Distribution Large Echelle d'un Algorithme Financier de Contrôle Stochastique

Constantinos Makassikis

RenPar'18 / SympA'2008 / CFSE'6
Fribourg, Suisse, du 11 au 13 février 2008

Distribution Large Échelle d'un Algorithme Financier de Contrôle Stochastique

Constantinos Makassikis*
SUPELEC, IMS research group, 2 rue Edouard Belin, 57070 Metz, France,
LORIA, ALGORILLE project team, BP 239, 54506 Vandoeuvre-lès-Nancy, France,
Constantinos.Makassikis@supelec.fr

Résumé
Face à la demande croissante en termes de puissance de calcul que connaît le monde des calculs financiers, les solutions fondées sur le parallélisme deviennent incontournables. De nombreuses applications de calculs financiers se décomposent simplement en tâches indépendantes et donnent lieu à du *bag of task*, mais certaines applications ne se prêtent pas à ce type de décomposition. Dans ce papier, nous présentons une parallélisation originale de l'algorithme d'une telle application de calcul financier appliquée à la valorisation d'un actif de stockage de gaz et réalisée dans le cadre du projet ANR GCPMF. Les expérimentations à large échelle que nous avons menées sur un cluster de PCs de Grid'5000 (jusqu'à 128 processeurs) et sur le supercalculateur Blue Gene/L d'EDF R&D (jusqu'à 1024 processeurs) affichent de très bonnes performances et permettent de comparer ces systèmes non seulement à partir leur temps d'exécution mais aussi en termes de fiabilité expérimentale.

Mots-clés : Contrôle stochastique, distribution large échelle, clusters de PCs, Blue Gene/L

1. Introduction

1.1. Contexte financier

Un actif de stockage de gaz est habituellement constitué d'une cavité destinée à contenir le gaz ainsi que de compresseurs permettant d'effectuer des injections, des sous-tirages ou simplement de maintenir le gaz sous pression dans la cavité. En temps normal, le propriétaire remplit son actif de stockage pendant les périodes où les prix du gaz sont bas pour répondre aux demandes des clients ou pour le revendre ultérieurement sur les marchés. Éventuellement, il peut envisager de louer une partie de sa capacité de stockage à une autre entité.

Les prix du gaz connaissent des fluctuations issues principalement de la modification de la demande. À cause de l'inélasticité entre la production et la demande, les prix du gaz sont, par exemple, plus élevés en hiver qu'en été. Le propriétaire d'un actif de stockage de gaz peut alors profiter de cette dynamique des prix en arbitrant entre les différentiels temporels des prix du gaz et ainsi valoriser son actif. La détermination du prix de location tient compte de l'opportunité d'arbitrage mais elle est également soumise à des contraintes. D'une part, il existe des contraintes physiques liées à la cavité et la manière de stocker le gaz. Par exemple, une injection (resp. un

* Ce travail est le fruit de la collaboration avec l'équipe Osiris de EDF R&D dans le cadre du projet ANR GCPMF.

sous-tirage) est d'autant plus difficile à réaliser (et donc coûteuse) que la cavité est remplie (resp. vide). D'autre part, le contrat de location peut imposer une valeur de stock minimale et le volume loué peut être variable au cours du temps et soumis à des restrictions en ce qui concerne la capacité journalière d'injection (resp. de sous-tirage) autorisée.
La valorisation fait appel à des algorithmes de contrôle stochastique et à des modèles de prix du gaz. De nombreux travaux de recherche récents dans le domaine de la valorisation des actifs de stockage de gaz (cf. [1, 2]) ont conduit à l'élaboration de ces modèles. Cependant, les besoins de ces modèles en termes de puissance de calcul et de consommation mémoire peuvent aisément surpasser les capacités des machines monoprocesseur actuelles ce qui rend très difficile leur utilisation en environnement industriel où ils sont, de surcroît, soumis à des contraintes de temps. Dès lors, la conception et l'implantation d'algorithmes parallèles faisant intervenir ces modèles devient incontournable. C'est l'un des objectifs du projet ANR GCPMF dans lequel s'inscrivent ces recherches.

1.2. Défis informatiques

D'un point de vue informatique, l'algorithme auquel on s'intéresse est un algorithme itératif de *programmation dynamique stochastique* qui est complexe à paralléliser et ce, malgré la présence de parallélisme (cf. Section 2.1). En effet, les calculs à chaque étape dépendent des résultats de l'étape précédente, et les données à calculer varient régulièrement. Ces contraintes conduisent à redistribuer les données et les calculs à chaque étape. Par ailleurs, les données requises par chaque processeur doivent être soigneusement identifiées afin de n'avoir à redistribuer et à stocker qu'un minimum de données sur chaque processeur. Cette stratégie est nécessaire pour pouvoir exécuter des problèmes de très grande taille sur de nombreux processeurs. Une diffusion (broadcast) à chaque étape de toutes les données sur tous les processeurs serait facile à implanter mais nécessiterait trop de mémoire sur chaque processeur.
Notre papier est organisé comme suit. Après la section 2 où est présenté en détail l'algorithme distribué que nous avons conçu, la section 3 se concentre sur l'analyse des performances obtenues par l'algorithme. Enfin, la section 4 résume notre travail de recherche et présente des perspectives.

2. Distribution de l'algorithme

2.1. Algorithme séquentiel et difficultés de distribution

La figure 1 présente l'algorithme séquentiel qui est utilisé pour effectuer la valorisation. Cet algorithme consiste à balayer toute la période de valorisation de la date finale à la date initiale et à déterminer, à chaque étape de ce parcours, l'action la plus intéressante à entreprendre : injecter, sous-tirer ou ne rien faire. La décision prise dépend d'une part des niveaux de stocks dont les valeurs sont conditonnées par différentes contraintes (cf. section 1.1), d'autre part de la valeur des prix renseignée par le modèle de prix utilisé et finalement des décisions et résultats du pas de temps précédent qui sont sauvegardés dans la structure de données *OldRes* à la fin de chaque itération en temps.
Une parallélisation au niveau de la boucle temporelle s'avère donc impossible. En revanche, une parallélisation de la boucle sur les niveaux de stocks est possible, mais nécessitera un échange complexe de données entre les processeurs à chaque pas de temps (cf. section 2.2).

2.2. Algorithme distribué
Stratégie de distribution

Afin de réaliser des accélérations importantes et permettre le passage à l'échelle, l'algorithme de la figure 1 a été parallélisé pour des architectures extensibles tels les clusters de PCs et les

```
For t := (N − 1)Δt to 0
    For c ∈ niveaux de stock admissibles
        For s ∈ valeurs de prix possibles
            NewRes[s, c] = calcul(OldRes, s, c)
    //Recopie pour le pas de temps suivant
    OldRes := NewRes
```

FIG. 1 – Algorithme séquentiel simplifié de contrôle stochastique.

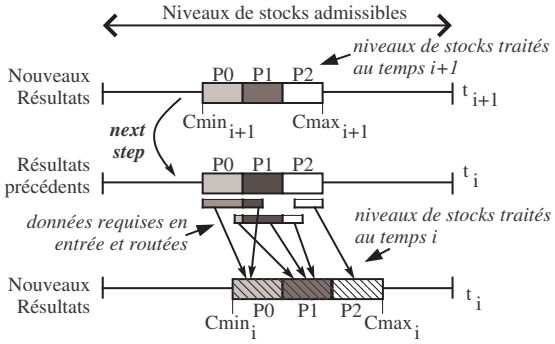

FIG. 2 – Exemple sur trois processeurs d'une redistribution de données optimisée.

supercalculateurs à mémoire partagée.
Comme le montre la figure 2, à l'étape $i+1$, chaque processeur effectue les calculs pour l'ensemble contigu des niveaux de stocks - compris entre $Cmin_{i+1}$ à $Cmax_{i+1}$ - qui lui ont été attribués. À l'étape i (étape suivante), la nouvelle boucle sur les niveaux de stock (de $Cmin_i$ à $Cmax_i$) est partagée entre les processeurs. Pour effectuer sa nouvelle tâche, chaque processeur a besoin de certains résultats de l'itération précédente dont il n'est pas nécessairement en possession. Ainsi, chaque processeur détermine d'une part les données en sa possession dont auront besoin les autres processeurs, et d'autre part les données qui lui manquent et sont détenues par d'autres processeurs ; ces informations sont alors utilisées pour dresser un plan de routage dans lequel figurent toutes les communications qu'il devra effectuer. Ce plan est finalement exécuté selon un schéma de communication préétabli.

Algorithme distribué détaillé

En suivant la stratégie décrite dans le paragraphe précédent, nous avons conçu l'algorithme distribué décrit dans la figure 3. La première étape de cet algorithme consiste à partager les calculs à t_n en utilisant des routines d'initialisation spécifiques. On rentre ensuite dans une boucle de n itérations (de t_{n-1} à t_0) qui est constituée de deux sous-étapes. La première consiste à *planifier et à exécuter l'échange des données*. La seconde correspond à la *nouvelle phase de calculs*.
Pendant la première sous-étape chaque processeur commence par déterminer la totalité du nouveau plan de partage des calculs et la nouvelle distribution des données. Les calculs impliqués par cette étape sont très simples et ne présentent donc aucun intérêt à être distribués. Ensuite chaque processeur établit son plan de routage et redimensionne ses tables locales conformément aux besoins de la nouvelle distribution des données. L'échange effectif des données s'effectue immédiatement après et repose sur des communications point-à-point.
La seconde sous-étape consiste en un calcul local à chaque processeur suivant l'algorithme de contrôle stochastique partiellement décrit dans la figure 1.

2.3. Implantations

L'algorithme présenté dans la section précédente a donné lieu à deux implantations. La première privilégiait la facilité d'utilisation en combinant Python avec du MPI et du C++. L'avantage de Python est de permettre de facilement paramétrer l'application et d'ajouter rapidement des modules tels que des modules de visualisation.
Quant à la deuxième implantation, elle vise la performance pure et a été écrite entièrement en

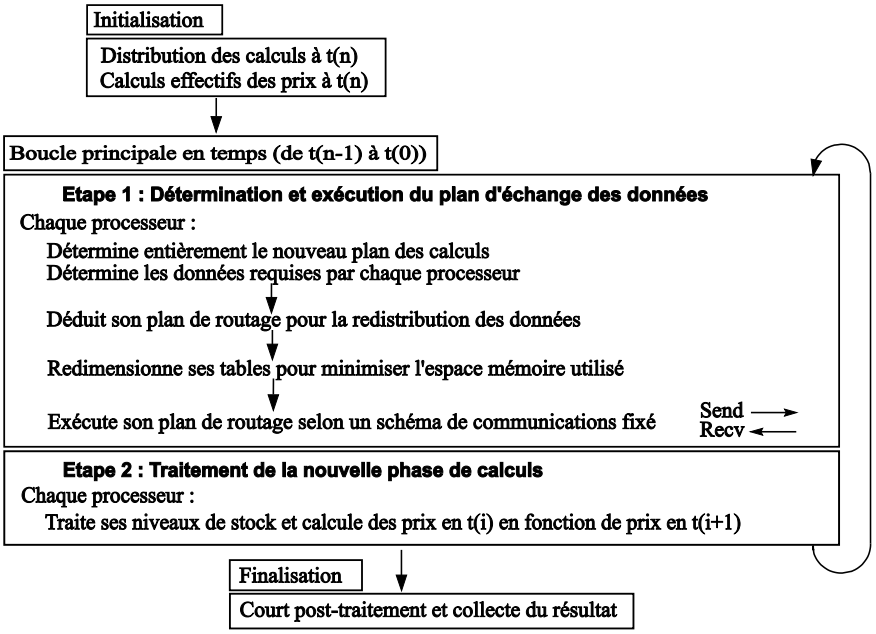

FIG. 3 – Étapes principales de notre algorithme de contrôle stochastique distribué.

C/C++ et MPI. Cette implantation offre le choix entre trois primitives MPI pour effectuer les communications : la primitive bloquante `MPI_Bsend()` ainsi que les primitives non-bloquantes `MPI_Ibsend()` et `MPI_Issend()`. La primitive `MPI_Send()` a rapidement été écartée à cause de son manque de portabilité entre implantations de MPI. Parmi les trois utilisées, les primitives `MPI_Ibsend()` et `MPI_Issend()`, en permettant plusieurs communications en parallèle (lorsque le réseau d'interconnexions le permet), se sont montrées les plus rapides. En échange d'un surcoût lié à la synchronisation inhérente à `MPI_Issend()`, cette dernière ne nécessite aucune allocation ni gestion explicite de buffers. Cette économie la rend très intéressante pour la distribution d'applications gourmandes en mémoire.

3. Expérimentations

3.1. Environnements et modèles de prix

L'évaluation des performances de notre algorithme parallèle a été réalisée sur trois architectures distribuées différentes et ce, en utilisant trois modèles de prix variés.

L'environnement d'expérimentation comprenait deux clusters de PCs et un supercalculateur :

- le *cluster de Pentium 4* de SUPELEC qui est composé de 32 PCs reliés par un réseau Gigabit-Ethernet bon marché : chaque PC comporte un processeur Pentium 4 à 3 GHz et 2 Go de mémoire ;
- le *cluster de bi-Opteron* de l'INRIA qui dispose de 72 PCs reliés par un réseau Gigabit-Ethernet de très bonne qualité : chaque PC comporte deux processeurs Opteron monocores à 2 GHz et 2 Go de mémoire ; ce cluster est situé sur le site de Sophia de la plate-forme française d'expérimentation Grid'5000 ;
- le *supercalculateur Blue Gene/L* d'EDF R&D doté de 4096 noeuds et d'un réseau Gigabit-Ethernet hyper spécialisé. Sa configuration par noeud est, de loin, la moins puissante parmi les trois architectures considerées : deux processeurs PowerPC à 700 MHz et 1 Go de RAM.

Les modèles de prix utilisés comprenaient :

- un modèle de prix *gaussien à un facteur* : c'est un modèle de référence qui est couramment

utilisé de part sa rapidité d'exécution ;
- un modèle de prix *normal inverse gaussien* qui est beaucoup plus lourd que le modèle gaussien en termes de calculs : ceci rend son utilisation moins fréquente ;
- un modèle de prix *gaussien à deux facteurs* : des trois modèles utilisés c'est de loin le modèle le plus lourd aussi bien en termes de calculs qu'en termes d'espace mémoire nécessaire à son exécution.

Plus de détails sur ces modèles mathématiques sont disponibles dans [3].
Avec la configuration des modèles prise pour les expériences, les zones de recouvrement sont constantes au sein d'un modèle ce qui conduit à un volume de données échangées constant par processeur et par itération. Les volumes de données par itération augmentent linéairement par rapport au nombre de processeurs et nous avons en moyenne sur 32 processeurs 1 Mo de données échangées pour les modèles gaussien et normal inverse gaussien contre 349 Mo pour le gaussien à deux facteurs.

3.2. Analyse des performances

Le cluster de bi-Opteron et le supercalculateur Blue Gene offrent la possibilité d'exécuter des applications selon deux configurations : soit en utilisant un processeur par noeud soit en utilisant deux processeurs par noeud. Ces deux configurations seront désignées par le suite respectivement par *mode monopro* et *mode bipro*.

Modèle gaussien à un facteur

Sur ce modèle, le supercalculateur Blue Gene se comporte aussi bien dans les deux modes. Ce n'est pas le cas du cluster de bi-Opteron dont les performances se dégradent considérablement en mode bipro à partir de 32 processeurs. Des mesures expérimentales ont montré que l'augmentation du nombre de processeurs accentuait de manière significative le temps passé sur les communications en mode bipro par rapport au mode monopro : il semblerait que l'accès à la carte réseau constitue une source d'engorgement pour les processeurs d'un même noeud. La suite se concentre sur l'analyse des performances en mode monopro présentes à la figure 4.
L'observation des courbes d'accélérations de la figure 4 rend compte d'une hyperaccélération entre 4 et 64 processeurs pour le supercalculateur Blue Gene et pour le cluster bi-Opteron que nous attribuons à une augmentation simultanée du nombre de processeurs et de la quantité totale de mémoire disponible. Sachant que les noeuds sur Blue Gene disposent de deux fois moins de mémoire que les noeuds du cluster bi-Opteron, il n'est pas surprenant de voir ce dernier hyperaccélérer davantage. Dans tous les cas, l'hyperaccélération tend à disparaître avec l'augmentation du nombre de processeurs et l'accélération maximale est obtenue par Blue Gene sur 512 processeurs. Le cluster de Pentium 4 ne réalise aucune hyperaccélération et sa courbe d'accélération croît lentement. Il s'avère donc qu'un réseau d'interconnexions très rapide est indispensable pour obtenir de bonnes performances sur ce modèle, mais dès lors un cluster de taille moyenne suffit pour en accélérer l'exécution (cf. section 3.3).
En dernier lieu, malgré quelques problèmes de passage à l'échelle que nous avons rencontrés, le meilleur temps séquentiel obtenu, proche de 15 min, a pu être réduit à $13 - 15$ s sur un cluster de PCs et sur un supercalculateur Blue Gene. Il s'agit donc d'une réelle amélioration pour les utilisateurs étant donné que c'est le modèle le plus utilisé.

Modèle normal inverse gaussien

Que ce soit en mode monopro ou bipro les performances obtenues par notre algorithme et son implantation sur le modèle normal inverse gaussien sont très bonnes. Sur la figure 5 où ont été

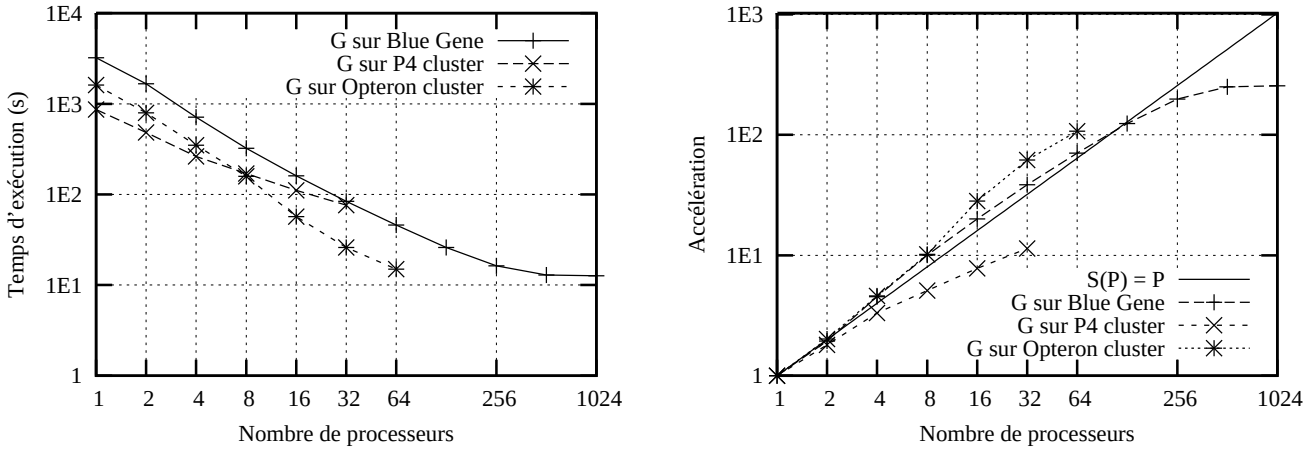

FIG. 4 – Temps d'exécution et accélérations (échelle logarithmique) avec le modèle gaussien sur trois architectures distribuées différentes en utilisant un seul processeur par noeud.

reportées les performances mesurées en utilisant le nombre maximal de processeurs par noeuds (i.e. : mode bipro), nous observons une parallélisation quasi-parfaite et ce, même sur le cluster bas de gamme à base de Pentium 4. Le meilleur temps d'exécution est réalisé sur Blue Gene avec 1024 processeurs. Cependant, le cluster de bi-Opteron réalise une performance similaire avec seulement 128 processeurs. Ainsi, une alternative intéressante pour exécuter le modèle normal inverse gaussien avec notre algorithme est un cluster de PCs de grande taille muni de processeurs puissants sans être nécessairement reliés par un réseau ultra rapide.
Finalement, le meilleur temps séquentiel qui est proche de $6h25$ et qui a été obtenu par un processeur Opteron, a pu être réduit à 3 *min* en utilisant 1024 processeurs de Blue Gene. Il en résulte que notre distribution, moyennant la disponibilité d'un nombre suffisant de processeurs, rend possible une utilisation beaucoup plus courante du modèle normal inverse gaussien.

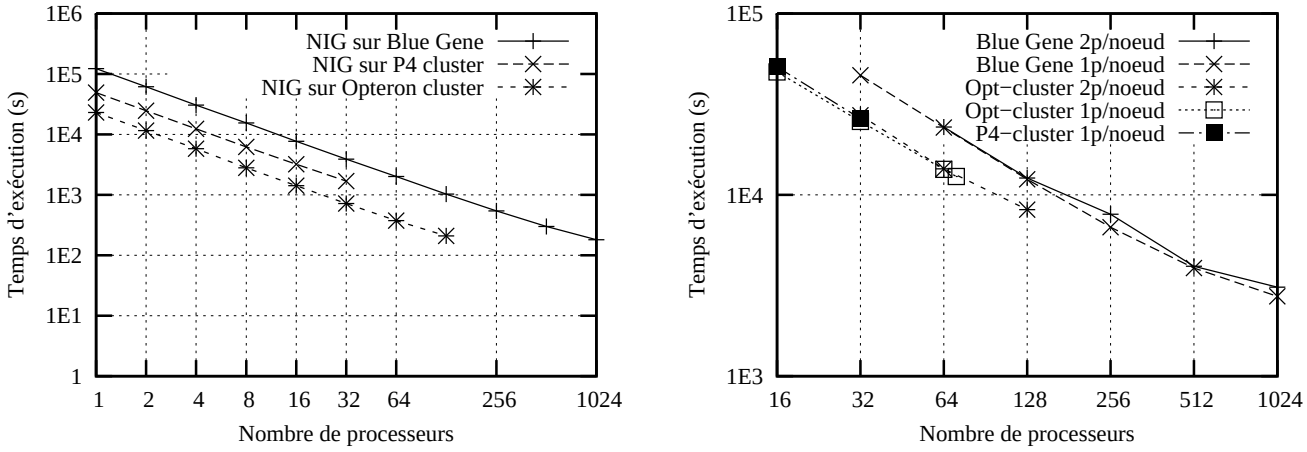

FIG. 5 – Temps d'exécution (échelle logarithmique) sur trois architectures distribuées différentes. À gauche : avec le *modèle normal inverse gaussien* en utilisant le nombre maximal de processeurs par noeud. À droite : avec le *modèle gaussien à deux facteurs* en utilisant le nombre minimal ou maximal de processeurs par noeud.

Modèle gaussien à deux facteurs

Ce modèle nécessite une puissance de calcul et un espace mémoire considérables en vue de s'exécuter dans des temps raisonnables/acceptables. Avec le jeu de paramètres courant et notre implantation qui utilise principalement deux tables - pour stocker les anciens et les nouveaux résultats - l'application aurait théoriquement besoin de 2×5895 Mo de memoire pour s'exécuter en séquentiel. En théorie, une distribution sur 8 noeuds nécessiterait 1474 Mo (i.e. : $(2 \times 5895)/8$) de RAM par noeud, donc 8 noeuds munis de 2 Go de RAM devraient supporter cette application. Cependant, en pratique, d'autres paramètres entrent en considération. Par exemple, la taille du noyau du système d'exploitation ne peut être négligée. Par ailleurs, à cause de la structuration de notre algorithme parallèle, la quantité de mémoire requise lors de la parallélisation est plus importante que dans le cas séquentiel. En l'occurrence, comme chaque processeur stocke les données influençant ses calculs, une légère réplication des données est possible au niveau des frontières de calculs entre processeurs. Il en résulte, qu'en pratique, cet algorithme nécessite au moins 10 processeurs dotés de 2 Go de mémoire chacun pour s'exécuter sans avoir recours au swap. Ainsi, selon le système, et sachant que l'on effectue nos tests avec des nombres de processeurs qui sont une puissance de 2, les courbes des temps d'exécution de la figure 5 débutent soit à 16 soit à 32 processeurs. Le petit cluster de Pentium-4 avec ses 32 noeuds monoprocesseurs équipés de 2 Go de mémoire a réussi à exécuter ce benchmark sur 16 et 32 processeurs. Avec une réduction du temps sur 32 processeurs d'un facteur deux par rapport au temps sur 16 processeurs, l'application semble réaliser un début de passage à l'échelle sur ce modeste cluster. Le cluster de bi-Opteron, avec 2 Go de memoire par noeud, réussi également à exécuter le modèle Gaussien à deux facteurs à partir de 16 noeuds en mode monopro. Les temps d'exécution affichés en mode bipro sont très similaires à ceux en mode monopro et le benchmark supporte très bien le passage à l'échelle jusqu'à 128 processeurs sur 64 noeuds et en utilisant deux processeurs par noeud. En ce qui concerne le supercalculateur Blue Gene, avec seulement 1 Go de mémoire par noeud, la mémoire de 32 noeuds est nécessaire. Les temps en mode bipro sont légèrement plus importants que ceux en mode monopro mais le ralentissement observé est relativement faible. Comparé aux clusters de PCs, le supercalculateur Blue Gene qui est doté de la configuration par noeud la moins puissante connaît des temps d'exécution sur 32 processeurs beaucoup plus importants. Cependant grâce à son architecture *ultrascalable* [4] qui allie un excellent réseau d'interconnexions avec de nombreux noeuds, le supercalculateur Blue Gene démontre sa supériorité en finissant loin en tête.

3.3. Vers la meilleure solution

Au cours de nos expérimentations le supercalculateur Blue Gene, avec ses milliers de processeurs, s'est avéré être la solution la plus intéressante pour des applications présentant un fort potentiel de passage à l'échelle. Dans d'autres cas, les clusters de PCs se présentent comme des alternatives très attrayantes étant donné leur bonnes performances et surtout leur prix modique à l'achat. Cependant, les clusters de PCs affichent deux inconvénients majeurs :

- ils ne sont pas fiables : plusieurs pannes sont venues perturber nos expériences sur les clusters de PCs, alors que rien de tel n'est survenu sur Blue Gene ;
- malgré un prix d'achat très faible comparé à un supercalculateur comme Blue Gene, le coût global d'un cluster de PCs peut devenir plus important. En effet, l'absence de fiabilité accroît les ressources humaines nécessaires pour les administrer. Par ailleurs, les applications destinées à y être exécutées ont parfois besoin d'algorithmes adaptés, plus longs à développer, afin de réaliser de bonnes performances sans pour autant disposer de réseaux d'interconnexions très efficaces.

Algo-rithme	Consommation		Système possible
	CPU	Mémoire	
G	Moyenne	Faible	cluster de PC, réseau Gigabit-Ethernet haut de gamme
NIG	Élevée	Faible	cluster de PC, réseau Gigabit-Ethernet bon marché
G-2f	Élevée	Élevée	supercalculateur Blue Gene

TAB. 1 – Solution matérielle la plus adaptée à chaque modèle ?

D'un point de vue technique, nos expériences démontrent la faisabilité d'obtenir de bonnes performances pour les modèles gaussien à un facteur et normal inverse gaussien sur des clusters de PCs. Cependant, d'un point de vue industriel, notamment au sein des laboratoires d'EDF R&D, le supercalculateur Blue Gene est apparu comme une solution très performante et fiable pour tous les modèles, et finalement très rentable.

4. Conclusion et perspectives

La distribution d'un algorithme de contrôle stochastique utilisé pour la valorisation d'actifs de stockage de gaz qui a été présentée ainsi que les expériences menées sur trois architectures distribuées différentes montrent qu'il est possible d'accélérer et de passer à l'échelle des calculs de contrôle stochastique. La stratégie de parallélisation adoptée, qui consiste à optimiser aussi bien la taille des données sur chaque processeur que la taille et le nombre de messages au prix de schémas de communication complexes, s'est averée convenir pour l'exécution de modèles aux besoins variés. En particulier, elle facilite l'étude d'un modèle très lourd en termes de calculs et de mémoire en rendant possible son exécution en des temps raisonnables sur des clusters de PCs et sur un supercalculateur Blue Gene. En dernier lieu, les performances obtenues présentent les clusters de PCs comme des alternatives intéressantes aux supercalculateurs sur certaines exécutions mais à condition qu'ils soient pourvus de mécanismes de tolérance aux pannes. C'est un des sujets de recherche que nous étudions actuellement et que nous comptons appliquer à notre algorithme distribué de contrôle stochastique.

Remerciements

Ce travail de recherche fait partie du projet ANR-CICG GCPMF qui est supporté par l'ANR et la Région Lorraine. Nous remercions également Xavier Warin de l'équipe Osiris à EDF R&D pour ses développements et son aide sur le contrôle stochastique, et la réalisation des mesures de performances sur le supercalculateur Blue Gene.

Bibliographie

1. P. Jaillet, E. I. Ronn, and S. Tompaidis, "Valuation of Commodity-Based Swing Options," *Manage. Sci.*, vol. 50, no. 7, pp. 909–921, 2004.
2. C. Barrera-Esteve, F. Bergeret, E. Gobet, and al, "Numerical methods for the pricing of Swing options : a stochasticapproach," *Methodology and Computing in Applied Probability*, vol. 8, no. 4, pp. 517–540, 2006.
3. C. Makassikis, X. Warin, and S. Vialle, "Distribution of a Stochastic Control Algorithm Applied to Gas Storage Valuation," *The 7th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT '07)*, 2007.
4. S. Chari, "Breakthrough Advantage in Computational Fluid Dynamics with the IBM System Blue Gene Solution," *Vetrei Inc. White Paper*, September 2006.