Évaluation de politiques d'adaptation au risque de collisions dans un consensus de type "Fast Paxos"

Izabela Moise, Michel Hurfin, Jean-Pierre Le Narzul, Frédéric Majorczyk

RenPar'20 / SympA'14 / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Evaluation de politiques d'adaptation au risque de collisions dans un consensus de type "Fast Paxos"

Izabela Moise[1], Michel Hurfin[2], Jean-Pierre Le Narzul[3] et Frédéric Majorczyk[1]

1 : Université de Rennes 1, IRISA / INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
2 : INRIA Rennes Bretagne Atlantique, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
3 : Institut Télécom/Télécom Bretagne, 2 rue de la Chataigneraie, 35576 Cesson Sévigné, France

---

Résumé
Aucune collision ne se produit durant une instance de consensus si toutes les valeurs proposées durant ce consensus sont identiques. Afin de réduire le temps nécessaire à une prise de décision, certains protocoles de consensus anticipent sur le fait qu'aucune collision ne se produira au cours du prochain consensus exécuté. Ce principe de conception a notamment été étudié par Leslie Lamport qui a proposé une variante au protocole *Paxos*, appelée *Fast Paxos*. En l'absence de collision, un gain de temps est observé. Malheureusement, le fait de déclencher cette optimisation revient à faire un pari sur l'avenir et, en cas de collision non prévue, le coût de la procédure de recouvrement s'avère être bien plus important que le gain initialement escompté. Dans cet article, nous décrivons brièvement le protocole Paxos-MIC qui permet d'exécuter une séquence d'instances de consensus. Après chaque consensus, le choix d'activer l'optimisation risquée durant le prochain consensus est fait localement et dynamiquement en évaluant une condition de déclenchement. Nous étudions différentes conditions de déclenchement et nous évaluons les gains susceptibles d'être obtenus en simulant l'exécution du protocole à l'aide d'une trace réelle correspondant à l'activité du site Web d'une grande école d'ingénieur durant quinze jours consécutifs.

Mots-clés : Protocole d' accord, Consensus, Collision, Paxos, Fast Paxos

---

1. Introduction

Nous considérons un système réparti où les noeuds communiquent par échange de messages et peuvent connaître des défaillances de type *panne franche* ou *omission* lors de communications. Un noeud qui ne tombe jamais en panne est dit *correct*. Un nœud *incorrect* s'exécute conformément à sa spécification mais arrête prématurément et définitivement son exécution durant le calcul. Un message peut être dupliqué ou perdu mais n'est jamais altéré. Dans ce système, nous nous intéressons au problème du Consensus. Au cours d'une instance de consensus (identifiée par k), des valeurs initiales (potentiellement distinctes) peuvent être proposées par un ou plusieurs noeuds appelés des *auteurs de propositions*. Durant l'instance k, le ou les auteurs communiquent leur proposition au même sous-ensemble de n noeuds du système. Ces n noeuds que nous appellerons par la suite les *acteurs* sont directement impliqués dans l'exécution du protocole de consensus. Ils doivent collaborer pour assurer que celui-ci converge inéluctablement (terminaison) vers une valeur de décision unique (accord uniforme) qui doit nécessairement être l'une des valeurs proposées (validité). La valeur de décision est alors retournée vers tous les auteurs qui ont fourni (ou fourniront) une valeur initiale concernant cette instance de consensus particulière. Le problème du consensus suscite beaucoup d'intérêts. Sur un plan pratique, une solution à ce problème peut être une brique de base pour résoudre d'autres problèmes d'accord (diffusion atomique, tolérance aux défaillances fondée sur la réplication, ...). Il convient donc d'optimiser les performances des protocoles proposés puisque ce service est en général destiné à être très fréquemment activé. Deuxièmement, sur un plan théorique, le résultat d'impossibilité attaché à ce problème [1], a motivé de nombreux travaux de recherche. En particulier, des protocoles déterministes indulgents ont été proposés afin de résoudre le problème du consensus dans un environnement asynchrone enrichi de services (détecteurs

de défaillances, élection de leader, ...) sous l'hypothèse que ces oracles garantissent inéluctablement un niveau minimum de qualité de service.

Dans cet article nous nous intéressons aux protocoles s'appuyant sur un service d'élection de leader et, plus précisément, nous considérons les protocoles dérivés du protocole Paxos [3] dont la version originale fut décrite par Lamport dans [2]. Ces protocoles reposent sur la notion de leader ultime : il existe un instant à partir duquel un acteur correct est considéré par tous comme étant le seul leader (et ceci durant un laps de temps suffisant pour qu'une décision puisse être prise). Les protocoles de la famille Paxos s'appuient sur la notion de quorum majoritaire [4] : si f est le nombre maximal de pannes pouvant affecter le groupe de n acteurs alors $f < n/2$. Depuis la présentation de la version initiale [2, 3], deux optimisations majeures ont été proposées dans le cas du protocole Paxos. La première, que nous dénotons par la suite $\mathbf{O_1}$, a été brièvement suggérée par Lamport [3] puis reprise dans plusieurs articles dont notamment [5, 6]. La seconde, dénotée ici $\mathbf{O_2}$, est au coeur de la solution Fast Paxos présentée par Lamport dans [7]. Les deux optimisations sont compatibles et toutes deux visent à réduire le nombre d'étapes de communication devant être effectuées entre l'instant où un premier auteur transmet une proposition initiale et l'instant où il recevra la valeur de décision. L'optimisation $\mathbf{O_1}$ permet un gain systématique : le nombre d'étapes de communication nécessaires passe de 6 à 4. Inversement les conséquences (bénéfiques ou néfastes) d'un déclenchement de $\mathbf{O_2}$ sont généralement imprévisibles. Dans les scénarios favorables, son activation permet de passer de 4 étapes à 3. Malheureusement, ce gain n'est pas garanti et dans les cas les moins favorables, le bénéfice attendu peut se transformer en un surcoût dû à l'exécution d'une procédure de recouvrement. Lorsqu'une collision se produit, $\mathbf{O_2}$ échoue et requiert un recouvrement. A défaut de pouvoir identifier les circonstances précises conduisant inéluctablement à un échec de $\mathbf{O_2}$, il est par contre très simple d'identifier une condition nécessaire (mais pas suffisante) pour qu'une collision se produise durant une instance de consensus : au moins deux auteurs de propositions doivent participer au consensus et fournir des valeurs initiales distinctes. Alors que $\mathbf{O_1}$ peut être assimilé à la suppression d'une phase de calcul inutile, $\mathbf{O_2}$ consiste à anticiper une partie du calcul en exécutant du code relatif au prochain consensus avant même que des valeurs initiales ne soient proposées durant ce consensus. Puisque les acteurs du protocole exécutent une séquence d'instances de consensus, le choix de déclencher $\mathbf{O_2}$ est donc pris, du point de vue des auteurs, entre deux instances de consensus à un instant où rien (semble-t-il) ne permet de présager de l'absence d'une collision lors du prochain consensus. Dans la littérature, le problème de l'activation d'une optimisation de type $\mathbf{O_2}$ est peu discuté. Dans [7], Lamport suggère que ce déclenchement se fasse selon un planning statique respecté à la lettre par tous les acteurs. Les tours *Any* (avec $\mathbf{O_2}$) et les tours *classiques* (sans $\mathbf{O_2}$) s'enchainent alors dans un ordre prédéfini et connu de tous. Dans [4], les auteurs font le choix de déclencher systématiquement une optimisation qui est également fondée sur l'utilisation de quorums de plus grande taille et l'espoir de ne pas observer de collision. En cas d'échec lors de cette première phase que l'on peut qualifier d'optimiste, de nouvelles phases de plus en plus pessimistes sont exécutées en séquence. Dans le pire cas, la dernière phase exécutée (la moins optimiste) correspondant en quelque sorte à une exécution sans activation de $\mathbf{O_2}$. Dans [8], l'activation de $\mathbf{O_2}$ est mise en oeuvre uniquement durant le premier tour d'un consensus. Dans [9], les auteurs proposent une approche originale consistant à gérer en parallèle et de manière cohérente, une exécution ne faisant intervenir que $\mathbf{O_1}$ et une exécution combinant $\mathbf{O_1}$ et $\mathbf{O_2}$. Quelles que soient les circonstances, le résultat est fourni par l'exécution la plus rapide et les échanges de messages (un peu plus nombreux) garantissent toujours l'égalité entre les deux résultats produits.

Dans [10], en proposant le protocole Paxos-MIC, l'un de nos objectifs était d'offrir un algorithme autorisant l'un des acteurs (en l'occurrence le leader) à décider dynamiquement et localement du déclenchement de $\mathbf{O_2}$. Ici, nous cherchons à évaluer l'intérêt du mécanisme de déclenchement. Nous identifions 3 stratégies statiques (« Toujours », « Jamais », « Aléatoire ») ainsi que 2 stratégies dynamiques prenant en compte le passé récent afin d'adapter constamment la stratégie de déclenchement au contexte. Nous considérons une application particulière nécessitant des consensus répétés et nous exploitons une trace réelle d'activation du protocole de consensus d'une part pour déterminer si l'optimisation $\mathbf{O_2}$ a réellement un intérêt pratique et d'autre part pour estimer la pertinence des tests statiques et dynamiques proposés. Dans le cas des tests dynamiques, la précision de la prédiction des collisions est un élément clé. Bien que la fiabilité des prédictions fluctue, celle-ci doit rester suffisamment élevée pour pouvoir influer à bon escient sur le caractère « plutôt optimiste » ou « plutôt pessimiste » des déclenchements.

Structure de l'article : Dans la section 2, nous présentons brièvement les principes généraux du pro-

tocole Paxos ainsi que les deux optimisations $\mathbf{O_1}$ et $\mathbf{O_2}$ considérées. Puis nous nous intéressons aux contextes d'expérimentations (section 3) et aux stratégies de déclenchement de $\mathbf{O_2}$ (section 4). Notre objectif est de déterminer l'intérêt de $\mathbf{O_2}$ et les niveaux de fiabilité des différents tests (section 5).

2. Le protocole Paxos et ses deux optimisations

Deux rôles distincts (*Coordinateur* et *Accepteur*) sont définis et chaque acteur joue un ou deux rôles. Plus précisément, les n acteurs (et de fait, une majorité de noeuds corrects) jouent le rôle d'accepteurs tandis qu'au moins $f+1$ acteurs (et de fait, au moins un noeud correct) jouent le rôle de coordinateur. Un coordinateur n'est actif que lorsque le service d'élection de leader le désigne comme leader. En pensant alors agir comme un leader unique et incontesté (ce qui n'est pas forcément le cas), le coordinateur tente d'imposer une valeur de décision aux autres acteurs. Dans les protocoles de la famille Paxos, un numéro de tour r est associé à chaque tentative. Ce numéro est propre au noeud leader N_i qui exécute la tentative : dans notre implémentation, si $1 \leq i \leq n$, la valeur de r choisie par le leader N_i pour identifier son tour courant doit être un multiple de i supérieur à tous les numéros de tour qu'il a pu observer par le passé (les siens et ceux utilisés par d'autres coordinateurs). Dans la version originale du protocole Paxos, un leader tente d'imposer une valeur de décision en exécutant successivement deux phases au cours d'un tour r. Au cours d'une première phase de *Préparation*, le leader s'assure que la valeur qu'il soumettra lors de la seconde phase n'est pas incompatible avec celles éventuellement soumises par d'autres coordinateurs ayant agi comme leader au cours du même consensus mais durant des tours précédents (*i.e.*, dont les numéros sont inférieurs à r). La phase de préparation implique la diffusion d'un message du leader vers l'ensemble des accepteurs puis la collecte, par le leader, de réponses favorables en provenance d'une majorité d'accepteurs (soit deux étapes de communication). La seconde phase est une phase de *Proposition*. Elle débute une fois que le leader a identifié une valeur qu'il peut soumettre sans risquer de violer les propriétés de sûreté (Accord et Validité). La valeur est diffusée par le leader vers l'ensemble des accepteurs puis le leader attend de collecter une majorité de réponses favorables avant de pouvoir considérer que cette valeur soumise est la valeur de décision (soit à nouveau deux étapes de communication). Un accepteur est une entité passive dont l'état fait référence à la dernière phase de préparation acceptée (numéro de tour r_1) et à la dernière phase de proposition acceptée (numéro de tour r_2 et valeur adoptée). Cet état ne peut évoluer que lors de la réception d'une requête diffusée par un coordinateur et à condition que cette requête soit acceptable. Une requête peut être ignorée par un accepteur si la mise à jour qu'elle entrainerait ne garantit pas i) que la valeur de r_1 croit, ii) que la valeur de r_2 croit ou iii) que $r_1 \leq r_2$ au moment d'une mise à jour de r_2.
Dans sa version originale, le protocole requiert donc 4 étapes de communications entre les acteurs auxquelles viennent s'ajouter 2 étapes "externes" correspondant à la diffusion des valeurs initiales des auteurs vers les coordinateurs et à la diffusion de la valeur de décision du leader vers les auteurs. Le chemin de communication correspond à : auteur $\rightarrow$ coordinateurs (leader) $\rightarrow$ accepteurs $\rightarrow$ leader $\rightarrow$ accepteurs $\rightarrow$ leader $\rightarrow$ auteurs. L'optimisation $\mathbf{O_1}$ consiste à supprimer une partie du calcul (la phase de préparation) lorsque celle-ci est inutile. Si le leader N_i a réussi à imposer une valeur de décision concernant le consensus $k-1$ durant la tentative r et si, du point de vue de N_i, aucun autre coordinateur N_j ne semble avoir agi avec un numéro de ronde supérieur à r ni durant l'instance de consensus $k-1$ ni durant l'instance k alors N_i peut agir en temps que leader durant le consensus k en utilisant le même numéro de tour r. De fait, un tour r comporte alors une seule phase de préparation qui est suivie par autant de phase de proposition que N_i peut en lancer avant d'être destitué. En conséquence, plusieurs instances de consensus peuvent donc être exécutées durant le même tour. Lorsque le leader élu reste stable (pas de défaillance, interactions avec les autres acteurs suffisamment synchrones), le chemin de communication est de longueur 4 et correspond à : auteur $\rightarrow$ coordinateurs (leader) $\rightarrow$ accepteurs $\rightarrow$ leader $\rightarrow$ auteurs.
L'optimisation $\mathbf{O_2}$ consiste à exécuter de façon anticipée la partie de la phase de proposition correspondant à la diffusion par le leader d'une requête et à sa réception par les accepteurs. L'objectif est de tirer profit du fait que, durant la plupart des instances de consensus, un seul auteur participe au consensus et donc une seule valeur initiale est disponible. Le leader qui ne connait aucune valeur initiale relative au consensus k adopte une valeur par défaut appelée ANY qu'il soumet lors de la phase de proposition. Tout accepteur recevant cette valeur fictive est alors autorisé à la remplacer par une vraie valeur initiale

directement reçue d'un auteur. Lorsque l'activation de $\mathbf{O_2}$ est un succès, le chemin de communication est de longueur 3 : auteur $\rightarrow$ accepteurs $\rightarrow$ leader $\rightarrow$ auteurs. Sans $\mathbf{O_2}$, une seule valeur initiale est soumise durant une phase de proposition, et donc tous les accepteurs qui acceptent une valeur durant cette phase adoptent nécessairement la même valeur. Avec $\mathbf{O_2}$, cette propriété n'est plus toujours vérifiée dès lors que deux auteurs fournissent deux valeurs initiales distinctes durant l'instance k. Ceci a deux conséquences majeures. D'une part, même lorsque les conditions sont favorables, la définition d'un quorum est plus contraignante : le leader doit collecter plus d'acceptations [4, 7]. Dans notre implémentation, le taux de retours attendus passe de $\lceil n/2 \rceil$ à $\lceil 3n/4 \rceil$. D'autre part, l'optimisation est un échec lorsque les retours collectés par le leader font référence à plus d'une valeur (occurrence d'une collision). Un recouvrement est alors nécessaire : un nouveau tour est démarré par le coordinateur (via l'exécution d'une phase de préparation) et une nouvelle phase de proposition est lancée mais cette fois, sans activer $\mathbf{O_2}$.

Les auteurs externes qui ne savent pas si l'optimisation est activée ou pas, doivent diffuser systématiquement leur valeur initiale à l'ensemble des coordinateurs et à l'ensemble des accepteurs. Dans le protocole Paxos-MIC, l'activation de $\mathbf{O_2}$ est gérée de la façon suivante. Lorsque l'exécution de l'instance de consensus $k-1$ se termine, un leader qui dispose déjà d'une proposition concernant le prochain consensus k ne déclenche pas $\mathbf{O_2}$ et effectue un consensus en n'utilisant que $\mathbf{O_1}$: nous dirons dans ce cas que les consensus $k-1$ et k s'enchainent. Si au contraire, aucune proposition n'est disponible, le leader est temporairement inactif et il évalue alors un test de déclenchement pour déterminer si $\mathbf{O_2}$ doit être activé ou pas. Si le test est faux (ou plus généralement si il ne devient pas vrai avant qu'une proposition parvienne au leader), le consensus k s'exécute en n'utilisant que $\mathbf{O_1}$: nous dirons dans ce cas que $\mathbf{O_2}$ est non activée (par choix).

3. Différents contextes d'expérimentation

Dans la suite de cette étude, nous considérons que $\mathbf{O_1}$ est toujours utilisée, soit seule, soit en même temps que $\mathbf{O_2}$. L'objectif de ces deux optimisations est de réduire le temps qui s'écoule entre l'instant où un auteur suggère une valeur initiale et l'instant où il prend connaissance de la valeur de décision. Les gains escomptés ont jusqu'à présent été exprimés non pas en temps mais en terme de nombre d'étapes de communication. Pour juger de l'intérêt de $\mathbf{O_2}$, nous avons procédé tout d'abord à une évaluation des performances du protocole Paxos-MIC. Pour cela, une mise en oeuvre du protocole a été effectuée en Java. Les expérimentations ont ensuite été conduites en utilisant la grille de calcul GRID-5000.

Nous avons d'abord cherché à analyser le comportement du protocole indépendamment de l'application qui l'utilise. Ces évaluations ont montré que l'intérêt de $\mathbf{O_2}$ dépend significativement du contexte d'exécution du consensus. Nous tenons compte ici de cette dépendance en considérant trois contextes dénotés respectivement C_{RR05}, C_{RR11} et C_{OR05}. La signification de cette notation est la suivante : dans le contexte C_{XRn}, les auteurs sont placés sur un site X (lettre R pour Rennes et O pour Orsay) alors que les acteurs (n accepteurs et $\lceil n/2 \rceil$ coordinateurs) sont placés sur le site R (Rennes). Les contextes C_{RR05} et C_{RR11} se distinguent donc uniquement de par le nombre d'acteurs avec un placement des auteurs et acteurs sur l'unique site de Rennes. Dans le cas de C_{OR05}, l'auteur est placé sur le site de Orsay et les acteurs sont tous placés sur le site de Rennes.

Les défaillances sont rares. De plus, la stabilité d'un leader est généralement assurée sur de longues périodes puisque, dans nos trois contextes, tous nos acteurs s'exécutent sur un même site. Dans ces circonstances qui sont les plus fréquentes, nous avons procédé à l'exécution de plus de 400 consensus pour déterminer la durée moyenne d'un consensus lorsque :

- $\mathbf{O_2}$ n'est pas activée. La durée ainsi mesurée est appelée durée *normale* et notée DN.
- $\mathbf{O_2}$ est activée et aucune collision ne se produit. La durée du consensus est *réduite* et est notée DR.
- $\mathbf{O_2}$ est activée et des collisions (provoquées) se produisent systématiquement. DD est alors la durée *dégradée* qui intègre le coût du recouvrement.

La table 1 donne les durées observées dans les trois contextes étudiés. Dans ces trois cas, DR $\leq$ DN $\leq$ DD. Sur la figure 1, nous représentons à nouveau les différentes valeurs de la table 1. A chaque contexte est associé une rangée de trois points qui, en allant de la gauche vers la droite, représentent la valeur de DR en fonction de DN, la valeur de DN et enfin la valeur de DD en fonction de DN. Sur l'axe horizontal sont représentées les valeurs (en ms) de DN - 1 ms, DN, ..., DN + 3 ms. Bien évidemment, cette représentation masque le fait que la valeur de DN varie d'un contexte à un autre : ainsi la valeur

	DR	DN	DD
Contexte C_{RR05}	1.312 ms	1.915 ms	3.149 ms
Contexte C_{RR11}	1.775 ms	2.121 ms	5.175 ms
Contexte C_{OR05}	9.399 ms	9.741 ms	11.505 ms

TABLE 1 – Durées moyennes (Réduites/Normales/Dégradées) en fonction des contextes

DN calculée dans le contexte C_{OR05} est 5 fois plus élevée que celle observée dans le contexte C_{RR05}. Au travers de la figure 1, nous voulons essentiellement mettre l'accent sur la variation du ratio entre un gain (DN - DR) et une perte (DD - DN). Dans le contexte C_{RR05}, une perte peut être compensée par deux gains. Par contre ce ratio est presque égal à 9 dans le contexte C_{RR11}.

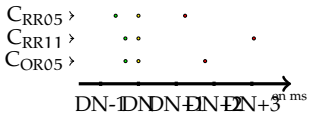

FIGURE 1 – Gains et pertes en fonction de DN selon les contextes

Le déclenchement de $\mathbf{O_2}$ étant un pari sur l'avenir, son activation peut être faite à tort ou à raison. Deux types d'erreur sont possibles :

- Erreur 1 (excès de pessimisme) : $\mathbf{O_2}$ n'est pas déclenchée alors qu'elle aurait du l'être. Dans ce cas, le gain possible DN - DR n'est pas réalisé. Dans le pire cas (contexte C_{RR05}), le retard engendré est de 0,6 ms environ.
- Erreur 2 (excès d'optimisme) : $\mathbf{O_2}$ est déclenchée alors qu'elle n'aurait pas du l'être. Dans ce cas, un surcoût de DD - DN vient retarder la prise de décision. Dans le pire cas (contexte C_{RR11}), le retard engendré est supérieur à 3 ms.

En résumé, la figure 2 identifie les deux conditions (sous la forme de deux losanges) qui doivent être satisfaites pour que $\mathbf{O_2}$ soit déclenchée. Premièrement, au moment où s'achève le consensus $k-1$, aucune proposition concernant le consensus k ne doit être à la disposition du leader : $\mathbf{O_2}$ ne peut être activée que durant une période d'inactivité. Deuxièmement, le test de déclenchement qui est évalué durant la phase d'inactivité doit être satisfait avant qu'une proposition ne parvienne au leader. La figure fait apparaitre les différents pourcentages que nous serons amenés à évaluer et à commenter durant nos analyses. En particulier, les pourcentages d'enchainement et d'activation apportent une information sur la fréquence d'utilisation de $\mathbf{O_2}$ tandis que les deux pourcentages d'erreur renseignent sur la qualité des prédictions. L'ensemble de ces pourcentages donne une image des capacités d'adaptation des différentes solutions.

Lors de l'instance de consensus k, le fait qu'une valeur soit disponible au moment où se termine le consensus $k-1$ dépend d'une part de la date à laquelle une première proposition concernant le consensus k est disponible et d'autre part de la date à laquelle le consensus $k-1$ se termine. De fait, le pourcentage d'enchainement dépend à la fois du contexte (puisque les durées moyennes de consensus sont différentes dans les trois contextes que nous avons retenus) et des activations - à raison ou à tort - de $\mathbf{O_2}$ au cours des consensus précédents (puisque la durée des consensus $k-1$, $k-2$, ... peut être réduite, normale ou dégradée).

4. Définition des différents tests évalués

Nous identifions ici deux tests irréalistes qui sont par nature impossibles à implémenter et cinq tests réalistes (trois statiques et deux dynamiques).

Tests irréalistes : le test *Parfait* est défini comme étant un test qui permet une prédiction parfaite : les deux taux d'erreur sont égaux à 0%. Par définition, les durées DR et DN sont alors les seules durées

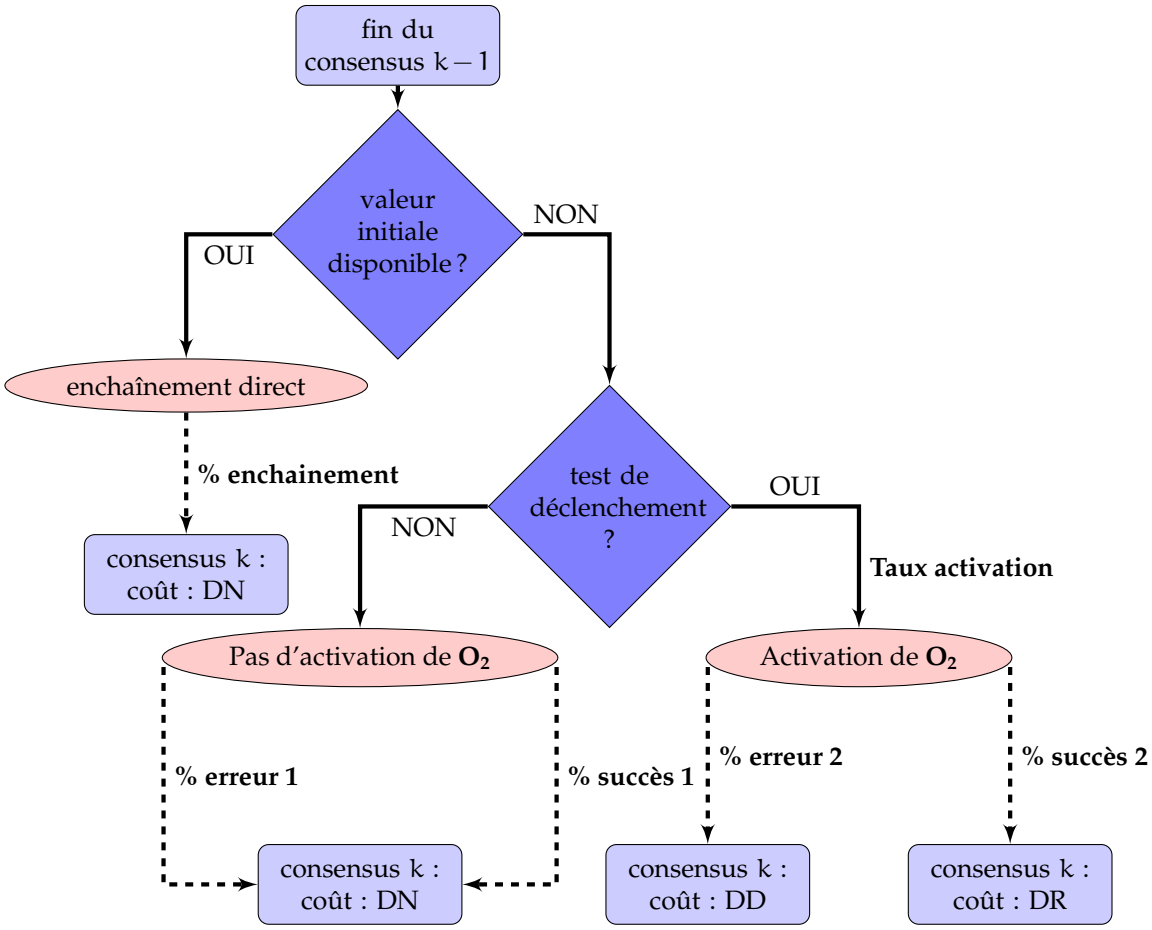

FIGURE 2 – Durées observées, conditions de déclenchement et erreurs

observables. Inversement, le test *Pire* est défini comme un test qui n'engendre que des prédictions erronées : les deux taux de succès sont égaux à 0%. Les seules durées observables sont DN et DD. Les tests irréalistes nous servent par la suite de points de référence lors de l'analyse des autres tests.

Tests réalistes statiques : le test *Toujours* déclenche l'optimisation $\mathbf{O_2}$ dès qu'il y a une période d'inactivité. Le taux d'activation est égal à 100% et les trois durées sont observables. Inversement, le test *Jamais* rend impossible l'activation de $\mathbf{O_2}$. Le test *Aléatoire* s'appuie sur une loi binomiale pour engendrer aléatoirement un taux d'activation proche de 80%.

Tests réalistes dynamiques : Le test *Temps* permet de déclencher l'activation de $\mathbf{O_2}$ lorsque plus de Δ_1 ms se sont écoulées depuis la dernière activation de consensus. Bien entendu plus la valeur de Δ_1 augmente, plus le taux d'activation diminue. Le test *Résultat* déclenche l'activation de $\mathbf{O_2}$ en fonction des erreurs potentiellement observées lors des consensus précédents. Nous choisissons de ne pas activer l'optimisation si au moins une erreur de type 2 (*i.e.*, une erreur engendrant une durée dégradée) a été observée durant les deux derniers consensus ($k-1$ et $k-2$).

5. Une analyse fondée sur une trace réelle d'activation

Dans le cadre de travaux sur la sécurisation d'un serveur Web, nous avons proposé une solution qui permet d'assurer une évolution cohérente des différents serveurs utilisés [11]. La solution repose sur l'utilisation d'une primitive de diffusion atomique dont la mise en oeuvre est fondée sur un protocole de consensus. Dans le cadre de cette activité, nous avions archivé l'ensemble des requêtes http adressées au serveur Web d'une grande école d'ingénieur durant une période de quinze jours consécutifs.

Nous réutilisons aujourd'hui ce log qui correspond donc à une trace réelle d'activation du protocole de consensus : à chaque arrivée de requête, le (ou les) receveurs agissent en tant qu'auteur(s) en proposant la requête reçue comme valeur initiale durant le prochain consensus. Les arrivées concomitantes de requêtes distinctes sont sources de collisions potentielles.
La connaissance de cette trace complète nous permet de faire une analyse aussi bien dans le cas des tests réalistes que dans le cas des tests irréalistes. Nous exploitons les résultats expérimentaux obtenus (durées moyennes de consensus DR, DN et DD) afin de déterminer les pourcentages d'enchainement et d'activation. En ce qui concerne les pourcentages d'erreur, notre analyse est résolument pessimiste puisque nous surestimons le risque de collision. Pour une requête R_a, nous considérons que son ordonnancement durant un consensus k (avec $\mathbf{O_2}$ activé) peut se faire sans qu'aucune collision ne se produise à condition que la prochaine requête R_b parvienne après un laps de temps supérieur à la durée réduite DR caractérisant le contexte d'exécution. Ce choix conduit très clairement à une augmentation du nombre de collisions observées dans notre analyse. Notre objectif étant d'évaluer l'intérêt de l'optimisation et la fiabilité des tests de prédiction, cette majoration (qui reste cependant dans des proportions raisonnables dans le cas de la trace utilisée) ne remet pas en cause nos conclusions.

Contexte C_{RR05}	Parfait	Pire	Toujours	Jamais	Aléatoire	Temps	Résultat
Pourcentage d'enchainement	2,95%	3,41%	3,00%	3,36%	3,07%	3,04%	3,02%
Pourcentage d'erreur 1	0%	94,24%	0%	94,28%	18,94%	5,27%	1,62%
Pourcentage de succès 1	2,39%	0%	0%	2,36%	0,48%	0,25%	0,20%
Taux d'activation	94,66%	2,35%	97%	0%	77,51%	91,44%	95,17%
Pourcentage d'erreur 2	0%	2,35%	2,38%	0%	1,89%	2,14%	2,18%
Pourcentage de succès 2	94,66%	0%	94,62%	0%	75,61%	89,30%	92,98%
Contexte C_{RR11}	Parfait	Pire	Toujours	Jamais	Aléatoire	Temps	Résultat
Pourcentage d'enchainement	3,31%	3,69%	3,43%	3,57%	3,46%	3,45%	3,43%
Pourcentage d'erreur 1	0%	93,66%	0%	93,76%	18,79%	4,87%	1,76%
Pourcentage de succès 1	2,67%	0%	0%	2,67%	0,53%	0,24%	0,22%
Taux d'activation	94,02%	2,65%	96,57%	0%	77,23%	91,44%	94,59%
Pourcentage d'erreur 2	0%	2,65%	2,65%	0%	2,13%	2,41%	2,43%
Pourcentage de succès 2	94,02%	0%	93,91%	0%	75,10%	89,02%	92,16%
Contexte C_{OR05}	Parfait	Pire	Toujours	Jamais	Aléatoire	Temps	Résultat
Pourcentage d'enchainement	8,63%	8,83%	8,69%	8,77%	8,71%	8,69%	8,69%
Pourcentage d'erreur 1	0%	84,78%	0%	84,83%	17,01%	0,24%	3,67%
Pourcentage de succès 1	6,42%	0%	0%	6,40%	1,28%	0,03%	0,78%
Taux d'activation	84,96%	6,38%	91,31%	0%	73,00%	91,04%	86,85%
Pourcentage d'erreur 2	0%	6,38%	6,40%	0%	5,12%	6,37%	5,62%
Pourcentage de succès 2	84,96%	0%	84,91%	0%	67,88%	84,67%	81,23%

TABLE 2 – Pourcentages d'erreurs et de succès calculés en fonction des contextes et des tests

La table 2 synthétise les résultats obtenus. Dans le cas du test *Temps* la valeur de Δ_1 est fixée à 10 ms. Les enseignements majeurs sont les suivants. Alors que nous avons surestimé le risque d'erreur, l'intérêt de $\mathbf{O_2}$ est très clair : les meilleurs résultats sont d'ailleurs obtenus avec le test *Toujours*. Les tests dynamiques se comportent néanmoins relativement bien et surpassent très nettement le test *Jamais*. Sur la trace complète on observe que le pourcentage d'enchainement reste inférieur à 5% dans les deux premiers contextes, mais avoisine les 10% dans le cas où les auteurs sont distants. En ce qui concerne les erreurs, les erreurs de type « excès d'optimisme »(erreur 2) sont relativement peu nombreuses et, de ce point de vue, les tests dynamiques ont tendance à améliorer légèrement la prédiction. Inversement, l'excès de pessimisme est rarement justifié (le pourcentage de succès 1 est très inférieur au pourcentage d'erreur 1). Le test *Résultat* permet d'atteindre la meilleure prédiction (avec cependant un rapport de 8 non activation injustifiées pour une non activation justifiée). La table 3 présente les gains en temps obtenus. A nouveau, le test *Résultat* semble offrir un bon compromis. Tout les tests réalistes donnent des valeurs supérieures à 60% à l'exception du test *Jamais* qui reste en dessous de 30%. L'intérêt de $\mathbf{O_2}$ dans ce scénario réel est clairement démontré.
Nous avons procédé aux mêmes analyses en considérant cette fois une portion de la trace (10 heures) correspondant à une période de forte activité. On observe alors une augmentation relativement importante

	Toujours	Jamais	Aléatoire	Temps	Résultat
C_{RR05}	95,06%	4,84%	76,96%	90,22%	93,82%
C_{RR11}	79,96%	19,90%	67,89%	77,58%	80,11%
C_{OR05}	71,94%	27,93%	63,12%	71,88%	72,25%

TABLE 3 – Gain en temps par rapport à Parfait (100%) et Pire (0%)

du pourcentage d'enchainement qui dans le cas du contexte C_{OR05} passe d'environ 9% à environ 17%. Le mécanisme d'enchainement des consensus joue donc un rôle important dans le processus d'adaptation. Par ailleurs, ce nouveau scénario qui conduit à plus de collisions rend le test *Toujours* moins intéressant tandis que le test *Jamais* redevient plus adapté. Les tests dynamiques démontrent tout leur intérêt : ces tests s'adaptent au contexte et à défaut d'offrir une prévision fiable, ils permettent d'obtenir un pourcentage d'activation adapté au contexte. Le test *Résultat* est à nouveau celui qui offre les meilleurs résultats.

6. Conclusion

Nous avons observé, dans le cas d'une application réelle, que le fait de s'appuyer sur la connaissance du passé récent pour tenter de prédire le risque de collision dans un avenir proche est une stratégie intéressante qui permet de tirer profit de l'optimisation $\mathbf{O_2}$ sans nécessiter une connaissance précise du comportement de l'application. Le test *Résultat* donne de bons résultats quelle que soit l'intensité du trafic de requêtes http. Une autre piste intéressante consiste à combiner l'utilisation des tests avec une phase d'apprentissage. L'analyse que nous avons menée peut ainsi permettre, en fonction de la date (jour ouvré ou week-end) et de l'heure (nuit ou jour) de choisir statiquement le meilleur test (statique ou dynamique) pour une période donnée. Les premières expérimentations menées dans ce sens (analyse du log sur 15 nuits et 15 jours) confirme l'intérêt de cette approche. Nous envisageons également de valider notre analyse au travers d'une exécution réelle et complète de la trace : les collisions se produiront alors de façon naturelle avec un taux certainement plus faible que celui que nous avons sur-estimé.

Remerciements
Les expériences présentées dans cet article ont été réalisées en utilisant la plateforme expérimentale Grid'5000, issue de l'Action de Développement Technologique (ADT) Aladdin pour l'INRIA, avec le support du CNRS, de RENATER, de plusieurs Universités et autres contributeurs (`https://www.grid5000.fr`).

Bibliographie

1. Fischer (M. J.), Lynch (N. A.) et Paterson (M. S.). – Impossibility of Distributed Consensus with One Faulty Process. – Journal of the ACM, 32(2) :374-382, 1985.
2. Lamport (L.). – The Part-Time Parliament. – ACM Transaction on Computer Systems, 16(2) :133-169, 1998.
3. Lamport (L.). – Paxos Made Simple. – ACM SIGACT News, 32(4) :51-58, 2001.
4. Guerraoui (R.) et Vukolic (M.). – Refined Quorum Systems. – Distributed Computing, pages 1-42, 2010.
5. Boichat (R.), Dutta (P.), Frolund (S.) et Guerraoui (R.). – Deconstructing Paxos. – ACM SIGACT News, 34(1) :47-67, 2003.
6. Martin (J. P.) et Alvisi (L.). – Fast Byzantine Consensus. – Proc. of the Int. Conference on Dependable Systems and Networks, pages 402-411, 2005.
7. Lamport (L.). – Fast Paxos. – Distributed Computing, 19(2) :79-103, 2006.
8. Charron-Bost (B.) et Schiper (A.). – Improving Fast Paxos : being optimistic with no overhead. – Proc. of the 12th Pacific Rim International Symposium on Dependable Computing, pages 287-295, 2006.
9. Dobre (D.), Majuntke (M.), Serafini (M.) et Suri (N.). – Hp : Hybrid Paxos for WANs. – Proc. of the European Dependable Computing Conference, pages 117-126, 2010.
10. Hurfin (M.), Moise (I.) et Le Narzul (J.-P.). – An Adaptive Fast Paxos for Making Quick Everlasting Decisions. – Proc. of the Int. Conference on Advanced Information Networking and Applications (AINA), 2011.
11. Hurfin (M.), Le Narzul (J.P.), Majorczyk (F.), Mé (L.), Saidane (A.), Totel (E.) et Tronel (F.). – A Dependable Intrusion Detection Architecture Based on Agreement Services. – Proc. of the 8th Int. Symposium on Stabilization Safety and Security, pages 378-394, 2006.