Publication avant LAFMC

Titre	Scalabilité des Performances des requêtes parallèles dans les SDDS-RP*
Auteurs	**Samba NDIAYE**, Mouhamed T. SECK, Mianakindila TSANGOU
Référence	Proceedings of the Fifth Workshop on Distributed Data and Structures, June 2002 (WDAS 2002)
Editeur	WDAS
Pages	
DOI	
URL	
Index	
ISBN	

Scalabilité des Performances des requêtes parallèles dans les SDDS-RP*

Samba NDIAYE[1] *Mouhamed T. SECK[2]* *Mianakindila TSANGOU[3]*

Résumé étendu

*Nous nous intéressons aux requêtes parallèles dans les SDDS RP. Après avoir lancé sa requête, l'application **cliente** doit être capable de réceptionner sans perte les flux de données provenant de tous les serveurs concernés par la requête. Nous proposons trois stratégies. Nos expérimentations montrent que la troisième, qui consiste à recevoir les données des serveurs de manière séquentielle, est la meilleure.*

1. Position du problème

Une nouvelle structure de données, dite **Structure de Données Distribuées Scalables (SDDS),** a été proposée ([LNS93], [LN94], [K98]) dans le but d'exploiter les ressources gigantesques et toujours croissantes des réseaux actuels d'ordinateurs. Un fichier SDDS existe dans les RAM de certains ordinateurs dits serveurs ou *buckets* et peut s'y étendre de manière dynamique.

Les SDDS sont divisées en deux grandes familles :

- les **SDDS-LH*** qui utilisent les algorithmes de hachage LH[L80] pour l'accès aux clés
- les **SDDS-RP*** qui sont ordonnés. Dans les SDDS-RP*, il existe trois sous-familles qui sont :
  - **RP*$_n$, RP*$_c$ et RP*$_s$.**

Il existe des implémentations du fichier SDDS-RP* ([D97], [N97], [T200]).

Nous nous intéressons aux SDDS-RP*$_n$ où les clients et les serveurs ne sont pas dotés d'index et utilisent donc des messages multicast pour communiquer. Un message multicast étant un message envoyé simultanément à tous les ordinateurs d'un groupe donné partageant une certaine adresse IP.

Chaque bucket est représenté par un intervalle ordonné [λ, Λ] où λ et Λ sont des nombres réels tels que λ < Λ. Un enregistrement r ∈ [λ, Λ] si sa clé C est telle que λ <C <= Λ. Le fichier SDDS-RP* est alors représenté par un ensemble d'intervalles [λ_i, Λ_i] avec i = 1,2,..., n.

Une application appelée *client* tournant sur une station de travail donnée envoie une requête par intervalle, elle reçoit des données en provenance des serveurs ou buckets concernés, qui peuvent être très nombreux.

Chaque requête par intervalle est caractérisée par un intervalle de recherche [CleMinReq, ClemaxReq].

Un bucket **[λi, Λi]** est concerné par la requête par intervalle si **[λi, Λi,] ∩ [CleMinReq, ClemaxReq] ≠∅.**

Dans ce cas, le serveur i essaie d'établir une connexion avec le client et, en cas de succès, lui envoie les données.

Si le nombre de demandes de connexions est très élevé, le client ne peut y répondre favorablement et il y a donc possibilité de perte de données.

C'est pourquoi nous proposons 3 stratégies pour résoudre ce problème.

1.1Stratégie N°1

Le client accepte automatiquement, autant qu'il le peut, toutes les demandes de connexions des serveurs. Le protocole TCP est utilisé comme protocole de transport des données. Nous étudions alors le comportement du client en fonction du nombre de demandes de connexions des serveurs.

1.2 Stratégie N°2

Le nombre de demandes de connexion possibles supportées par le client est fixé arbitrairement à M (M entier positif). Alors, seuls M serveurs au maximum peuvent envoyer simultanément des données au client

1.3 Stratégie N°3

D'abord, chaque serveur concerné établit une connexion UDP avec le client et lui envoie son adresse son nom, intervalle et adresse. Ensuite, après avoir construit la liste de tous les serveurs concernés, le client établit des connexions TCP de façon séquentielle avec eux, afin de recueillir toutes les données.

(1 : ndiayesa@ucad.sn 2 : seckm@ucad.sn 3 : tsangou@esp.simes.sn)
*Université Cheikh Anta Diop de Dakar – Equipe de recherche sur les SDDS RP**

2. Expérimentations

2.1 Architecture du client

En utilisant le langage Java, nous proposons une architecture du client qui exploite les threads.

Le client java est constitué de 2 modules tournant en parallèle : le module de construction et le module d'écoute. Le premier permet de construire la requête et de l'envoyer aux différents serveurs RP* du fichier et le second a la charge de la réception de toutes les réponses venant des serveurs.

2.2 Protocole de mesures

Notre critère de comparaison des trois stratégies est le temps de réponse d'une requête parallèle. Celui-ci est la différence entre l'instant de réception de la dernière réponse et l'instant d'envoi de la requête.

- La taille des buckets est fixée à 100 enregistrements.
- Nous construisons des requêtes [a, b] de différentes longueurs.
- Pour chaque requête [a, b], un certain nombre de buckets $[\lambda_i, \Lambda_i]$ sont concernés.
- Pour chaque requête [a, b] nous déterminons le temps d'exécution.

Pour chaque requête [a,b], nous faisons la moyenne de cinq mesures.

En réalité, nous allons nous intéresser aux temps de réponse selon le nombre de serveurs concernés par les requêtes.

De plus, nous avons eu à calculer le temps de recherche d'un enregistrement qui représente le temps de réponse de la requête [a, b] divisé par le nombre d'enregistrements contenus dans [a, b].

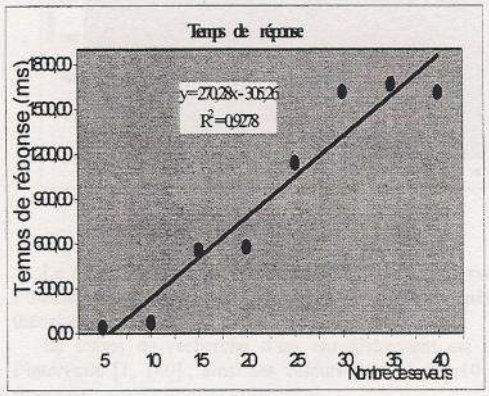
2.3 Environnement expérimental

La plate-forme de tests est constituée de 8 PC Pentium 400 Mhz avec 64 MO de RAM et 450 MO de mémoire virtuelle sur lesquels tourne Windows NT Server 4.0. Les PC sont reliés par un réseau Ethernet 10 MO/s.

Nous avons lancé 140 buckets sur 7 machines distinctes à raison de 7 par PC : les intervalles des buckets sont :]0,50],]50,100],]100,150],]150,200],]200,250],]250,300],...,]6950,7000]. Chaque bucket contient 50 données et chaque PC 20 buckets tournant sur 20 ports. Le client tourne sur PC distinct.

2.4 Résultats

2.4.1 Stratégie N°1



. Fig. n°1 : courbe du temps de réponse

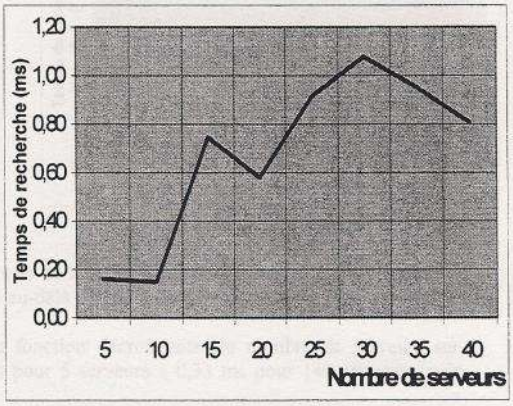


fig. n° 2 : : temps de recherche d'un enregistrement

Pour une requête concernant 5 buckets, les temps de réponse sont tournent autour de 40 millisecondes (ms) et 1612 ms pour 40 serveurs. Pour les mêmes nombres de serveurs, le temps de recherche moyen d'un enregistrement varie de 0,15 ms à 1,08 ms.

Comme on le constate sur la figure n°1, une linéarité existe entre le nombre de serveurs et le temps de réponse ; mais cette dernière disparaît dès que le nombre de serveurs dépasse 40..

2.4.2 Stratégie N°2

Le temps de réponse est de 44 ms pour 5 serveurs et de 4877 ms pour 140 serveurs. On constate que le plafond de 40 serveurs dans la stratégie n°1 est dépassé. On constate aussi une dépendance linéaire entre le temps réponse et le nombre de serveurs. Cette linéarité existe jusqu'à une limite de 140 serveurs.

Le temps moyen de recherche d'un enregistrement en fonction du nombre de serveurs (figure 4) varie de 0,18 ms à 1,02 ms. De plus ce temps est inférieur à 1 ms pour la plupart des mesures effectuées. Au-delà de 100 serveurs le temps de recherche varie très peu à tel point qu'il peut être considéré comme constant.

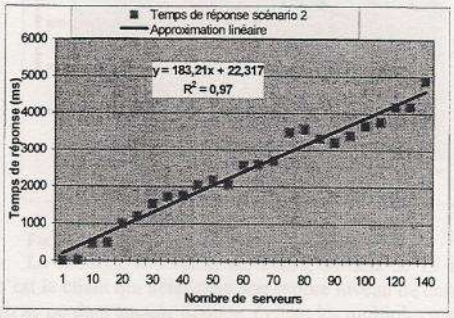


fig. n° 3: Temps de réponse

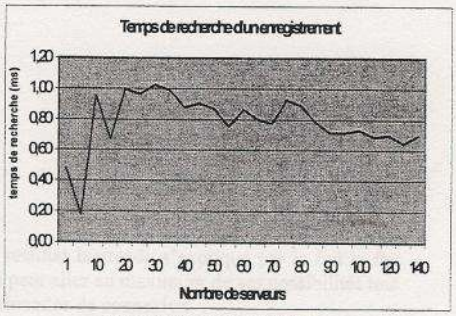


fig. n° 4 : Temps de recherche

Nous constatons qu'une seule tentative de connexion suffit aux serveurs quand leur nombre ne dépasse pas 40, par contre, il faut en moyenne 2 ou 3 tentatives si le nombre de serveurs dépasse 40.

2.4.3 Stratégie N°3 ou stratégie à jeton

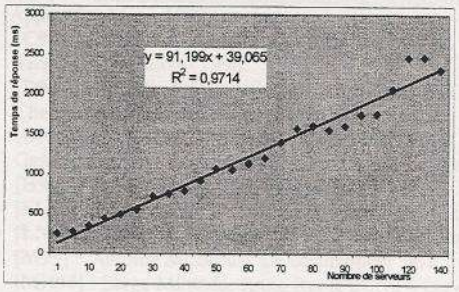


fig n°5 : temps de réponse

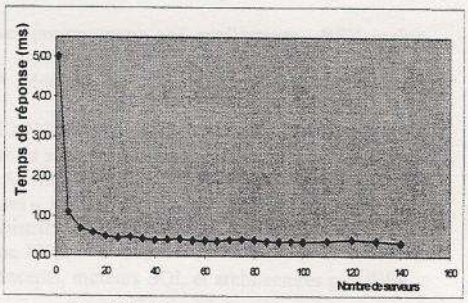


fig. n°6 : Temps de recherche

Le temps de réponse varie de 274 ms pour 5 serveurs et de 2313 ms pour 140 serveurs. On constate que l'inexistence de plafond et la linéarité est constante, même au-delà de 140 serveurs. Cette linéarité est prévisible théoriquement car le client gère lui-même sa charge.

Le temps de recherche d'un enregistrement est une fonction décroissante du nombre de serveurs sur l'intervalle [1, 140] ; ainsi ses valeurs passent de1,10 ms pour 5 serveurs à 0,33 ms pour 140 serveurs (voir figure 6).

3. Conclusion et perspectives

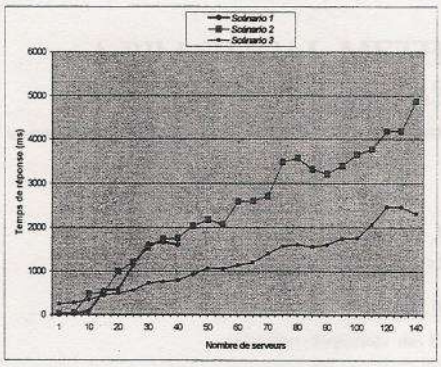


Fig n°7 : temps de réponse pour les troistratégies

La stratégie n°3 est la meilleure, au-delà de 15 serveurs. Ce résultat inattendu s'explique par le fait le fait c'est le client qui détermine son propre niveau de charge. Ainsi, il peut aller au maximum de ses possibilités tout en ne perdant aucune donnée, évitant les multiples tentatives de demandes de connexion.

Enfin, les performances pourraient être améliorées si le client pouvait gérer plusieurs files d'attentes au lieu d'une seule.

Pour chaque stratégie mise en œuvre le principe de scalabilité a été vérifié.

4. Références

- **[AJ] :** " http://www.inf.enst.fr/~charon/coursJava/index.html " Apprendre Java – 1999.
- **[C96]** : Comer, D. E and D. L. Stenvens Client – Server programming and applications , BSD socket version 2 nd edition - 1996.
- **[D97]** : DIENE, A. Organisation interne des SDDS RP*. Mémoire de DEA d'Informatique – Université Cheikh Anta Diop de Dakar – 1998.
- **[DNSNB99]** : DIENE A., NDIAYE Y., SECK T., BENNOUR F. Conception et Réalisation d'un Gestionnaire de Structures de Données Distribuées et Scalables sous Windows NT - 1999.
- **[H98]** : Craig Hunt – O'REILLY & Associates. TCP / IP Administration de réseau – 1998.
- **[FL87]** : J.Fourastié, J.-F Laslier – Probabilités et Statistiques – 1987.
- **[LNS94]** : RP* : A Family of Order-Preserving Scalable Distributed Data Structure – 1994.
- **[M99]** : Antoine Mirecourt. Le développeur Java 2 - Edition 1999.
- **[MR94]** : Serge Miranda, Anne Ruols Client – Serveur Concepts, moteurs SQL et architectures parallèles - 1994.
- **[N97]** : NDIAYE, Y. Protocole de communication SDDS RP*. Mémoire de DEA d'Informatique – Université Cheikh Anta Diop de Dakar – 1998.
- **[PN96]** : Patrick Niemeyer, Joshua Peck - Java par la pratique - 1996.
- **[PSI]** : " http://www.ecst.csuchico.edu/~chafey/prog/sockets/ " Programming the BSD Socket interface .
- **[SNDLL]** : Seck M. T, Ndiaye S., Diene A. W., Litwin W., Levy G. Implémentation des structures de données distribuées et scalables RP*. CARI'98 Dakar Sénégal - 1998.
- **[S96]** : Alok k. Sinha . Networking programming in Windows NT Addison – Wesley publishing company - 1996.

Mots clé : ***scalabilité, requête parallèle, SDDS RP*, multicast, multiordinateur***.