Rapport

Sur l'article : Arbitration Policies for On-Demand User-Level I/O Forwarding on HPC Platforms[2]

GOEDEFROIT Charles

5 décembre 2022

1 L'objectif de l'article

L'objectif de l'article est d'améliorer l'utilisation de la bande passante entre les *noeuds de calculs* et les *noeuds de stockage des données*, que j'appellerai *noeuds I/O* dans le reste de ce rapport. Plus précisément, l'objectif est de permettre le changement dynamique des politiques d'accès aux *noeuds I/O*. Pour cela, l'article propose :

— une implémentation qui permet ce changement en fonction des patterns d'accès aux *noeuds I/O*.
— ainsi qu'une politique basé sur le problème du sac à dos à choix multiple (*Multiple-Choice Knapsack problem*).

Il propose aussi une solution qui permet d'utiliser dynamiquement différentes politiques d'allocation de *noeuds I/O forwarding*.

2 Le context de l'article

Cet article se place dans le contexte ou les supercalculateurs font de plus en plus d'accès aux données. Cette augmentation est due à plusieurs facteurs comme :

— l'augmentation de la puissance de calcul.
— l'augmentation des quantités de données a traité.
— la variété des types d'applications à accès hétérogène aux données, l'intelligence artificielle, le bigData...

Pour répondre au besoin de puissance croissante, on augmente le nombre de *noeuds de calculs* ce qui a pour effet d'augmenter le nombre de requêtes au système de fichier parallèle (*PFS*). Ce système fini par ne plus pouvoir traité autant de requêtes, car tous les *noeuds I/O* se retrouvent saturés. Pour réglé ce problème les *PFS* utilise la technique du *I/O forwarding* qui diminue le nombre de noeuds du *PFS* accessible par les *noeuds de calculs*.

2.1 La technique du *I/O forwarding*

Cette technique est très utilisée pour le calcul haut performance, notamment par des machines du top 500 [1]. Elle permet d'augmenter les performances globales en diminuant la concentration des requêtes sur les *noeuds I/O*. Pour cela, elle évite les accès directs aux *noeuds I/O* en se plaçant entre les *noeuds de calculs* et les *noeuds I/O*. Concrètement, un groupe de *noeuds I/O* reçoit les requêtes et appliquent des traitements avant de les transmettre aux bons *noeuds I/O*. Cette technique est transparente pour les utilisateurs, elle cherche à appliquer une répartition uniforme entre les noeuds et permet le contrôle des demandes d'accès aux données ce qui permet l'application de technique d'optimisation. Ces techniques peuvent être de différentes natures comme l'agrégation des requêtes (en combinant plusieurs petites requêtes) ou la planification des requêtes.

Habituellement les *noeuds I/O* qui s'occupent de la réception des requêtes sont assigné statiquement à un *noeud de calculs* et celle-ci peut dépendre de la topologie du réseau, du nombre de *noeuds I/O*... Cette assignation ne correspond pas tout le temps au besoin de l'application qui peut ne pas utiliser toute

1. November 2020 TOP500 : https ://www.top500.org/lists/2020/06/.

la bande passante ou être ralentie par celle-ci. L'assignation statique amène donc à une mauvaise utilisation des ressources. Dans la suite du rapport, j'appellerai les *noeuds I/O* qui s'occupent de la réception les *noeuds I/O forwarding*.

3 Le problème traité par l'article

L'article cherche à corriger les problèmes de l'allocation statique en proposant une allocation dynamique. Les problèmes de l'allocation statique sont :

- Le nombre de *noeuds I/O* et prédéfinis et donc ne peut pas changer pendant l'exécution ou s'adapter à une application.
- Ce n'est pas flexible, car la politique d'allocation des *noeuds I/O* reste la même tout au long de l'exécution.
- On peut être amené à de mauvaises allocations des ressources (L'article cite les travaux de Yu et al. [3] Bez et al. [1]).
- En fonction de l'application, on peut ce retrouvé avec une bande passante peu utiliser ou qui ralentit l'application. Ce qui fait qu'on utilise pas assez ou trop de *noeuds I/O*.
- Ne permets pas le changement facile de la stratégie d'allocation sans avoir un mauvais impacte sur les performances.

3.1 Évaluations des politiques d'allocations

Dans l'article, ils ont commencé par évaluer les politiques d'allocation suivante :

- *ZERO and ONE Policies* chaque application à 0 ou un *noeud I/O* alloué.
- *STATIC Policy* le nombre de *noeuds I/O* est déterminé par rapport au nombre de *noeuds de calculs*.
- *SIZE and PROCESS Policies* le nombre de *noeuds I/O* est réparti proportionnellement entre les applications par rapport à leur nombre de *noeuds de calculs* ou leur nombre de processus.
- *ORACLE Policy* chaque application se voit allouer un nombre de *noeuds I/O* qui maximise l'utilisation de la bande passante. Cette allocation est déterminée par une évaluation des performances.

Ils ont effectué ces évaluations en faisant plusieurs lancements avec différents nombres de *noeuds I/O*, différentes politiques d'allocation et avec différentes applications pour avoir différents patterns d'accès au *PFS*. Ces évaluations, ont été effectuées sur le supercalculateur *MareNostrum 4* (*MN4*) en utilisant un outil appelé FORGE (I/O **For**warding **E**xplorer) qui permet de relancer les profils *I/O* des applications.

Pour obtenir les résultats, ils ont fait des traces pour connaître les patterns d'accès des applications au *PFS*. Ils ont collecté le volume total des données transféré et le nombre de processus qui font des requêtes *I/O*. Avec ces traces, ils ont créé des petits benchmarks pour reproduire les patterns d'accès et faire des tests de performance.

Ils n'ont pas eu besoin de faire de profiling, car ils ne s'intéressent qu'aux politiques d'allocation.

Ils ont exécuté 189 patterns sur la machine *MN4* et ils ont fait des groupes de 16 patterns pour simuler chaque politique (Un pattern représente une application). Ils ont donc généré 10 000 groupes de 16 patterns qui prennent le

même temps d'exécution. Ils ont utilisé jusqu'à 128 *noeuds I/O forwarding*, 8 par application ($128/16 = 8$). Avec un nombre de *noeuds de calculs* entre 88 et 512 et une médiane de 256.

La bande passante de chaque groupe est calculée par la somme des nombre d'écriture plus le nombre de lecture le tout divisé par le temps d'exécution. $((nbEcriture + nbLecture)/tempsDExecution)$.

$$aggregate\ BW = \sum_{a=1}^{16} \left(\frac{W_a + R_a}{runtime_a} \right)$$

Cette évaluation montre qu'il est possible d'améliorer l'utilisation de la bande passante. Ils ont aussi vu que toutes les applications n'ont pas les mêmes performances avec le même nombre de *noeuds I/O*. Cette différence existe, car chaque application a un pattern d'accès au *PFS* différent. Ils ont remarqué que certaines applications avaient des patterns similaires.

4 Propositions de l'article

La problématique d'une politique d'allocation peut être représentée comme suit : Pour un ensemble de tâches ou job à exécuter et un nombre fixe de *noeuds I/O*, il faut déterminer le nombre de *noeuds I/O forwarding* qui permet de maximiser l'utilisation de la bande passante globale. Cette problématique peut être considérée comme un problème d'optimisation.

Pour répondre à cette problématique et permettre l'allocation dynamique l'article propose une nouvelle politique d'allocation ainsi qu'un service permettant l'allocation dynamique.

4.1 La politique d'allocations basée sur *MCKP*

La politique d'allocation basée sur **MCKP** (*Multiple-Choice Knapsack Problem*) cherche la meilleure répartition possible des *noeuds I/O forwarding* aux *noeuds de calculs*. Pour cela, cette politique cherche à maximiser la bande passante globale en donnant plus de *noeuds I/O forwarding* aux applications qui en ont le plus besoin et moins aux autres. Ce problème d'optimisation est dérivé du problème *0-1 Knapsack*. Pour effectuer la maximisation de la bande passante, l'algorithme vérifie que chaque tâche a bien une taille et que le nombre global de *noeuds I/O forwarding* calculé ne dépasse pas le nombre de *noeuds I/O* disponibles.

Ce problème fait partie de la classe de complexité en temps *NP-hard* mais une solution peut être obtenue par "Dynamic Programming" qui est en temps *pseudo-polynomial*.

Le nombre de *noeuds de calculs* doit être divisible par le nombre de *noeuds I/O forwarding* pour améliorer l'équilibrage de la charge.

4.2 Le service permettant l'allocation dynamique (*GekkoFWD*)

L'article propose un service *I/O forwarding* appelé *GekkoFWD* qui implémente la politique **MCKP**. Ce service permet à la demande (dynamiquement)

et sans perturber l'application le changement de politiques d'allocation et le changement du *noeuds I/O forwarding*. Il fonctionne au niveau utilisateur, il ne nécessite pas de modification du code des applications et il est simple à déployer. Cette simplicité en fait une solution utile. Ce service est basé sur *GekkoFS* qui est un système de fichier local à un noeud et qui peut se connecter à la plupart des *PFS* existants.

4.2.1 *GekkoFS*

GekkoFS est un système de fichiers local qui est exécuté sur les *noeuds de calculs*. Ce système est utilisé comme un *burst-buffer*. Comme son nom l'indique, c'est une mémoire tampon qui se met entre l'application et le *PFS* dans le but d'améliorer la performance des accès *I/O*. Cette amélioration est en partie due à la diminution de la charge sur le *PFS* quand il y a un pic de demande.

Ce système permet aussi l'exécution des requêtes *I/O* en parallèle de l'exécution du calcul, ce qui permet d'augmenter l'utilisation de la bande passante.

Il fournit aussi un système de nom qui est partagé entre tous les *noeuds de calculs*. Ce mécanisme peut être remplacé par un *PFS*.

4.2.2 Application de techniques d'optimisation (*AGIOS*)

GekkoFWD permet d'appliquer des techniques d'optimisation de façon transparente. Pour ce faire, ils ont ajouté la bibliothèque *AGIOS* dans *GekkoFWD*. Cette bibliothèque fournit plusieurs algorithmes d'ordonnancement. *GekkoFWD* peut donc utiliser différents ordonnanceurs sur les requêtes. Ceci permet la planification des requêtes au niveau des données.

5 Algorithme

Grâce à *GekkoFWD*, *GekkoFS* est utilisé comme un noeud intermédiaire entre les *noeuds de calculs* et les *noeuds I/O*. Pour cela, *GekkoFS* capture les requêtes que fait l'application au *PFS*. Cette capture est transparente et se fait en interceptant les appels systèmes. Une fois les requêtes capturées, *GekkoFWD* transfère les requêtes à un seul serveur qui va les passer à *AGIOS* pour ordonnancer le moment où elles seront traitées. Grâce aux requêtes, il est possible de déterminer quelle politique d'allocation appliquée.

Pour savoir quand le mappage à changer, on ajoute un thread grâce à *GekkoFS*.

Le changement de politique se fait au démarrage ou au changement de la liste des applications lancées.

6 Les performances

Ils ont fait les expérimentations sur la plateforme *Grid 5000* (*G5K*) avec 2 clusters à Nancy : *Grimoire* (8 noeuds) et *Gros* (124 noeuds).

Ils ont utilisé 5 noyaux d'applications différentes ainsi que les micro-benchmarks *IOR* :

1. *S3D I/O Kernel* : S3D
2. *MADBench2* : MAD

3. *HACC-IO* : HACC
4. *S3aSim* : SIM
5. *NAS BT-IO* : BT-C, BT-D

IOR *MPI-IO / POSIX* : IOR-MPI, POSIX-S, POSIX-L

Ils ont mesuré la bande passante pour chaque application et il confirme les résultats de *FORGE*. Ils ont fait ce test avec des nombres de noeuds et des nombres de processus différents. Les ensembles de tâches pour tester le changement d'allocation dynamique étaient composés de BT-C, BT-D, IOR-MPI, POSIX-L, MAD, et S3D.

Il constate bien que les accès statiques ne sont pas très efficaces sauf pour S3D. Même constatation pour l'allocation par taille. Dans ces 2 cas, on voit que par exemple *IOR-MPI* pourrait être bien meilleurs avec plus de *noeuds I/O forwarding*. Il constate bien une nette amélioration avec leur implémentation *MCKP*. Le gain global d'utilisation de la bande passante est jusqu'à 85% par rapport à la politique statique.

Ils ont comparé différents scénarios pour montrer que leur implémentation fonctionne bien.

7 Les articles qu'il référence

Les articles référencés parlent de :
— Tests de performance des différentes politiques d'allocation.
— De l'optimisation.
— Des gestions d'*I/O* des supercalculateurs.
— De *I/O Forwarding* à la demande.
— Du service *GekkoFS*.
— D'ordonnancement.

Références

[1] Jean Luca Bez, Francieli Z. Boito, Alberto Miranda, Ramon Nou, Toni Cortes, and Philippe O. A. Navaux. Towards on-demand i/o forwarding in hpc platforms. In *2020 IEEE/ACM Fifth International Parallel Data Systems Workshop (PDSW)*, pages 7–14, 2020.

[2] Jean Luca Bez, Alberto Miranda, Ramon Nou, Francieli Zanon Boito, Toni Cortes, and Philippe Navaux. Arbitration policies for on-demand user-level i/o forwarding on hpc platforms. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 577–586, 2021.

[3] Jie Yu, Guangming Liu, Wenrui Dong, Xiaoyong Li, Jian Zhang, and Fuxing Sun. On the load imbalance problem of i/o forwarding layer in hpc systems. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 2424–2428, 2017.