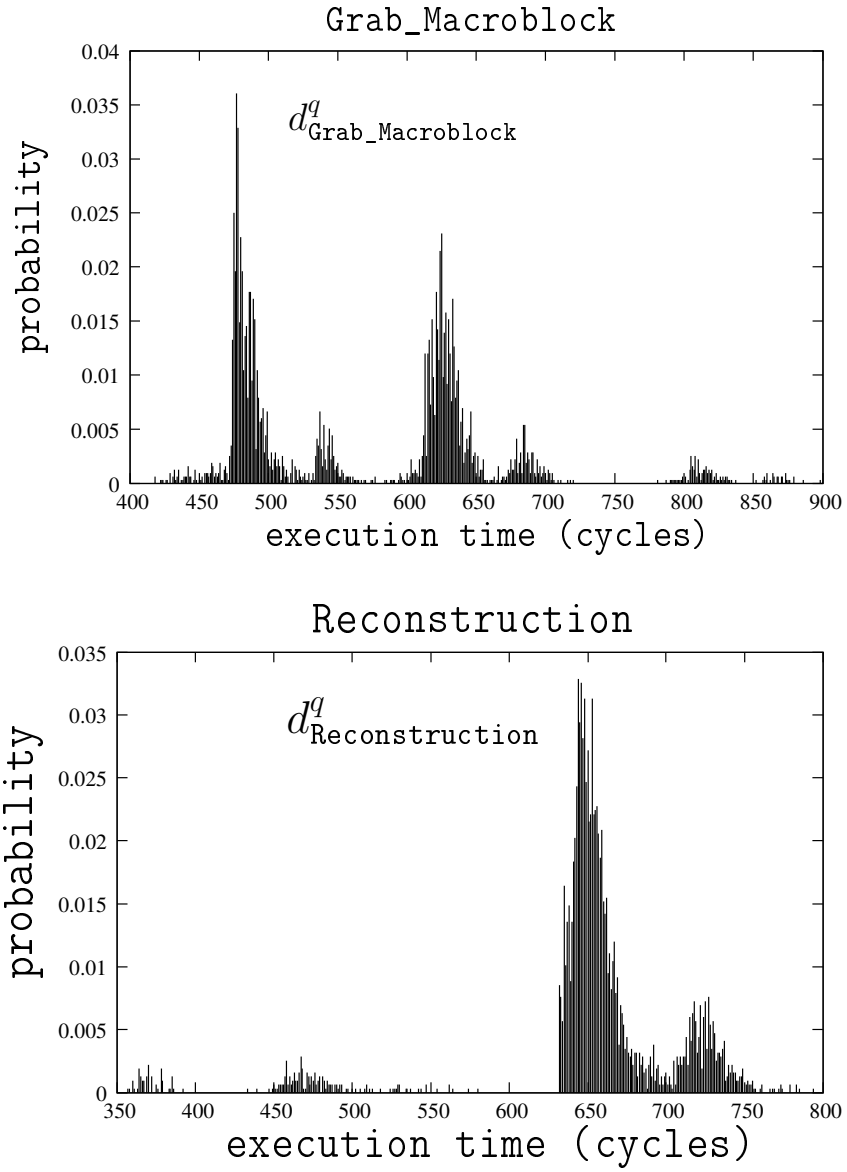


Figure 5.13 — Exemple de distributions des actions non contrôlables

La figure 5.14 représente les distributions de l'action `MotionEstimation`. Nous avons représenté les distributions de cette action pour quatre niveaux de qualité $q = 3$, $q = 4$, $q = 5$ et $q = 6$. Nous voyons sur la figure 5.14 les différences entre les durées d'exécution pour les différentes qualités. Nous pouvons observer que pour la qualité $q = 3$, la durée d'exécution possède une forte probabilité d'avoir une valeur comprise entre 1000 cycles et 2000 cycles. Tandis que pour le niveau de qualité $q = 6$ la valeur sera probablement

supérieure à 5000 cycles.

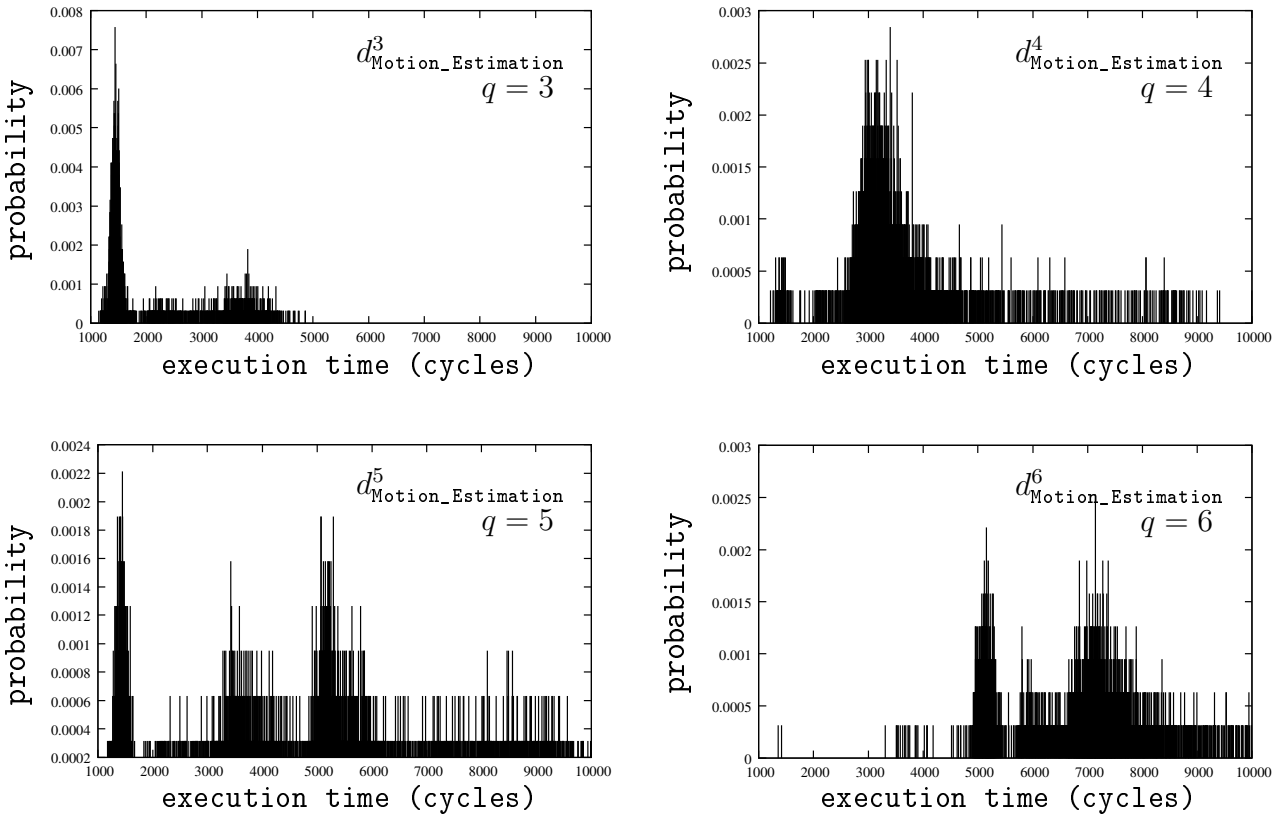


Figure 5.14 — Exemple de distributions des actions contrôlables

À partir de ces entrées, l'outil va pouvoir calculer des durées d'exécution pire-cas et moyennes basés sur la technique présentée dans la section 4.1. Ces durées d'exécution vont nous permettre de donner une analyse de performances axée sur deux points, le taux de violation des échéances et le taux d'utilisation du budget.

5.5.2 Analyse de performances

L'analyse de performances est réalisée à partir de la comparaison entre deux ordonnancements. Ces ordonnancements sont constitués des mêmes actions. Le premier est celui présenté dans la section 5.1.2.2, nous le nommerons *ordonnancement #2*. Le deuxième est celui utilisé par certains encodeur MPEG-4 (par exemple Xvid). Il est représenté par la séquence d'actions suivantes :

`Grab_Picture` $\{$`Grab_Macroblock Motion_Estimation`$\}^n\{$`DCT Quant IQuant IDCT Intra_Prediction Coding Reconstruction`$\}^n$

Nous le nommerons *ordonnancement #1*.

Pour ces deux ordonnancements, nous avons calculé le taux de violation des échéances attendu $\mu(\Gamma_\tau^{mx})$ et l'utilisation attendue du budget de temps $E(t_n)$ pour différentes valeurs du seuil de tolérance τ (voir figure 5.16 et 5.17). Nous exprimons l'utilisation du budget de temps en termes de pourcentage par rapport à l'échéance D. Plus on se rapproche de l'échéance, plus l'utilisation des ressources est optimale.

Le calcul de $\mu(\Gamma_\tau^{mx})$ et $E(t_n)$ nécessite de construire la distribution de probabilité d_n de la date de terminaison de l'application t_n. Le graphique 5.15 montre un exemple des distributions d_n. Celles-ci ont été obtenue pour l'ordonnancement #1 et pour des seuils de tolérance de $\tau = 0$, $\tau = 0\,.\,25$ et $\tau = 0\,.\,5$.

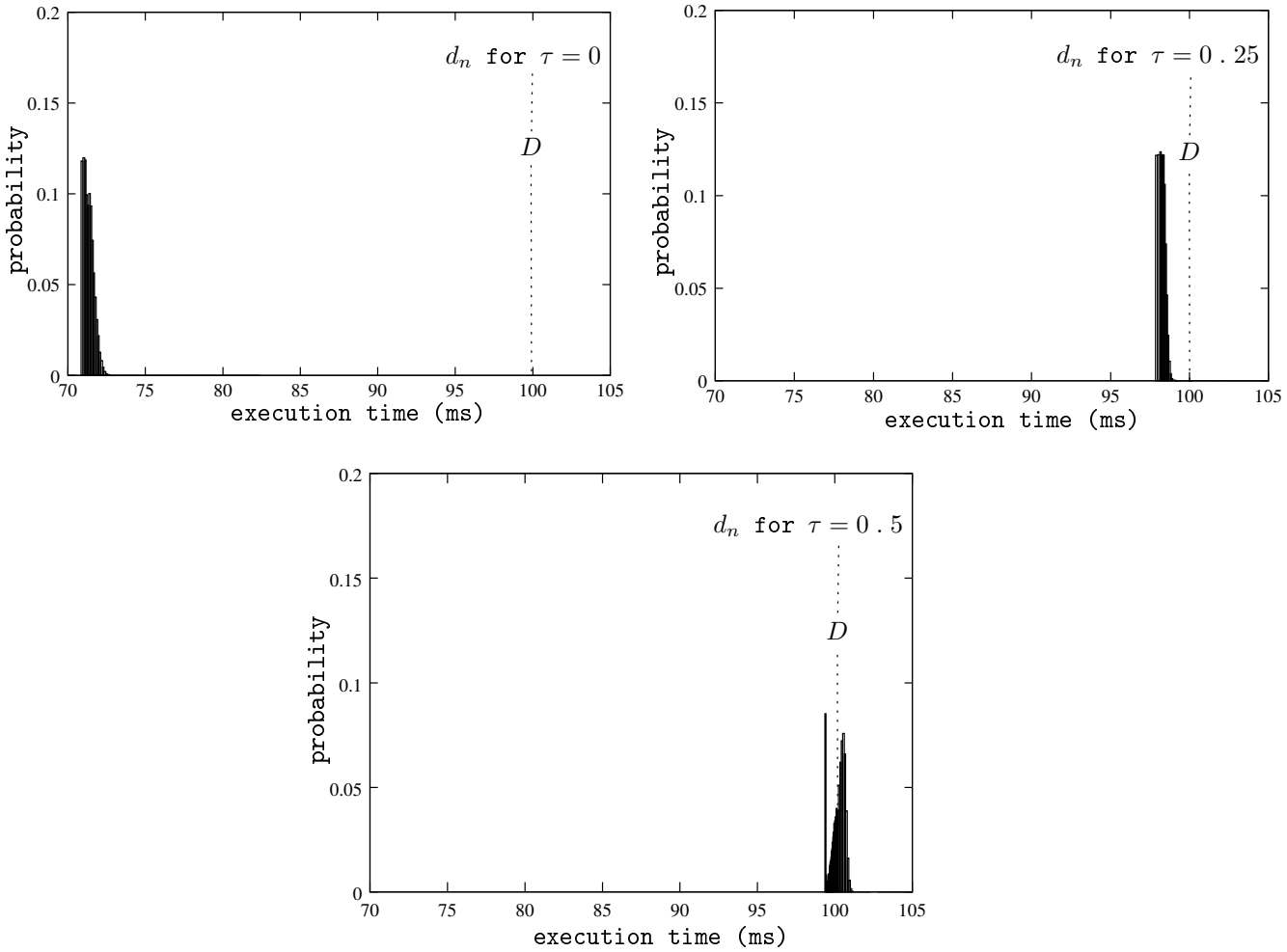


Figure 5.15 — Distributions finales

Comme nous l'avons expliqué dans le chapitre 4, l'utilisation de budget de temps et le non respect des échéances est croissant quand le seuil de tolérance τ est lui aussi croissant. Pour des valeurs de τ suffisamment élevées (approximativement $\tau > 0\,.\,4$), le

taux de violation des échéances tout comme le budget d'utilisation devient constant. En effet pour ces valeurs, la politique de gestion de qualité stochastique est équivalente à la politique de gestion de qualité moyenne, c'est-à-dire $C_\tau^{mx} = C^{av}$. Ainsi l'encodeur vidéo se comporte comme une application issue uniquement du monde du temps réel mou.

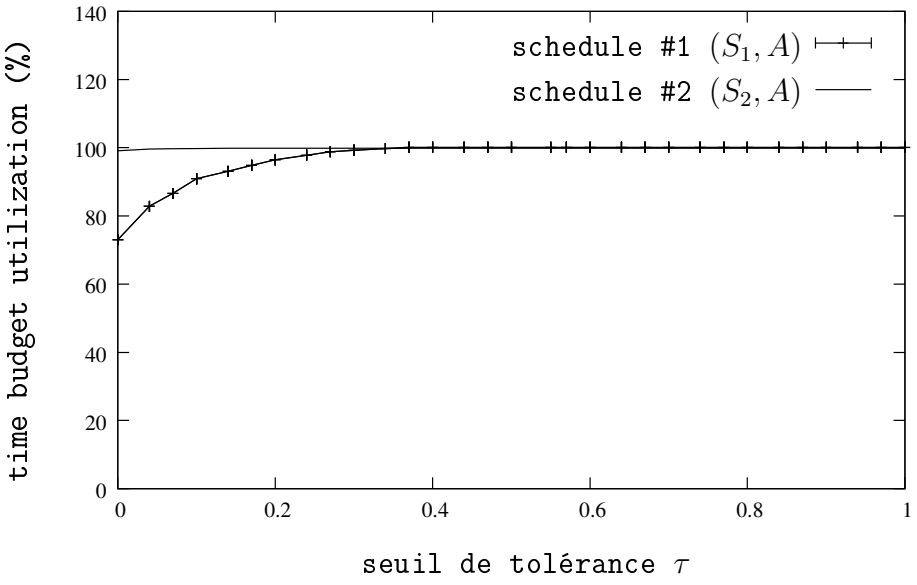


Figure 5.16 — Taux du budget de temps utilisé

La différence entre les résultats obtenus avec les ordonnancements #1 et #2 vient de la position des actions contrôlables dans l'ordonnancement. Les actions contrôlables sont réparties tout au long de l'ordonnancement #2, tandis que pour l'ordonnancement, elles ont été regroupées aux début de la séquence. En conséquence, le Gestionnaire de Qualité garde le contrôle sur les durées d'exécution tout au long de l'exécution de l'application pour l'ordonnancement #2. Dans le cas de l'ordonnancement #1, une fois que toutes les actions contrôlables ont été exécutées, le Gestionnaire de Qualité ne peut plus contrôler l'application. En effet, même si les durées d'exécution dépassent les durées d'exécution attendues, le Gestionnaire de Qualité ne peut intervenir pour rattraper les erreurs probables. En d'autres termes, il n'a plus le contrôle sur l'incertitude des durées d'exécution des actions non contrôlables.

Si nous ajoutons l'incontrôlabilité à l'incertitude sur les durées d'exécution, cela mène à des performances très pauvres avec l'utilisation de l'ordonnancement #1. Le graphique 5.16 montre une perte d'utilisation des ressources importante en appliquant un seuil de tolérance de $\tau < 0.2$ sur l'ordonnancement #1. De plus nous voyons sur le graphique 5.17 qu'avec un seuil de tolérance de $\tau > 0.4$ nous atteignons un taux de violation des échéances de 50 %.

Pour une application de ce type, notre approche stochastique peut être utile pour laisser la possibilité au programmeur de faire un compromis adéquat entre le taux de violation des échéances et l'utilisation des ressources. Cette possibilité lui est laissée en

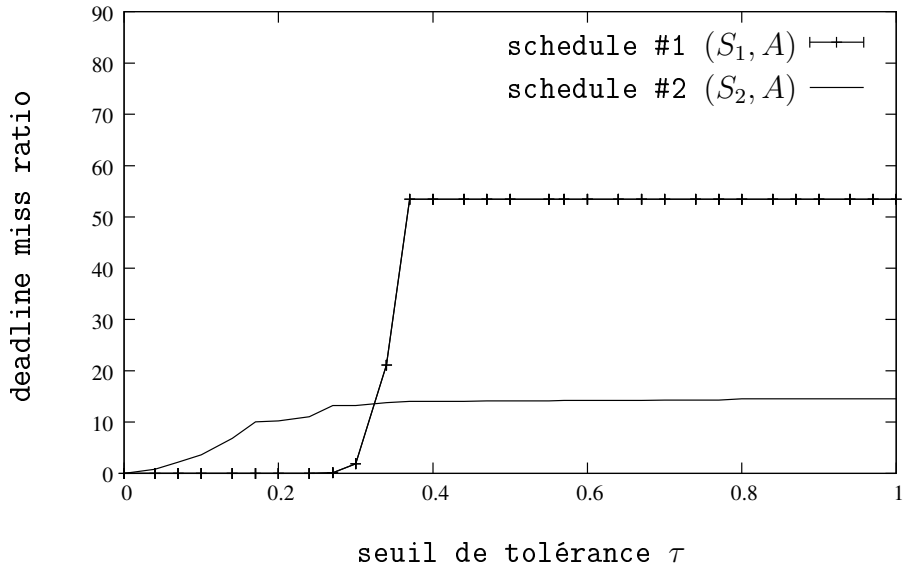


Figure 5.17 — Taux de violation des échéances attendues

ajustant le seuil de tolérance τ.

5.5.3 Résultats sur la plateforme

Nous allons pour finir présenter des résultats réalisés sur l'encodeur vidéo s'exécutant sur la plateforme cible. Nous prendrons pour ces résultats une séquence d'entrée de 80 frames. L'ordonnancement choisi pour ces expérimentations est l'ordonnancement #1.

Les résultats que nous obtenons sur la plateforme confirment ceux obtenus par l'analyse de performances. En effet le taux de violation des échéances et le budget de temps utilisé correspondent à ce que l'on a pu observer sur l'analyse de performances. Sur le graphique 5.18, nous proposons d'observer les taux d'utilisation du budget de temps pour deux valeurs différentes du seuil de tolérance, $\tau = 0\ .\ 1$ et $\tau = 0\ .\ 3$. Les pics correspondent aux *frameskip* qui se produisent lorsque l'échéance est dépassée. Pour un seuil de tolérance $\tau = 0\ .\ 1$ aucune échéance n'a été dépassée. Au contraire, le nombre de *frameskip* augmente dès que l'on utilise un seuil de tolérance plus élevé. En ce qui concerne l'utilisation du budget de temps, nous pouvons là aussi constater que les résultats obtenus sur la plateforme correspondent à ceux de l'analyse de performances. En effet l'utilisation du budget de temps moyen pour $\tau = 0\ .\ 1$ est égale à 85% et à 95% pour $\tau = 0\ .\ 3$.

Les résultats obtenus pour l'ordonnancement #2, nous confirment que l'utilisation de la politique de gestion de qualité stochastique n'est pas pertinente pour ce type d'ordonnancement. En effet, les propriétés de l'ordonnancement #2 permettent d'exécuter l'application avec un seuil de tolérance $\tau = 0$, ce qui correspond à l'utilisation de la politique d'ordonnancement mixte standard, sans réduire l'utilisation du budget de temps.

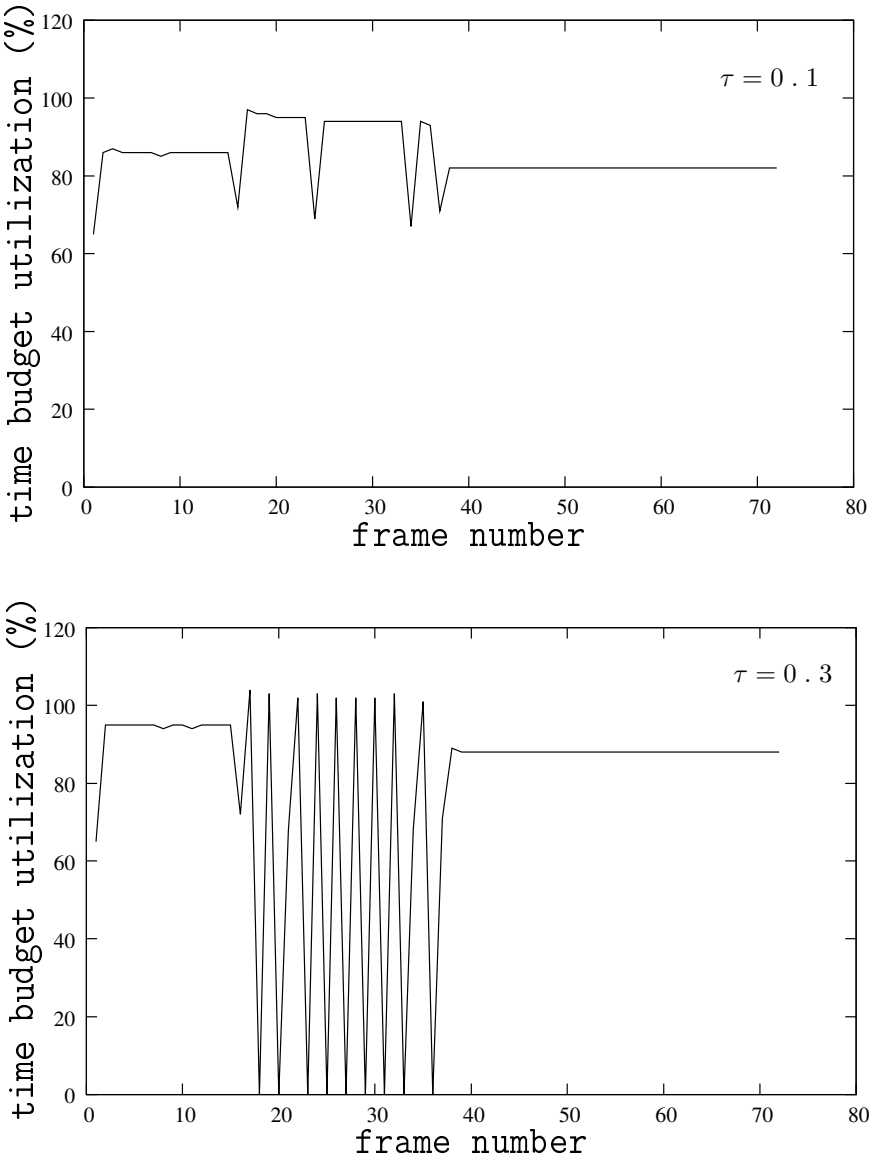


Figure 5.18 — Utilisation du processeur

5.6 Conclusion

Ce chapitre nous montre l'apport de nos contributions sur une cas de test réel, un encodeur vidéo s'exécutant sur une machine nue.

Nous avons dans une première partie pu voir l'apport de l'utilisation d'un contrôleur de qualité sur une application multimédia. Dans un deuxième temps, nous avons proposé les résultats des différentes politiques proposées dans le chapitre 2. Ainsi les résultats théoriques de la politique de gestion de qualité mixte ont été appuyés par les résultats pratiques.

La deuxième partie de ce chapitre a été entièrement consacrée aux contributions apportées par ce travail de thèse. Premièrement nous avons pu comparer l'approche symbolique et l'approche énumérative. Malgré un surcoût supérieur en ce qui concerne la mémoire, nous avons pu observer un gain de temps permettant à l'approche symbolique d'avoir des choix de qualité plus élevés. Dans une dernière partie, nous avons montré l'utilisation de l'approche stochastique. Nous avons montré que cette approche donne la possibilité au développeur de connaître à priori le comportement de son application mais aussi d'en mesurer les performances.

CHAPITRE

Conclusion et Perspectives

6

6.1 Conclusion

Ce travail de thèse s'est orienté vers l'étude des applications multimédia embarquées. Comme nous avons pu le voir, ce type d'applications possède certaines contraintes devant être gérées pour le bien de leurs exécutions. Par exemple, il est nécessaire de répondre à des contraintes d'utilisation des ressources disponibles. Dans ce travail, nous nous sommes intéressés aux contraintes de temps. Nous avons montré que le respect de ces contraintes était utile pour une application multimédia embarquée, comme un encodeur vidéo. En effet, cela permet d'obtenir un qualité de service que l'utilisateur va juger satisfaisant, ce qui n'aurait pas été le cas avec des échéances violées. Il est assez simple de trouver des mécanismes permettant de ne violer aucune échéance, cependant ceux-ci ne respectent pas le fait que l'on veuille assurer une utilisation maximale des ressources. Pour cette raison, il est intéressant de contrôler de telles applications. Le contrôleur aura pour but d'utiliser une qualité maximale tout en respectant les contraintes de temps. Nous nous sommes aussi intéressés à une troisième propriété qui influe elle aussi sur le rendu de l'application. Il s'agit d'avoir un choix de niveau de qualité régulier tout au long de l'application. [34] propose une technique de contrôle basée sur une contrôleur ad-hoc. Celui-ci est appelé pour chaque action contrôlable de l'application. La politique de gestion de qualité mixte proposée permet de répondre à ces trois propriétés correctement. C'est-à-dire qu'elle permet le respect des contraintes temporelles tout en utilisant le maximum des ressources attribuées à l'application, mais elle assure aussi une certaine régularité dans le choix des qualités.

Cependant cette méthode est basée sur la combinaison des comportements pire-cas et moyens-cas et l'utilisation d'une marge de sécurité. Ce qui la rend difficile à comprendre et à interpréter. De plus la complexité des calculs de cette méthode peut mener à un surcoût en termes de durées d'exécution assez grand. Ce surcoût peut s'avérer intolérable pour les industriels. En effet, le temps passé à réaliser le contrôle est du temps perdu pour l'application. Nous visons aussi des applications multimédia qui sont des applications n'ayant pas la nécessité de respecter toutes les échéances. Si ce phénomène se produit, cela ne produira pas une situation critique, mais seulement une dégradation de la qualité du rendu. On peut donc se demander s'il est nécessaire de respecter à tous prix les échéances. Un dernier point qui n'est pas proposé par la technique de contrôle précédente, correspond à la possibilité de connaître à priori le comportement de l'application. Dans l'état actuel des choses, il était nécessaire d'exécuter l'application et de voir si le comportement cor-

respond effectivement à ce que le programmeur voulait. Dans ce travail de thèse, nous avons donc proposé deux outils permettant de résoudre les problèmes précédents.

Contrôleur Symbolique

Le premier est un contrôleur basé sur une approche symbolique de l'application. Pour permettre une interprétation de la politique mixte, nous avons proposé une représentation graphique du comportement de l'application contrôlée. À partir de cette représentation, nous avons pu expliquer le fonctionnement d'un contrôleur, mais nous avons donné une interprétation plus précise de la politique de gestion de qualité mixte. En effet, nous avons pu comprendre l'utilité de la marge de sécurité. Mais aussi visualiser son utilisation permettant à l'application contrôlée de répondre aux propriétés de sûreté et d'optimalité. D'autre part nous avons, à partir de ce diagramme des vitesses, pu représenter les régions de qualité. Ces régions spécifient l'ensemble des états où il est possible de choisir un niveau de qualité. A partir de ces régions, nous avons pu proposer une technique de contrôle symbolique.

Grâce à ce travail sur le symbolique nous avons obtenu des résultats non seulement au niveau de l'interprétation de la politique de contrôle mixte mais aussi sur le surcoût qui existe avec l'utilisation de telles techniques de contrôle.

Contrôleur Stochastique

Dans une seconde partie de ce travail de thèse, nous avons proposé une technique de contrôle stochastique. Cette méthode conserve les bases fixées par la technique de gestion de qualité mixte. Cependant elle s'oriente d'une manière totalement différente.

En effet, contrairement à la technique de contrôle initiale qui se base sur un modèle de temps composé des durées d'exécution moyens et pire-cas, l'approche stochastique prend pour modèle de temps une distribution de probabilités des durées d'exécution des actions. Cette première différence avec la politique précédente va permettre d'obtenir une estimation des durées d'exécution d'entrée plus précise. En effet, nous allons à partir de ces distributions générer des durées d'exécution moyennes et des durées d'exécution pire-cas. Ces dernières seront fixées à l'aide d'un seuil de tolérance fourni par l'utilisateur. La technique de contrôle n'est pas fondamentalement modifiée en ce qui concerne le contrôle, mais des résultats intéressants pour le programmeur ont pu être présentés.

Le premier d'entre eux vient de l'utilisation des distributions en entrées. Le fait d'avoir les distributions de chaque action pour chaque qualité, nous donne la possibilité de construire une distribution de l'ensemble de l'application. Nous avons proposé un algorithme de convolution paramétrée. Il va nous permettre de construire la distribution de l'application pour la dernière action en tenant compte de la politique de gestion de qualité utilisée mais aussi des niveaux de qualité qui pourraient être choisis lors d'une exécution réelle. Cela correspond à une avancée intéressante dans le sens où l'on va pouvoir

connaître à priori le comportement de l'exécution en fonction des paramètres de l'application. La construction de la distribution finale va donc permettre au programmeur de modifier ses données d'entrées et de savoir également comment les changer.

Le deuxième et principal résultat vient de l'utilisation du seuil de tolérance. Les applications que nous visons, ne nécessitent pas de respecter toutes leurs contraintes temporelles, mais seulement en manquer le moins possible. Pour cela la technique de contrôle mixte est trop restrictive du fait de son aspect sûr. Nous proposons donc de borner les temps d'exécution non plus avec un pire-cas surévalué mais avec un pire-cas probabiliste. Celui-ci est fixé par le seuil de tolérance en prenant en compte les distributions de probabilité des actions. C'est-à-dire que nous allons tolérer que les durées d'exécution réelles dépassent une certaine valeur pour une probabilité fixée par l'utilisateur. A partir de là, nous avons présenté une méthode permettant de connaître, en fonction du seuil de tolérance qui a été fixé, la probabilité de dépasser l'échéance globale.

Si l'on regroupe les deux résultats précédents, on obtient une analyse de la performance pour l'application. En effet, il est possible de calculer non seulement le taux de violation des échéances, mais aussi le taux d'utilisation des ressources. Le programmeur aura donc un contrôle global de son outil, et il pourra adapter les paramètres en fonction des résultats obtenus par l'analyse.

Cette approche peut être considérée comme du temps-réel mou mais maîtrisé. En effet, nous laissons l'application violer les échéances, mais le programmeur connaît exactement le taux de violation de celle-ci et le fixe lui-même.

6.2 Perspectives

Nous pouvons voir par rapport à ce travail de thèse, deux types de perspectives. Le premier est celui qui se rapporte directement aux approches abordées dans les chapitres précédents, c'est-à-dire des améliorations des techniques de contrôle présentées. Le deuxième type de perspectives correspond à des ouvertures vers un travail sur le contrôle d'application en général.

Amélioration de l'approche Symbolique

Pour la première partie du travail, nous avons réduit le surcoût du contrôleur en termes de durée d'exécution, mais nous avons pu voir que cette technique reste coûteuse en termes d'espace mémoire. De plus nous ne pouvons pas stocker toutes les valeurs de relâchement possibles. Il serait bon d'orienter la suite du travail en réduisant ce surcoût en mémoire. Quelques pistes de représentation peuvent être mises en avant, comme une représentation linéaire des bornes des régions, ou l'utilisation d'enveloppes convexes. A partir de ces représentations, de nombreux outils existent pour savoir si tel ou tel point fait partie de l'enveloppe. Ce pas peut être nécessaire à l'utilisation de la technique de

relâchement au niveau industriel.

Amélioration de l'approche Stochastique

L'approche stochastique prend pour hypothèse une abstraction forte, l'indépendance sur les distributions de probabilités des durées d'exécution. Bien que cette abstraction ne soit pas néfaste pour la théorie, il serait plus précis d'utiliser la dépendance entre ces durées d'exécution. Ainsi la méthode serait complète et pourrait être généralisée à d'autres applications.

Modèle d'entrée de l'application

Le modèle d'entrée de l'application est un graphe de précédence paramétré par des niveaux de qualité, muni des durées d'exécution moyennes et pire-cas. Ce modèle est assez naturel pour représenter des applications multimédia de type flots de données. Cependant il subsiste un effort à faire si l'on veut rendre l'approche utilisable dans le monde industriel. Le graphe de précédence reste très restrictif, dans le sens où il ne capte pas la notion de conditionnelle. Cette abstraction ne permet pas une modélisation à un niveau de granularité intéressant lorsque l'algorithme comporte des structures conditionnelles à haut niveau. Il existe quelques langages de programmation (Ptolemy, StreamIt) proposant des structures de ce type. Il peut être intéressant de greffer notre technique de contrôle sur leur modèle et ainsi de voir comment contrôler ces modèles flots de données. D'autre part le modèle des niveaux de qualité entier est très pauvre, et ne tient pas compte des interactions qu'il peut exister entre les différents niveaux de précision des calculs effectués. Par exemple, il n'est pas nécessaire d'encoder toutes les zones d'une image avec le même niveau de qualité. De plus, la répartition du budget de temps ne doit pas nécessairement être réalisée de manière uniforme, et peut dépendre des données d'entrée. Pour l'instant tous ces aspects sont gérés manuellement par un choix intelligent de la signification des niveaux de qualité. Il serait intéressant de faire évoluer le modèle des qualités pour traiter ces aspects à travers l'outil.

Ajout du niveau d'apprentissage

Pour améliorer les politiques déjà utilisées, il pourrait être intéressant d'ajouter au contrôleur une partie apprentissage des durées d'exécution. Ainsi en fonction des données d'entrée, il serait possible d'actualiser les durées d'exécution moyennes. Pour exemple, à partir des paramètres d'entrée d'une action, l'utilisation des réseaux de neurones pourrait raffiner en ligne les durées d'exécution moyen. L'utilisation de ces durées d'exécution raffinées peut mener à une meilleure utilisation du budget de temps et ainsi réduire la charge du processeur. Ces techniques d'apprentissage permettent de mieux prévoir le comportement du système et donc de faire des choix de contrôle plus pertinents. Ils pourraient aussi être utilisés pour générer uniquement à partir de l'apprentissage les

durées d'exécution ce qui faciliterait la génération du modèle de temps.

Architecture parallèle

Notre approche consiste à simplifier le problème du contrôle d'application en prenant pour hypothèse l'utilisation d'une architecture mono-processeur. Cette hypothèse est très restrictive, surtout dans le domaine du multimédia où l'on s'oriente le plus souvent vers des plateformes multi-processeurs. Une perspective pourrait être d'adapter l'utilisation du contrôleur à de telles architectures.

Application à la minimisation de la mémoire utilisée

La technique de contrôle peut aussi se baser sur d'autres critères. Nous avons vu tout au long de cette thèse un seul critère d'optimisation de l'application, la qualité. Cependant d'autres travaux nous permettent de penser qu'il serait tout aussi possible de contrôler les ressources de la plateforme, comme la taille de mémoire utilisée. La théorie de contrôle peut être adaptée en ne se focalisant plus uniquement sur le budget de temps à utiliser mais aussi en ayant pour borne une quantité de mémoire que l'on ne veut pas dépasser. Ce problème pourrait être traité efficacement avec le contrôleur actuel, en prenant en compte les consommations mémoire pire-cas et non les durées d'exécution pire-cas. Nous pourrions introduire un deuxième critère d'optimisation de l'application.

Application à la consommation d'énergie

Une autre application de la technique de contrôle pourrait servir à contrôler la fréquence d'horloge du processeur dans le cadre de la réduction de la consommation d'énergie. Le contrôleur pourrait donc agir sur la qualité mais aussi sur la fréquence du processeur. Ainsi la minimisation d'énergie pourrait faire partie des critères d'optimisation du contrôleur.

Contrôleur multi-critères

Les deux dernières perspectives permettraient de composer un contrôleur, pouvant gérer plusieurs critères, que ce soit le budget de temps, la consommation mémoire ou la fréquence du processeur. En effet une perspective importante est la gestion du multi-critère. Il serait intéressant de voir comment gérer intelligemment les différents critères sans perdre d'efficacité, ou sans contraindre l'un plutôt que l'autre. L'utilisateur pourrait aussi avoir la possibilité de choisir des priorités par rapport aux différents critères à respecter.

Bibliographie

[1] F.S. Rovati, D. Pau, E. Piccinelli, L. Pezzoni, and J.-M. Bard. An innovative, high quality and search window independent motionestimation algorithm and architecture for mpeg-2 encoding. In *IEEE Transactions on Consumer Electronics*, volume 46, pages 697–705. IEEE, 2000.

[2] R. J. Bril, M. Gabrani, C. Hentschel, G. C. van Loo, and E. F. M. Steffens. QoS for consumer terminals and its support for product families. In *Proceedings of the International Conference on Media Futures*, 2001.

[3] Damir Isovic, Gerhard Fohler, and Liesbeth Steffens. Timing constraints of MPEG-2 decoding for high quality video : misconceptions and realistic assumptions.

[4] S. Graham and P. Kumar. The convergence of control, communication, and computation, 2003.

[5] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *Trans. on Embedded Computing Sys.*, 7(3) :1–53, 2008.

[6] Kelvin D. Nilsen and Bernt Rygg. Worst-case execution time analysis on modern processors. In *Workshop on Languages, Compilers, Tools for Real-Time Systems*, pages 20–30, 1995.

[7] NIST Report. The economic impacts of inadequate infrastructure for software testing. Technical report.

[8] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1) :46–61, 1973.

[9] A. Burns. Preemptive priority based scheduling : An appropriate engineering approach, 1994.

[10] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : Exact characterization and average case behavior. In *Real-Time Systems Symposium*, pages 201–209, 1989.

[11] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. *rapport technique*, 1952.

[12] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Cambridge, MA, USA, 1983.

[13] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th ACM Symposium on Operating Systems Principles, SOSP'01C*. IEEE, 2001.

[14] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *RTSS '98 : Proceedings of the IEEE Real-Time Systems Symposium*, page 123, Washington, DC, USA, 1998. IEEE Computer Society.

[15] J.P Lehoczky, L. Sha, and J. K. Strosnider. An optimal algorithm for scheduling soft-aperiodic tasks fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time System Symposium*, pages 110–123, 1992.

[16] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 222–231, 1993.

[17] J.P Lehoczky and S.Thuel. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Proceedings of the IEEE Real-Time System Symposium*, 1994.

[18] Dieter Haban and Kang G. Shin. Application of real-time monitoring to scheduling tasks with random execution times. *IEEE Trans. Softw. Eng.*, 16(12) :1374–1389, 1990.

[19] N. C. Audsley, R. I. Davis, and A. Burns. Mechanisms for enhancing the flexibility and utility of hard real-time systems. In *Real-Time Systems Symposium*, pages 12–21. IEEE, 1994.

[20] Prabha Gopinath and A. Gupta. Applying compiler techniques to scheduling in real-time systems. In *IEEE Real-Time Systems Symposium*, pages 247–256, 1990.

[21] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th ACM Symposium on Operating Systems Principles, SOSP'01C*. IEEE, 2001.

[22] K. J. Liu, S. Natarajan, J.W.-S. Liu, and T. Krauskopf. Concord : A system of imprecise computations. In *Compsac*. IEEE, Oct. 1987.

[23] K. J. Liu, S. Natarajan, and J.W.-S. Liu. Imprecise results : Utilizing partial computations in real-time systems. In *Real-Time Systems Symposium*. IEEE, 1987.

[24] K. J. Liu, S. Natarajan, J.W.-S. Liu, and T. Krauskopf. Scheduling real-time, periodic jobs using imprecise results. In *Real-Time Systems Symposium*. IEEE, 1987.

[25] Jen-Yao Chung, Jane W. S. Liu, and Kwei-Jay Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Comput.*, 39(9) :1156–1174, 1990.

[26] C. Lu, J Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling : Framework, modeling and algorithm. *special issue of RT Systems Journal on Control-Theoric Approach To Real-Time Computing*, 23(1/2) :85–88, 2002.

[27] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *RTSS '98 : Proceedings of*

the IEEE Real-Time Systems Symposium, page 307, Washington, DC, USA, 1998. IEEE Computer Society.

[28] M. Mattavelli, S. Brunetton, and D. Mlynek. Computational graceful degradation for video sequence decoding. In *ICIP '97 : Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 1*, page 330, Washington, DC, USA, 1997. IEEE Computer Society.

[29] G. Lafruit, L. Nachtergaele, K. Denolf, and J. Bormans. 3D computational graceful degradation. In *The 2000 IEEE International Symposium on Circuits and Systems*, volume 3, pages 547–550. IEEE, 2000.

[30] Gene Franklin, Dave Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*. Addison Wesley Publishing Co.

[31] Clemens C. Wüst, Liesbeth Steffens, Reinder J. Bril, and Wim F.J. Verhaegh. Qos control strategies for high-quality video processing. In *Euromicro Conference on Real-Time Systems*, pages 3–12. IEEE, 2004.

[32] G. Koren and D. Shasha. Skip-over : Algorithms and complexity for overloaded systems that allow skips. Technical Report TR1996-715, , 1996.

[33] Laurentiu Papalau, Clara M. Otero Pérez, and Liesbeth Steffens. In Steve Goddard, editor, *Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems*, pages 33–36, 2004.

[34] Jacques Combaz. *Conception de Systèmes Adaptatifs Sûres et Optimaux*. PhD en informatique, Université Joseph Fourier, 2006.

[35] J. Combaz, J-C. Fernandez, T. Lepley, and J. Sifakis. QoS Control for Optimality and Safety. In *Proceedings of the 5th Conference on Embedded Software*, September 2005.

[36] Guido M. Schuster, Gerry Melnikov, and Aggelos K. Katsaggelos. A review of the minimum maximum criterion for optimal bit allocation among dependent quantizers. *IEEE Transactions on Multimedia*, 1(1) :3–17, 1999.

[37] P.H. Westerink, R. Rajogopalan, and C.A. Gonzales. Two-pass MPEG-2 variable bit-rate encoding. 43(4), 1999.

[38] J. Combaz, J.C. Fernandez, T. Lepley, and J. Sifakis. Fine grain QoS control for multimedia application software. In *Design, Automation and Test in Europe (DATE'05) Volume 2*, pages 1038–1043, 2005.

[39] Vivek Sarkar. Determining average program execution times and their variance. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 298–312, 1989.

[40] Joseph Sifakis Jacques Combaz, Jean-Claude Fernandez and Loïc Strus. Using speed diagram for symbolic quality management. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, mars 2007.

[41] Joseph Sifakis Jacques Combaz, Jean-Claude Fernandez and Loïc Strus. Symbolic quality control for multimedia applications. *Journal of Real Time System*, 2008.

[42] ISO/IEC 14496-10 H264.

[43] http://iphome.hhi.de/suehring/tml/index.htm.

[44] ISO/IEC 14496-2 MPEG-4.

[45] http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.

Table des figures