Publication avant LAFMC

Titre	Implementation d'un Client Universel RP*
Auteurs	**Samba Ndiaye**, Joseph E. Ndiaye, Mohamed T. Seck, Witold Litwin
Référence	7e Colloque africain sur la recherche en informatique et mathématiques appliquées (CARI 2004)
Editeur	CARI
Pages	171 - 177
Année	2004
DOI	
URL	http://www.cari-info.org/actes-2004/17-Ndiaye.pdf
Index	
ISBN	
Encadreur	Non
Extrait d'une thèse	Non

Accéder au rapport CARI-2004 / Access to CARI'2004 report

PROCEEDINGS

Implémentation d'un Client Universel RP*

Samba Ndiaye[1], Mohamed T. Seck[1], Joseph E. Ndiaye[1], Witold Litwin[2]

[1]Université C.A.Diop de Dakar (Sénégal)
{ndiayesa, seckm, joendiaye}@ucad.sn

[2]Université de Paris Dauphine (France)
witold.litwin@dauphine.fr

RÉSUMÉ. Les structures de données distribuées et scalables (SDDS) constituent un nouveau type de données conçues pour les réseaux d'ordinateurs. La famille des SDDS-RP* permet de créer des fichiers ordonnés pouvant s'étendre dynamiquement sur plusieurs ordinateurs. Plusieurs travaux de grande qualité ont déjà effectués. Pour l'essentiel, ils ont concernés le développement des SDDS, côté serveur au sens de l'architecture Client/Serveur. Nous nous intéressons ici à la conception et au développement d'une interface « client web» pour permettre leur utilisation transparente par le grand public. Nous montrons que les navigateurs peuvent être utilisés comme interface. Ce résultat va dans le sens de la tendance actuelle qui fait du navigateur Web le client universel sur le poste client pour tous les applicatifs.

ABSTRACT. A Scalable Distributed Data Structure (SDDS) is a data structure of a new type specifically designed for multicomputers, P2P and grid computing systems. SDDS RP* provides a range partitioned file scaling up dynamically over distributed nodes. Our concern is the conception and the implementation of web interface for SDDS servers.

MOTS-CLÉS : SDDS, multiordinateur, Composant, COM, Internet, Applicatif Web, Client RP*

KEYWORDS: SDDS, multicomputer, Composant, COM, Internet, Web Application, RP* Client

1. Introduction

Un multiordinateur est une collection d'ordinateurs faiblement couplés, sans mémoire partagée. Aujourd'hui toutes les entreprises, centres de recherche, universités, etc. sont dotés de réseaux d'ordinateurs. Ceux-ci constituent des multiordinateurs dits *réseaux* fonctionnant sans interruption.

Litwin et autres ([LNS93]) se sont appuyés sur ce concept de multiordinateur pour proposer une nouvelle classe de structures de données capables d'exploiter ce potentiel presque infini : les structures *de données distribuées et scalables* (SDDS). Les SDDS sont des fichiers qui n'existent qu'en mémoire centrale, qui s'étendent de façon dynamique sur un ou plusieurs ordinateurs tout en maintenant à un niveau constant leurs performances d'accès. Ces structures de données sont basées sur le modèle client / serveur.

Nous nous intéressons principalement dans notre étude à la famille des SDDS RP* qui préserve l'ordre des enregistrements. ([LNS94]). Cette famille RP* est divisée en trois sous familles : $RP*_N$, $RP*_C$ et $RP*_S$. Le protocole de communication de $RP*_N$ utilise une adresse multicast pour diriger les requêtes (recherche, insertion, mise à jour) du client vers tous les serveurs. Le protocole $RP*_C$ est une reprise du protocole $RP*_N$, avec en plus une représentation de l'image du fichier SDDS au niveau de chaque client. L'image du fichier au niveau du client est une table d'index T telle que chaque ligne est formée d'un couple (adresse de serveur, clé maximale). Finalement, le protocole de communication $RP*_S$ ajoute à $RP*_C$ une table d'index au niveau de serveurs indexant toutes les cases. Dans ce dernier cas, les requêtes et les redirections sont envoyées par des messages unicast.

Les travaux de recherche précédents sur les SDDS [**DIE98**], [**DIE01**], [**NDI98**] ont permis l'implémentation d'un prototype de fichiers SDDS téléchargeables sur le site Web http://ceria.dauphine.fr. De tels fichiers sont créés et s'étendent sur des ordinateurs dits serveurs SDDS. Ainsi chaque serveur SDDS contient une partie du fichier SDDS. Les mesures de performance réalisées donnent des résultats de loin supérieurs à ceux des fichiers classiques qui n'existent que sur un seul ordinateur [**DIE01**].

Notre travail consiste à concevoir et à réaliser des interfaces permettant d'exploiter les fichiers SDDS notamment par le grand public à l'aide de navigateurs Web, d'un serveur http et de composants COM. Il s'agit de permettre l'accès à un fichier SDDS à partir d'un navigateur et de faire de façon transparente les opérations d'insertion, de modification, de suppression d'enregistrements.

La section 2 présente l'architecture de l'applicatif Web. Les mesures de performance sont exposées à la section 3. Enfin, à la section 4, nous présenterons la conclusion de ce travail.

2. Notre Architecture

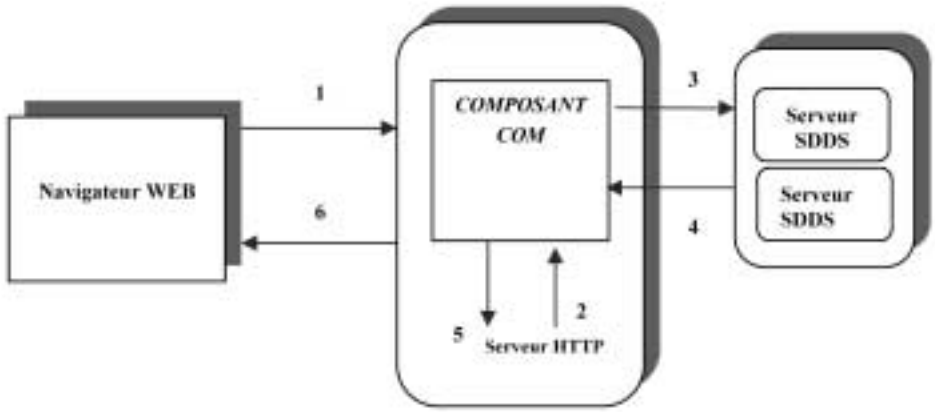

Figure 1. *Architecture générale*

Etape 1 : Un utilisateur via un navigateur Web effectue une opération (insertion, modification, recherche ou suppression de données) sur un fichier SDDS. La requête est acheminée vers un serveur HTTP concerné.

Etape 2 : Le service http transmet la requête à l'objet COM.

Etape 3 : L'objet COM envoie la requête vers les serveurs qui contiennent le fichier SDDS via les protocoles de communication choisis c'est-à-dire par :

* ***multicast*** (cas client RP*N) : Dans ce type de protocole la requête est destinée à tous les serveurs appartenant au groupe multicast.

* ***unicast*** (cas client RP*C) : Dans ce type de protocole la requête est adressée à un destinataire unique sur le réseau.

Etape 4 : Le ou les serveurs SDDS concernés renvoient la ou les réponses à l'objet COM.

Etape 5 : L'objet COM renvoie la ou les réponses au service HTTP.

Etape 6 : Le service http envoie la ou les réponses au navigateur.

Conception et implémentation du composant COM

Un composant COM est vu comme une classe qui expose un certain nombre de fonctions membres, ces fonctions étant regroupées en ***interfaces.*** Nous développons une DLL composée des classes suivantes :

Classe CInitSocket, Classe ClientEnvoi, Classe ClientRecoit.

Ensuite, la DLL est encapsulée la dans un composant COM ayant comme interface les méthodes suivantes :

Envoyer; Reponse; EnvoyerRequeteParall ; EnvoyerRequeteParallelleUDP; EnvoiRequeteSimpleMultiple; RecevoirReponseParallelleUDP; ReceptionInsertionSimpleMultiple.

Nous avons également défini les propriétés suivantes :

get_NombreReponseParallelleTCP, get_ NombreReponseParalleUDP, get_NombreReponseInsertionSimpleMultiple.

Ces méthodes et propriétés de l'objet COM ont fait appel aux différentes méthodes définies dans la DLL.

Enfin, le composant COM est utilisé dans une page Web qui constitue notre **client universel**. Le langage VBScript est utilisé dans nos pages ASP. Les appels suivants *« UnObjet.Envoyer Chaine,0 »* , *« reponse= UnObjet.Recevoir»* font appel aux méthodes de l'objet créé.

3. Mesures de performance

Environnement expérimental

Dans ces mesures, nous avons placé les cinq serveurs SDDS sur cinq ordinateurs ayant les caractéristiques suivantes: Windows NT Server, Pentium III 733 MHz, 64 Mo. Quand au serveur http, il a les caractéristiques suivantes : Windows NT Server, Intel Celeron 633 MHz, 128 Mo. Le réseau local possède un débit de 10 Mbit/s. Nous avons fixé à 10 octets la taille des enregistrements à insérer lors de nos différentes expériences.

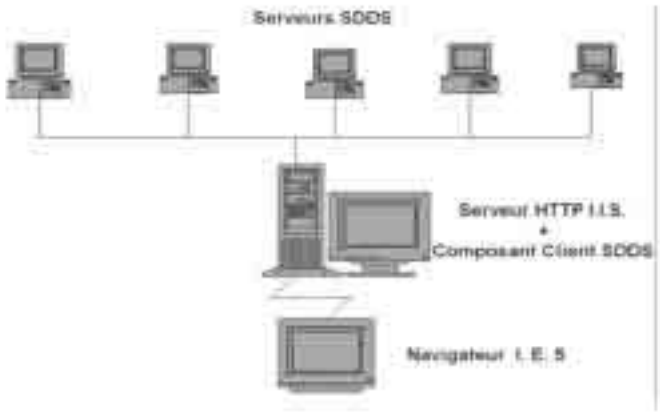

Figure 2 : Environnement Expérimental

3.1. Recherche par intervalle

Recherches sur l'intervalle de clés [0 ; 450] s'étendant sur cinq serveurs SDDS

Serveur 1 :] 0, 90] ; **Serveur 2** :]90, 180] ; **Serveur 3** :] 180, 270] ; **Serveur 4** :] 270, 360] ; **Serveur** :] 360, 450]

En mesurant le temps de réponse de cette recherche, nous obtenons les résultats suivants :

Nombre enregistrements sur Nombre serveurs	Intervalle de recherche	Temps de réponse (en ms)	Temps moyen de recherches d'un enregistrement (en ms)
450 enregistrements sur 5 serveurs	] 0 , 450]	280.2	0.62

En mesurant le temps de recherche moyen d'un enregistrement nous avons obtenu une moyenne qui tourne autour de 0.62 ms. Nous pouvons donc en déduire que la recherche par intervalle sur plusieurs serveurs SDDS se trouvant sur des machines différentes offre de bons résultats. Donc le fait d'augmenter le nombre de serveurs dans une requête par intervalle n'altère pas grandement les temps moyen de recherche d'un enregistrement.

5.2. Insertions aléatoires sur cinq serveurs SDDS avec génération d'éclatements

Il s'agit dans ce cas d'effectuer des insertions en ayant des éclatements des fichiers SDDS. Pour cela, il suffit donc de prégénérer les intervalles. Nous distinguons ici comme intervalle pour les cinq serveurs SDDS :] 0 , 100] ;] 100 , 200] ;] 200 , 300] ;] 300 , 400] ;] 400 , 500]. Les insertions s'effectuent également comme pour les précédentes recherches de façon aléatoire.

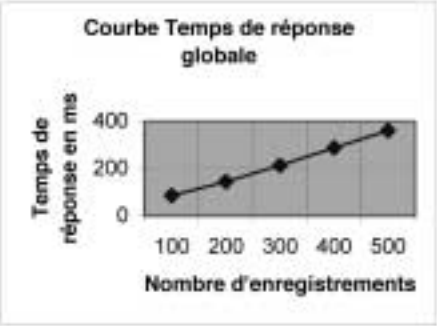

Figure 3 : temps de réponse

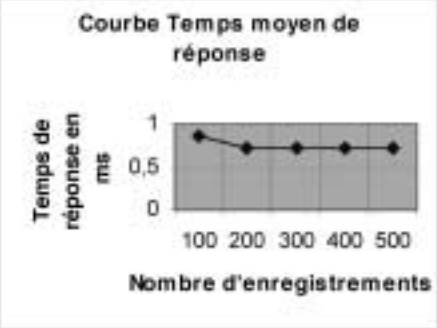

Figure 4 : temps moyen de réponse

Les résultats obtenus montrent que les temps de réponse d'une insertion moyenne tournent aux alentours de 0.71 ms par enregistrement. Pour observer cet éclatement, il faut nécessairement que l'ensemble des clés du serveur SDDS soit rempli. En passant de quatre à cinq serveurs, nos mesures commencent à se stabiliser. L'augmentation du nombre de serveurs SDDS sur d'autres machines ne bouleverse pas nos temps d'insertions obtenues. Le résultat de tous ces tests dépend de la puissance des machines utilisées pour le client comme pour les serveurs.

5. Conclusion

Nous avons rendu dans notre étude le client SDDS RP*N universel à travers d'un applicatif Web. Nous avons proposé une architecture générale qui supportera l'applicatif. Nous y disposerons : d'un client Web avec son navigateur, d'un serveur **HTTP**, et des serveurs SDDS. Nous avons ensuite développé un composant **COM** qui a encapsulé les fonctionnalités des clients RP*. Ce composant est alors enregistré au

niveau du serveur **HTTP,** et les Browsers du client Web feront appel à lui à travers des pages **ASP**.

Les mesures de performances effectuées à partir de l'applicatif, ont prouvé que le fait de rendre le client universel ne perturbe pas le bon fonctionnement des **SDDS**. Ces mesures ont porté sur une série d'insertions simultanées, et de recherches par intervalles. Les temps de réponse sont corrects par rapport aux mesures effectuées dans les précédents travaux de recherche. Nous pourrons affiner les mesures de performance par des tests sur des serveurs plus rapides.

13. Bibliographie

[DIE98] Diène A. W., *Organisation interne d'une case SDDS RP**, Mémoire de DEA, Département Informatique, Université Cheikh Anta Diop de Dakar, 1998.

[DIE01] Diène A. W., http://ceria.dauphine.fr

[NDI98] Ndiaye Y. B. A. K., *Mise en œuvre de l'architecture de communication des Structures de Données Distribuées et Scalables RP*$_N$ et RP*$_C$*, Mémoire de DEA, Département Informatique, Université Cheikh Anta Diop de Dakar, 1998.

[LNS93] 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-Preserving Scalable Distributed Data Structure. VLDB-1994*

[AYF00] Aly W. Diène, Yakham Ndiaye, Fethi Bennour: Gestionnaire de SDDS RP* sous Windows NT 2000, Rapport de recherche, CERIA, Université Paris Dauphine.

[GRIM99] Grimes Richard, *La Référence du programmeur*, Eyrolles, 1999.

[CU00] C.Ullman, D. Buser, J. Ducket, B. Francis, J. Kauffman, J.T. Libre, D.Sussman, *Initiation à ASP 3.0*, Traduit de l'anglais par Florence Thierry, Geneviève Vassaux, Ingrid Pigueron, Eyrolles 2000.