Systèmes de fichiers distribués

« Nouvelle implémentation des SDDS dans un environnement Java/Linux»

Samba NDIAYE, Balla Wade DIACK

Département de Mathématiques et Informatique
Université Cheikh Anta DIOP de Dakar
18522 DAKAR
SENEGAL

diackballa@yahoo.fr,samba.ndiaye@ucad.edu.sn

RÉSUMÉ. Aujourd'hui les Systèmes de fichiers distribués en mémoire centrale sont incontournables pour l'accès rapide aux données. Les Structures de Données Distribuées et Scalables qui, créées à partir d'un seul site et peuvent s'étendre à un nombre illimité d'autres sites constituent un système de gestion de fichiers distribués idéal. L'objectif de ce papier est une nouvelle implémentation de ces Structures dans un environnement Linux/Java après qu'elle ait été implantée en C++ et dans un environnement Windows et une amélioration de l'architecture des clients et serveurs SDDS qui y sont proposée.

ABSTRACT. The exponential increasing of data and growing of storage capacities coupled with a few growing of transfer speed in networks are making the Distributed File Systems (DFS) crucial in data transmitting. Data Structures Scalable Distributed, created from a single site and may extend to a theoretically unlimited number of other sites are optimal for DFS Management. The aim of this paper is a new implementation of these structures in a Linux/Java environment and also an improvement of the precedent architecture in SDDS-2000.

MOTS-CLÉS : Multiordinateurs, Structures de Données Distribuées et Scalables, Linux.

KEYWORDS: Multicomputers, Scalable Distributed Data Structures, Linux.

1. Introduction.

Les années 2000 ont vu une grande avancée au niveau des langages de développement de haut niveau. L'amélioration de son API l'enrichit continuellement de nouveaux paquetages. Ceux-ci permettent de gérer la programmation concurrente de même que les files d'attente; ce qui se trouve être très utile pour la communication au niveau des systèmes distribués scalables. Linux est en 2010 le système dominant dans le monde des serveurs. Il a rattrapé son retard technologique sur Windows et l'a même dépassé en termes de robustesse !

En 2010, il nous semble pertinent de réaliser une nouvelle implémentation des SDDS dans l'environnement JAVA/LINUX, dix ans après la première implémentation SDDS - 2000 dans l'environnement Visual C++/Windows 2000.

2. SDDS et SDDS RP*.

2.1. Les Structures de Données Distribuées Scalables :

Les structures de Données Distribuées et Scalables ont été proposées pour la gestion de données en mémoire centrale distribuée [LNS94]. Elles fournissent un mécanisme général d'accès à des données réparties dynamiquement. Elles ont été définies pour des fichiers distribués sur un multiordinateur sans partage de mémoire. Ces structures assurent des temps d'accès beaucoup plus courts que les performances d'accès aux données stockées sur les disques. Cette nouvelle structure dispose, de plus du traitement parallèle, d'une capacité de stockage potentiellement illimitée. Ces caractéristiques assurent aux SDDS des performances de traitement supérieures à celles des structures de données traditionnelles. Le premier schéma de cette famille : LH* [LNS93a], était une adaptation du hachage linéaire aux multiordinateurs. Les données d'un fichier SDDS sont stockées sur des ‘‘Serveurs’’, des ‘‘Clients’’ y accèdent et gardent des paramètres pour le calcul de l'adresse des Serveurs. Ces paramètres constituent l'image du client du fichier SDDS. Un fichier SDDS contient des données sous forme typique d'enregistrements, identifiés chacun par une ou plusieurs clés. Étant donné le schéma d'adressage en cours du fichier, la valeur des clés détermine le serveur correct où l'enregistrement doit être stocké. Un fichier SDDS débute sur un seul site serveur et peut être étendu par insertion à un nombre quelconque de sites.

2.2. SDDS RP*

Ces fichiers sont composés de cases contenant des enregistrements. Chaque case est délimité par un intervalle (λ, Λ] contenant les enregistrements de clé c appartenant à (λ, Λ]. Les enregistrements sont ordonnés dans les cases suivant l'ordre croissant des clés. On distingue dans cette famille des RP trois sous-familles.

Dans les premiers ($RP*_N$), les clients n'ont pas d'images du fichier SDDS; ils envoient leurs requêtes par messages *multicast*. Dans les $RP*_C$, les clients ont une image du fichier SDDS. Les requêtes sont envoyées par messages *unicast*; ils sont plus efficaces que les RP*N. La dernière sous famille ($RP*_S$) ajoute à $RP*_C$ un index réparti au niveau de serveurs spécifiques appelés serveurs d'index. Cet index représente une image de la structure globale de la répartition du fichier distribué. Ses objectifs sont d'éliminer l'usage du *multicast* pour envoyer les requêtes simples, et d'accélérer la convergence de l'image du client pour réduire le nombre d'erreur d'adressage.

3. Architecture des Serveurs et Clients SDDS-2010.

3.1. Serveur SDDS 2010

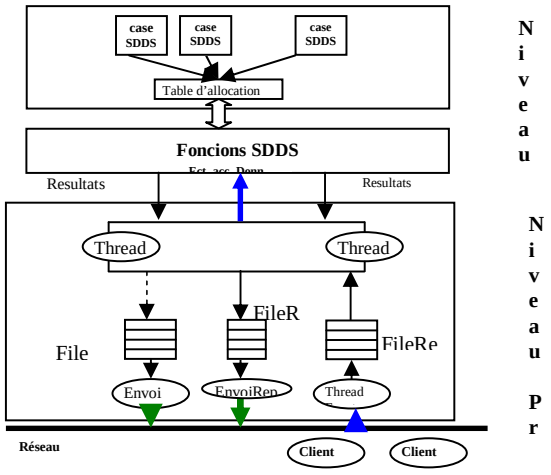


Figure 1. *Architecture du serveur SDDS -2010*

3.1.1. Niveau interne

La gestion interne d'une case est réalisée à ce niveau. Les cases SDDS s'y trouvent de même qu'une table d'allocation des cases permettant de connaître les fichiers qui ont des cases au niveau de ce serveur.

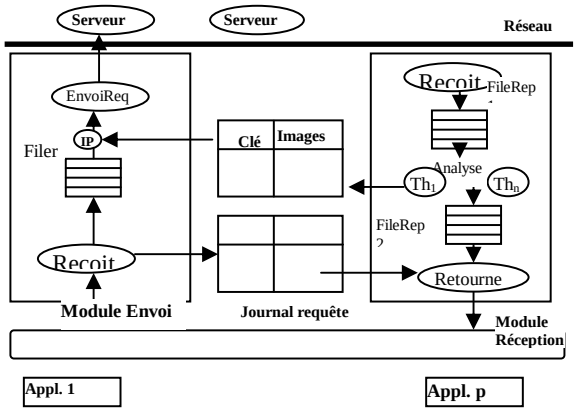
3.1.2. Niveau des Fonctions SDDS

Ce niveau s'appuie sur deux bibliothèques de fonctions. Une première qui fournit une interface d'accès aux données d'une case. En fait, ce sont ces fonctions qui sont appelées par les Threads de travail (*WorkThreads*) pour manipuler les données dans une case. Ensuite une deuxième qui est constituée de fonctions systèmes. Toutes les opérations sont réalisées de façon synchrone.

Dans ce niveau il y a un thread qui écoute continuellement sur un port UDP les requêtes provenant des clients. Dès qu'une requête arrive, ce thread se charge de le mettre dans une file d'attente et de lancer l'évènement *ArriveeRequete*. Le premier thread de travail libre retire la requête en tête de file, lit le type de traitement approprié et invoque la fonction à laquelle il correspond. Contrairement à SDDS-2000 [D01] les *workThreads* ne retournent plus directement les réponses au client mais les mettent dans une autre file où un autre thread les extrait et les renvoie aux clients. Dans le cas d'une redirection provenant d'un autre serveur pour une demande d'éclatement ou un multicast dû à une erreur sur l'adresse du serveur : il y a un thread de service qui écoute sur un port UDP. Ce thread récupère ce message et envoie un acquittement au serveur demandeur au cas où le il est favorable pour l'éclatement. Sinon il ignore ce message et continue à attendre.

3.2. Client SDDS 2010

Le client SDDS est un middleware entre les applications locales et les serveurs SDDS. Il dispose de deux modules.



U C A D

Figure 2. *Architecture du client SDDS -2010*

3.2.1. Module d'envoi

Un thread *RecoitReq* se charge de placer la requête reçue d'une application locale dans une file d'attente *FileReq* et signale l'événement *RequeteAEnvoyer*. Un thread *EnvoiRequete* la récupère pour en faire un message qu'il envoie au serveur approprié. Si la requête porte sur un intervalle de clé [*clé_min*, *clé_max*], le client peut définir un délai après l'envoi de la requête en multicast aux serveurs. Si après expiration tous les enregistrements envoyés sont reçus, il ferme la connexion sinon il renvoie les messages aux serveurs qui n'ont pas répondus. Nous pouvons utiliser deux approches; nous lançons la requête à tous les serveurs en précisant un port TCP où recevoir des données puis ce thread ouvre ce port et fixe un délai suffisant. A l'expiration de ce délai il ferme le port et juge que les serveurs qui n'ont pas envoyé de réponses n'ont pas reçu la requête ou sont en panne. L'autre approche consiste à spécifier à chaque serveur d'envoyer un accusé montrant qu'il a envoyé tous les enregistrements qu'il contenait. Et quand il aura reçu tous les accusés, le client ferme la connexion.

3.2.2. Module de réception

Dans le cas d'une réception, un thread *RecoitRep* écoute sur un port UDP. Dès réception, il déclenche l'événement *ArriveeRequete*. Les requêtes sont classées en fonction de leur arrivée dans une file d'attente. Un des threads d'analyse de réponse extrait une requête de la liste et met à jour l'image du serveur en cas d'erreur. Ensuite, l'identifiant de la requête et le résultat du traitement sont constitués dans une structure et mis dans une seconde file d'attente FileReq2. Les réponses sont récupérées dans cette file et transmises aux clients par le thread *RetourneReponseAppl* aux applications via l'interface SDDS qui tourne sur l'ordinateur local. C'est le journal où sont enregistrées les ID des applications et de leurs requêtes correspondantes. Il permet de reconnaître l'appartenance des requêtes. En effet l'ID de la requête est incluse au message envoyé au serveur qui la retourne dans sa réponse et le journal indiquent l'application avec laquelle elle correspond.

4. Expérimentation et validation.

Nous implémentons dans l'environnement Linux Debian. Java 6 nous sert de langage de programmation dans l'environnement de développement intégré Eclipse. Le prototype est déployé sur un réseau local Ethernet 1 Gb/s reliant six machines Pentium Dual Core (respectivement 2.50 GHz et 1.60 GHz) à 2 Go de RAM chacune et fonctionnant en dual boot Windows et Linux/Debian. On réalise cette expérience sur 6 machines dont 5 serveurs et 2 clients (sachant que l'un se comporte aussi bien en serveur qu'en client).

Les clés des enregistrements sont générées sur java.util.Random. Les requêtes sont des insertions et des recherches simples. La taille de données envoyées par message est de 100 octets. Le temps de création du fichier, le temps d'éclatement, le temps moyen d'insertion sont les indicateurs de performances dans cette expérimentation.

4.1. Création de fichiers

Dans cette expérience on étudie le temps moyen d'insertion par enregistrement lors de la création du fichier. Il est calculé avec 100.000 enregistrements. Concernant les l'expérimentation dans avait permis de montrer que les temps moyens par insertions étaient plus importants avec eux qu'avec les $RP*_C$. Il est vérifié dans [D01] que la différence du temps moyen d'insertion dans les $RP*_N$ peut aller de 2 à 6% selon la présence ou non de contrôle de flux.

4.2. Recherches simples

Le client lance des requêtes de recherche simple (i.e. pas par intervalle). Comme chaque client dispose de l'IAM des serveurs, il envoie la requête à celui à qui correspond l'adresse de l'enregistrement concerné. On considère qu'il n'y a pas d'erreur d'adressage pour cette expérimentation. On envoie 100.000 requêtes de recherche simple. Pour un seul serveur le temps est au dessus de 1100 millisecondes. Ce temps dépasse légèrement 900 millisecondes quand on augmente le nombre de serveurs. Le temps moyen pour la recherche d'un enregistrement avec les hypothèses sus-citées est de l'ordre de 0,009 un peu meilleur que dans l'expérimentation de [D01]. Malgré cela, il confirme l'accès très rapide aux données en mémoire garanti par les SDDS.

4.3. Eclatement d'une case

Nous étudions le temps moyen pour envoyer un enregistrement dans le serveur destinataire lors de l'éclatement des données. Le serveur débordé envoie un message multicast pour chercher un serveur libre. Ce dernier envoie un message pour montrer qu'il est prêt pour l'éclatement. Ensuite le serveur envoie la moitié supérieure de ses enregistrements. Nous avons réalisé le calcul des temps moyen d'envoi d'un enregistrement à l'éclatement lors de la création du fichier. Quand on crée un fichier de 3.000.000 d'enregistrements, avec des cases de capacité 1.000.000 il s'étend sur 3 serveurs. Les temps moyen d'envoi par enregistrement convergent ici vers 0,009 ms. Comparé aux résultats de [D01] (sic 0,037 et 0,007 resp. pour $RP*_C$ et $RP*_N$) ce temps moyen est acceptable par rapport aux théories prévues par les publications sur ces systèmes de fichiers.

4.4. Comparaison des résultats avec des travaux antérieurs sur les SDDS

Une comparaison avec les analyses théoriques contenues dans les premiers articles sur les SDDS a été réalisée dans [D01]. Les résultats de cette expérimentation seront comparés à ceux de notre implémentation pour montrer sa validité dans ce nouvel environnement ainsi que pour ce nouveau langage.

	RP^*_N Thr.	RP^*_{N1}		RP^*_{C1}		RP^*_{C2}
		FC+	FC-	FC+	FC-	
t_c	40250	69209	47798	67838	45032	33000
$t_{i,c}$	0,268	0,461	0,319	0,452	0,279	0,220
t_{ins}	0,161	0,229	0,095	0,281	0,086	0.018

Table 4 : *Comparaison des performances*

FC+ : avec control de flux
FC- : sans control de flux
Thr. : théorique.
RP^*_{N1} : RP^*_N dans [D01]
RP^*_{C1} : RP^*_C dans [D01]
RP^*_{C2} : RP^*_C dans SDDS – 2010
t_c : temps de création d'un fichier
$_{ti,c}$: temps d'insertion pendant la création du fichier
t_{ins} : temps moyen d'une insertion

Une différence importante se fait sentir au niveau du temps de création d'un fichier SDDS selon l'absence ou pas de contrôle de flux dans [D01]. Sans les contrôles de flux, les temps obtenus sont proches de celles prévu par la théorie sur les SDDS RP^*_N selon les même données initiales. Avec des données initiales quasi similaires nous sommes parvenus à avoir des temps nettement meilleurs. Ceci démontre que l'implémentation dans java constitue un avantage. L'intérêt principal de ce travail constituait à voir si l'implémentation dans ce nouveau langage permettait de conserver au moins les performances théoriques de ces fichiers.

Le temps moyen par insertion connait une nette diminution avec cette expérimentation. Ceci s'explique en partie par la vitesse de traitement plus importante

au niveau des serveurs SDDS-2010, mais certainement aussi de la stabilité et de la fiabilité des communications assurés par Linux.

5. Conclusion

Cet article consistait à montrer qu'on pouvait implémenter les SDDS sur Java/Linux en garantissant les performances (temps d'accès, scalabilité…).

5.1. Bilan

Nous avons décrit la famille des SDDS RP* qui préserve l'ordre ; elle est constituée de trois sous-famille : les RP*N, RP*C et les RP*S. Les images des fichiers se trouvent aussi bien au niveau des clients que des serveurs pour les derniers et n'existent pas sur les premiers. Au niveau des RP*C les images se trouvent au niveau des clients seulement. Ensuite nous avons présenté notre architecture SDDS-2010 avec les modifications qu'on a faites sur les clients et les serveurs de SDDS-2000. Nous avons terminé par faire des expérimentations pour valider notre implémentation sur Linux/Java.

5.2. Perspectives

Hadoop est un Framework développé sur Java par Apache et qui est en évolution continuelle, il est décisif pour les systèmes de fichiers distribués. Dans un proche avenir nous allons concevoir et implémenter un système de gestion de fichiers distribué robuste en RAM, en s'inspirant des Structures de Données Distribuées et Scalables.

5. Bibliographie et biographie

5.1 Bibliographie

[BS04] BOULARIAS A., SAAD B. *Implémentation et étude des performances de la SDDS Compact Trie Hashing (CTH) sous Windows 2000*. Mémoire d'ingénieur d'Etat. INI Algérie, 2004.

[D01] Diène, A. W. *Contribution à la Gestion de Structures de Données Distribuées et Scalables*. Thèse de doctorat. Université de Paris 9 Dauphine, 2001.

[LNS93a] Litwin, W. Neimat, M-A., Schneider, D. LH*: Linear Hashing for Distributed Files. *ACM-SIGMOD Int. Conf. On Management of Data*, 1993.
[LNS94] Litwin, W. Neimat, M-A, Schneider, D. RP*: A Family of Order- Preserved Scalable Distributed Data Structures. *20th Intl. Conf. On Very Large Data Bases* (VLDB), 1994.

5.2. Biographie.

5.2.1. Parcours universitaire :

Après l'obtention d'un baccalauréat en Mathématiques et Physiques en 2002 nous avons fréquenté l'Université Gaston Berger de Saint-Louis où nous avons obtenu un DEUG de Mathématiques Appliquées et Informatique (2005, Mention assez Bien) ; une Licence puis une Maîtrise en Informatique (2006-2007, mention Très Bien)

A partir de Novembre 2007 nous nous sommes inscrits au département de Mathématiques et d'Informatique de l'UCAD où nous avons obtenu respectivement une Attestation d'étude approfondies et un DEA (2008-2010, mention Assez Bien). Nous sommes inscrits en Doctorat d'Informatique pour l'année 2010/2011.

5.2.2. Activités professionnelles :

Janvier à Mai 2007 : Mise en place d'un logiciel de gestion de GAB en Java

Février à Juillet 2007 : Projet de mise en place d'une plateforme de traitement d'images : SANARVISION.

Depuis novembre 2008: Enseignement à l'Université Cheikh Anta Diop de Dakar à la Faculté des Sciences et Techniques (vacataire).

Décembre 2010 : Mémoire de DEA « Nouvelle implémentation des Structures de Données Distribuées et Scalables dans un environnement Java/Linux. »