Estudio de actualización de réplicas de datos en entornos de Bases de Datos Distribuidas

A.C. Sergio Len[1]
Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina
slen@lycos.com

Lic. Rodolfo Bertone[2]
LIDI - Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina
pbertone@lidi.info.unlp.edu.ar

Lic. Sebastián Ruscuni[3]
LIDI - Facultad de Informática – UNLP
(1900) La Plata, Buenos Aires, Argentina
sruscuni@lidi.info.unlp.edu.ar

LIDI – Laboratorio de Investigación y Desarrollo en Informática
Facultad de Informática
50 y 115 – La Plata – Buenos Aires
Argentina

Resumen

La capacidad de compartir grandes volúmenes de información a nivel mundial es una realidad y una necesidad de mercado. Cuando se tiene un problema que involucra distribución de información, luego de definir el modelo de datos, se tiene la necesidad de determinar la forma de fragmentación de la información y el volumen de replicación aconsejable para cada modelo junto con la ubicación de las réplicas a lo largo de la red. Estos aspectos son afectados, además, por las técnicas de manejo y actualización de la BDD para determinar la performance final del sistema implementado.

En este proyecto, una tesina de grado de la Licenciatura, avanza sobre las características de replicación y fragmentación de la información, en particular la propagación y regulación de las actualizaciones de datos. El esquema de replicación planteado en este trabajo se considera estático y preestablecido, interesando para este desarrollo el comportamiento de los métodos, en cuanto al tratamiento de lecturas y modificaciones de los datos. Se profundiza el estudio y evaluación de técnicas lazy (perezosas) y eager (on-line), presentado resultados experimentales comparativos entre ambas técnicas, en función del porcentaje de lecturas y actualizaciones y número de replicas.

El soporte de simulación elegido es Java, dado que permite una mayor versatilidad en las construcciones de aplicaciones.

Palabras Claves

Procesamiento distribuido. Base de datos distribuidas. Replicación de datos

[1] Alumno Licenciatura en Informática

[2] Profesor Adjunto Dedicación Exclusiva

[3] Ayudante Diplomado Semi Dedicación

Introducción

Hoy, la mayoría de los negocios deben poder crear y mantener múltiples copias de datos redundantes. Denominamos a esto replicación de datos. Estas copias redundantes son utilizadas para mejorar la performance de las aplicaciones con la disponibilidad de datos en muchos lugares sobre una red de área global. Un aspecto que se debe considerar es como distribuir los datos a lo largo de la organización, optimizando tanto recursos de red como de hardware y tareas administrativas. [1]

La replicación de datos ayuda a la performance desde el punto de vista que requerimientos conflictivos de usuario pueden resolverse de una forma más sencilla, aumentado la disponibilidad de la información. Por ejemplo, los datos que son accedidos por un grupo de usuarios pueden ser ubicados "cerca" de dicho grupo, en lugares de la red que incrementan la referencia local. Más, si una máquina falla, una copia de los datos sigue disponible en otra máquina sobre la red. [2][3]

La decisión sobre que datos deben o no replicarse, y cuantas copias de los objetos de la base de datos deben tenerse, depende considerablemente del tipo de aplicaciones de usuario. Asumiendo que los datos están replicados, debemos garantizar transparencia en el acceso y actualización de los mismos. En todos los casos, el usuario debe actuar como si en el sistema existiera solamente una copia del dato.

Otro concepto relacionado es la transparencia de fragmentación. Esta consiste en la división de la base de datos en pequeños fragmentos y tratar cada fragmento como una base de datos separada. En nuestro problema, los datos consisten de imágenes, podemos clasificar estas imágenes en muchos grupos y podemos efectuar una fragmentación que respete esta clasificación, pero no es posible fragmentar una imagen de dato.

Los datos pueden ser distribuidos a lo largo de los lugares de una red de múltiples formas. Cada dato puede estar completamente replicado, en cuyo caso la base de datos entera es almacenada en cada nodo (máquina). En el otro extremo, una base de datos puede estar completamente fragmentada, en este caso cada dato no se encuentra replicado, sino que residen en diferentes lugares. [4]

Una versión completamente replicada se utiliza cuando conceptos como tolerancia a fallos y performance son importantes, los datos están disponibles aún si un sitio o la red de comunicaciones falla. Por otro lado, se complican las actualizaciones y recuperaciones ante fallos.[5][6]

Una situación con fragmentación completa es común en problemas con múltiples bases de datos donde las bases de datos locales son integradas. Hay otra solución posible, replicación de datos parcial, en definitiva la más utilizada.

El esquema de propagación de actualizaciones influye en la performancedel sistema de BDD. Estos esquemas, sobre los cuales se basa el estudio de esta trabajo, pueden ser Eager o Sincrónicos, en el otro los esquemas Lazy o Asincrónicos. [7]

El Esquema Eager o Sincrónicos conserva todas las réplicas exactamente sincronizadas en todos los nodos, actualizándolas como parte de una transacción atómica, es decir, se aplican actualizaciones a todas las réplicas de un objeto dentro de los límites de la transacción original. En los esquemas Lazy o Asincrónicos, la transacción origen termina localmente sin esperar por el cometido en los otros nodos, propagando asincrónicamente las actualizaciones de la réplica, típicamente como una transacción separada para cada nodo. [8]

Para clasificar los métodos de regulación de actualizaciones, se utiliza un parámetro basado en quien puede realizar las actualizaciones o mejor dicho que copias pueden ser actualizadas. Hay dos posibilidades: los esquemas Master-Slave (actualizaciones centralizadas) y los esquemas Group (actualizaciones distribuidas).

La idea de la opción Master Slave es designar una copia determinada de cada ítem de dato como copia Master, distinguida o también denominada copia primaria, las demás copias son copias Slave o secundarias. El nodo que almacena la copia primaria de un dato es llamado Master para este dato, mientras que los nodos que almacenan copias secundarias son llamados Slaves. El nodo Master del dato , es responsable de realizar las actualizaciones sobre este dato. Si un nodo Slave desea modificarlo, el requerimiento de actualización debe enviárselo al nodo Master, donde se hace el cambio. Cuando una copia primaria es actualizada, el nuevo valor debe ser propagado a todas las copias secundarias.

Los esquemas Group permiten que cualquier nodo pueda realizar actualizaciones sobre sus réplicas locales, logrando tanto acceso de lectura como de escritura sobre los mismos. Cuando una copia es actualizada, el nuevo valor debe ser propagado a las demás.

Motivación y Objetivos del proyecto

En la actualidad, un área de creciente importancia dentro de las Ciencias de la Computación es el relacionado con el procesamiento distribuido sobre arquitecturas heterogéneas, incrementando la eficiencia en la ejecución de un algoritmo global. Dentro del tratamiento de Bases de Datos Distribuidas uno de los temas de interés consiste en el estudio de actualizaciones sobre diversas réplicas de datos existentes en el entorno. Las ventajas que nos brinda la replicación de datos en cuanto a rendimiento y disponibilidad de los datos, en ocasiones se ven empañadas por las actualizaciones ya que el sistema tiene que asegurar que las copias del dato sean actualizadas apropiadamente, lo cual suponen una sobrecarga mayor. Por estos motivos se consideró un punto de estudio muy importante para la replicación de datos. [9][10]

Los objetivos específicos de este trabajo son:

1. Implementar cuatro técnicas de replicación basadas en diferentes esquemas de regulación y propagación de actualizaciones sobre las réplicas.

2. Medir las características de performance de dichas técnicas.

3. Comparar las técnicas evaluando las mediciones de los resultados obtenidos.

4. Decidir que esquema se adapta mejor al cada problema particular, evaluando los beneficios e inconvenientes de la replicación de datos en cada caso planteado.

Implementación de la solución

Componentes básicos de la aplicación

El diseño de la implementación desarrollada, puede describirse con la siguiente arquitectura de procesos y subprocesos. Las figuras 1 y 2 presentan una descripción gráfica de estos elementos. Los procesos y subprocesos utilizados son:

- **Coordinator** (Proceso Coordinator) Se dedica a la carga y distribución de las consultas, también recibe los resultados de las mismas. Esta compuesto por dos subprocesos, el *Manager* y el *Monitor*

    - **Manager** (Subproceso administrador de la carga y distribución de consultas) Es el encargado de tomar cada una de las consultas, determinar sobre que réplica se puede realizar la operación, ya sea local o remota, y enviar la consulta con un Identificador de Réplica al proceso *DataServer(MasterSender).* Es el responsable del procesamiento de las operaciones. También administra el esquema de replicación y toda información de conectividad sobre los sitios.

- **Monitor** (Proceso de Monitorización) Es el encargado de recibir de los subprocesos *SlaveSender y SlaveListener*, los resultados de las consultas, analizarlos, realizar cálculos y escribir en almacenamiento permanente los resultados de la ejecución de la aplicación.

- **DataServer** (Proceso Servidor de Datos) Es el responsable de manejar todos los requerimientos locales de datos que vienen de *Coordinator (Manager)*, y devolver los resultados a *Coordinator (Monitor)*. También manipula los requerimientos remotos de otros sitios. Para esto, esta compuesto por cuatro tipos de subprocesos, el *MasterSender, MasterListener, SlaveSender* y el *SlaveListener.*

  - **MasterSender** (Subproceso Master de Envíos) Es el proceso responsable recibir las consultas que genera *Manager* y delegar dicha tarea a un *SlaveSender* si es una consulta local, en tanto que la envía a un *MasterListener* si es una tarea remota. En este último caso se comunica con un *SlaveListener* para que espere el resultado de la consulta remota. Con los procesos *SlaveSender* o *SlaveListener*, primero se evalúa si hay algún proceso que esté manejando la réplica requerida, en ese caso le asigna la nueva consulta, en su defecto crea uno nuevo. Con esta posibilidad podemos manejar múltiples réplicas simultáneamente.

  - **MasterListener** (Subproceso Master de Recepción) Su función es recibir los requerimientos (sobre los datos locales) provenientes de un *MasterSender* remoto, y distribuirlos a los diferentes *SlaveSender* para que realicen tales requerimientos.

  - **SlaveSender** (Subproceso Slave de Envíos) Se encarga de gestionar todos los requerimientos de datos de una réplica determinada. Una vez que llega un pedido, ya sea local o remoto se comunica con el *DataManager* para realizar la consulta. Luego retorna los resultados a un *SlaveListener* remoto, si fue un requerimiento remoto, o al Monitor, en caso de un requerimiento local.

  - **SlaveListener** (Subproceso Slave de Recepción) Es el proceso responsable de recibir los resultados de las consultas remotas sobre una réplica en particular y enviárselos al Monitor.

- **DataManager** (Proceso Administrador de los Datos) Responsable del almacenamiento y la recuperación de los datos. Cuando un *SlaveSender* requiere una lectura sobre una determinada réplica, éste proceso debe recuperar su valor almacenado en el *DataFile*. De igual manera, ante un requerimiento de actualización escribe los datos en el DataFile y si es necesario propagar tal actualización se comunica con el *Propagator*.

- **Propagator** (Proceso Propagador) Responsable de realizar las propagaciones hacia las réplicas remotas. Recibe las actualizaciones sobre datos locales provenientes del DataManager y las envía a los procesos *Receiver* remotos.

- **Receiver** (Proceso Receptor) Su función es la de recibir mensajes provenientes de los *Propagators* remotos y realizar los pedidos correspondientes de actualización al *DataManager*.

Además de los procesos, existen los siguientes almacenamientos de datos permanentes:

- **DataFile**: es un repositorio de información que almacena los datos involucrados en la aplicación. Cumple la función de simular el almacenamiento local de la BD para toda la réplica.

- **Queries**: contiene las trazas de ejecución de las operaciones a realizar.

- **Results**: mantiene toda la información referente a los resultados de la aplicación.

Circuito de información de una operación

En una *operación de lectura* (Figura 3), el Manager primero determina cuál es el host más conveniente sobre el cual realizar la misma, siendo una copia local el mejor de los casos. Por un lado, estos datos iniciales de la operación son enviados al Monitor para que los grabe en almacenamiento permanente. Además, envía estos datos al MasterSender quien controla si es una lectura local o remota.:

- Si es local otorga el requerimiento al SlaveSender que maneja ésta réplica.
- Si es una consulta remota envía tal requerimiento al MasterListener del host que contiene tal réplica y crea un nuevo SlaveListener (si no existe uno que esté manejando tal réplica) para recibir los resultados remotos. El MasterListener remoto recibe el requerimiento y lo otorga al SlaveSender que maneja ésta réplica El SlaveSender requiere el dato al DataManager para luego ser retornados al SlaveListener remoto.

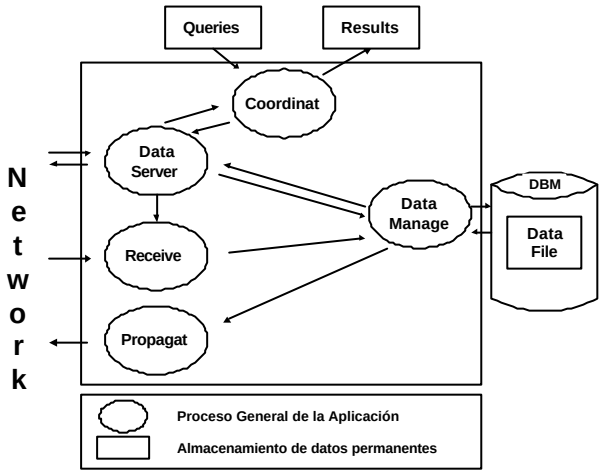


Figura 1. Procesos de la aplicación distribuida

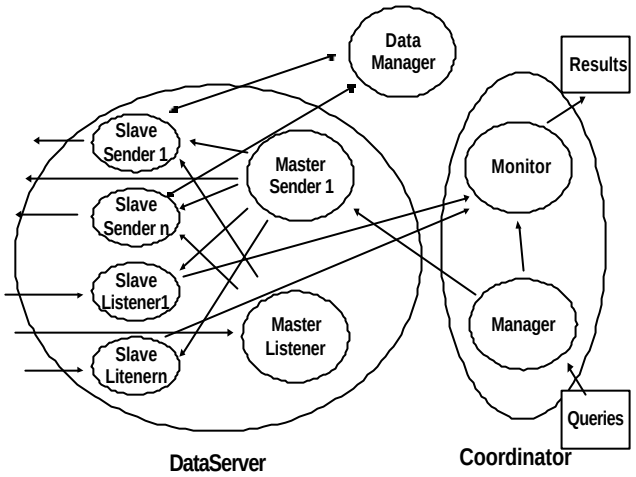


Figura 2. Subprocesos de Data Server y Coordinator

El circuito de información de una *operación de actualización* es similar al de una lectura, con la diferencia que al realiza una actualización local, el DataManager recibe tal requerimiento, y además de realizar la actualización sobre la réplica se encarga de comunicarse con el Propagator para propagar a los demás host remotos tales cambios. Para realizar esto existirán en cada uno de ellos procesos Receivers que aceptarán tales propagaciones y las enviarán a sus respectivos DataManager locales. Si la actualización se debe realizar sobre una réplica remota, ya sea por no poseerla o no ser master de la misma, tal actualización se verá reflejada en la réplica local mediante su propagación (Figura 4).

Proceso de actualización de datos

Los métodos implementados para actualización de datos son Lazy y Eager. Se describe a continuación algunas características de implementación que se utilizaron para la actualización de réplicas.

El esquema Lazy debe tener la capacidad de detectar y solucionar los conflictos que surgen de actualizaciones simultáneas de datos. Para esto se utilizó la técnica de *timestamps* que permite garantizar la propiedad de convergencia entre réplicas. Cada actualización propagada lleva un timestamps de cada réplica original. Si el timestamp de la réplica local viola el orden, existe un conflicto y se necesita algún mecanismo de reconciliación. El mecanismo de reconciliación utilizado es el definido por la regla de escritura de Thomas. [12] El método de actualización Eager utiliza técnicas de bloqueo, mediante las cuales se detectan las anomalías, conviertiéndolas en esperas o bloqueos. Para esto puede utilizar el

esquema de 2PL para bloqueo (two phase locking) y el 2PC para garantizar atomicidad de actualizaciones (two phase commit).

En cuanto a la implementación de los procesos implicados en el diseño de la aplicación no existen diferencias entre los métodos de actualización Group y Master.

Análisis y evaluación de resultados

Se comparan los esquemas de actualización de réplicas implementados: Lazy Master-Slave, Lazy Group, Eager Master-Slave, Eager Group. Los parámetros de evaluación utilizados para el cálculo de resultados quedan fijos, para, de esta forma, poder comparar resultados obtenidos. Los parámetros utilizados fueron: tamaño de la bases de datos (simulada), número de localidades o host en el sistema, número de operaciones en la traza de ejecución, tiempo de comunicación entre localidades, probabilidad de bloqueo de datos (este parámetro es necesario para generar el los casos de bloqueos en un esquema Eager).

Además, existen algunos parámetros "variables", estos son los que definen el tipo de esquema de propagación, y de regulación de actualizaciónes, el número de réplicas y el porcentaje de lecturas. El número de réplicas define un esquema particionado, un esquema parcialmente replicado, o un esquema totalmente replicado.

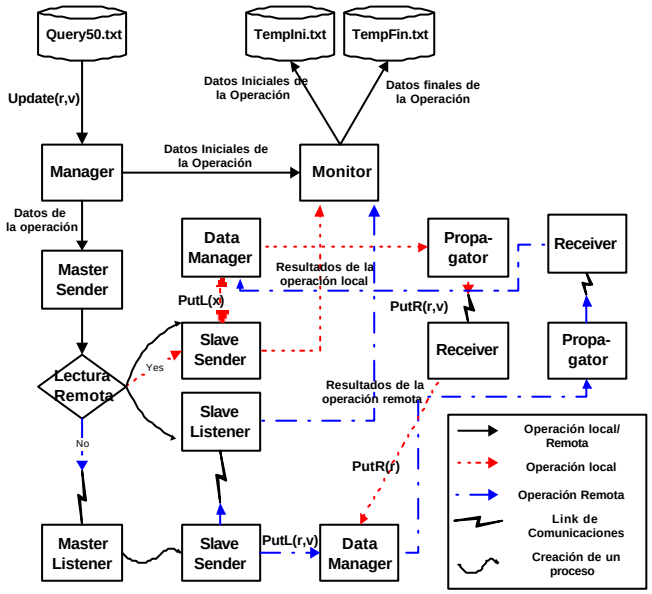


Figura 3. Circuito de información para una lectura

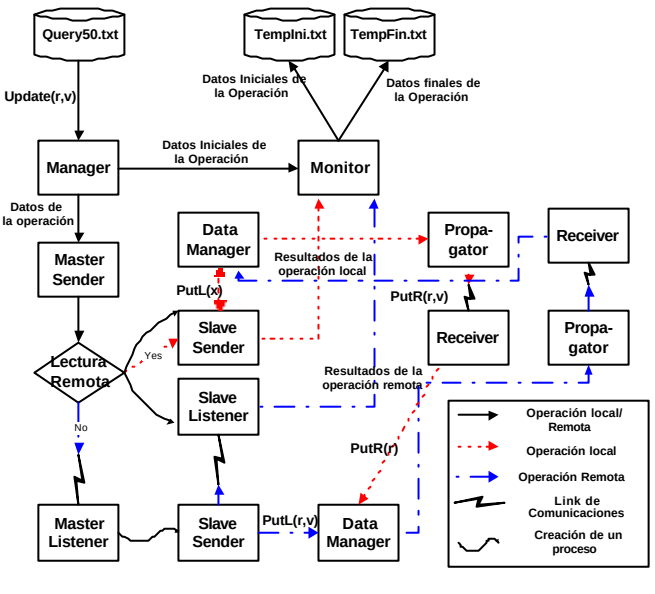


Figura 4. Circuito de información para una actualización

Un índice de evaluación es un valor que caracteriza el modelo durante una ejecución. Estos índices serán usados para realizar las correspondientes comparaciones. En nuestro caso se tuvo en cuenta el tiempo de respuesta y la productividad.

Se mide el tiempo de respuesta de la siguiente forma:

$$Tiempo\_Respuesta(C) = \frac{\sum_{o \in O} Tpo(o)}{\#C}$$

Donde *o* es una operación, *tpo(o)* es el tiempo para procesar una operación, desde que se emite *o* hasta que se recibe el resultado. *O* define un conjunto de operaciones y *#C* define el número de operaciones ejecutadas en la ejecución *C*.

La productividad se define como el número de operaciones cometidas o ejecutadas en una localidad por período de tiempo. De esta forma:

$$Productividad(C) = \frac{\#C}{}$$

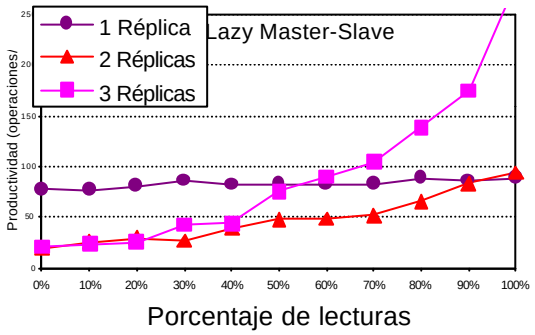
$$\text{Duración de C} = \frac{\sum_{i \in M} \text{Duración de Ci}}{\#M}$$

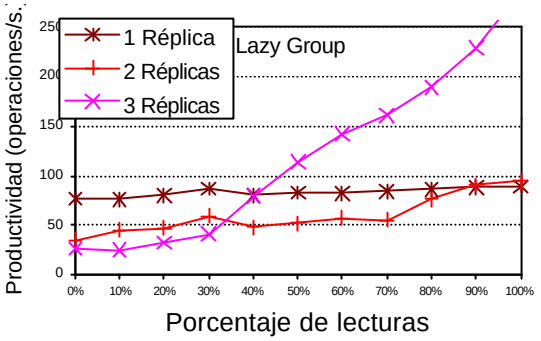
siendo la duración de C

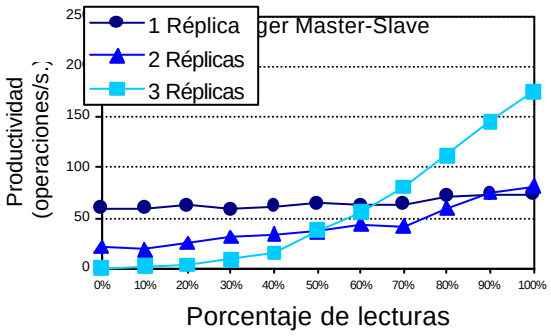
donde *#M* define el número de localidades y *duración de Ci* se define como el tiempo entre la emisión de la primer operación y el momento en el cual se recibe la respuesta de la última operación emitida en la localidad i durante la ejecución C.

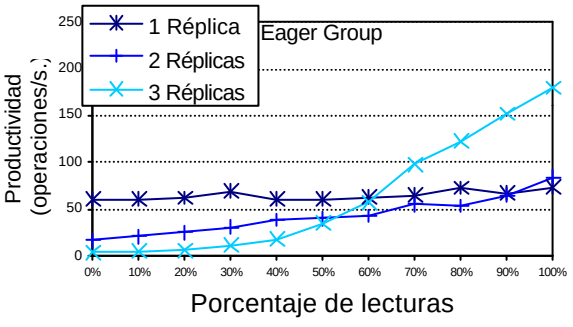
Porcentaje de lecturas

De acuerdo al porcentaje de lecturas en las operaciones sobre la BD y dependiendo del esquema de réplica (1=particionado, 2=parcialmente replicado, 3= totalmente replicado) se obtuvieron los resultados que se presentan a continuación. En todos los esquemas observamos que la productividad para altos porcentajes de lectura es mejor que cuando se trata de casos medios o bajos de accesos. El resultado es esperable dado que una lectura solo requiere el acceso a una réplica solamente, lo cual significa una menor utilización de recursos. El caso de réplica 1, donde solo hay particionamiento, no presenta mejoras ya que no hay replicación.



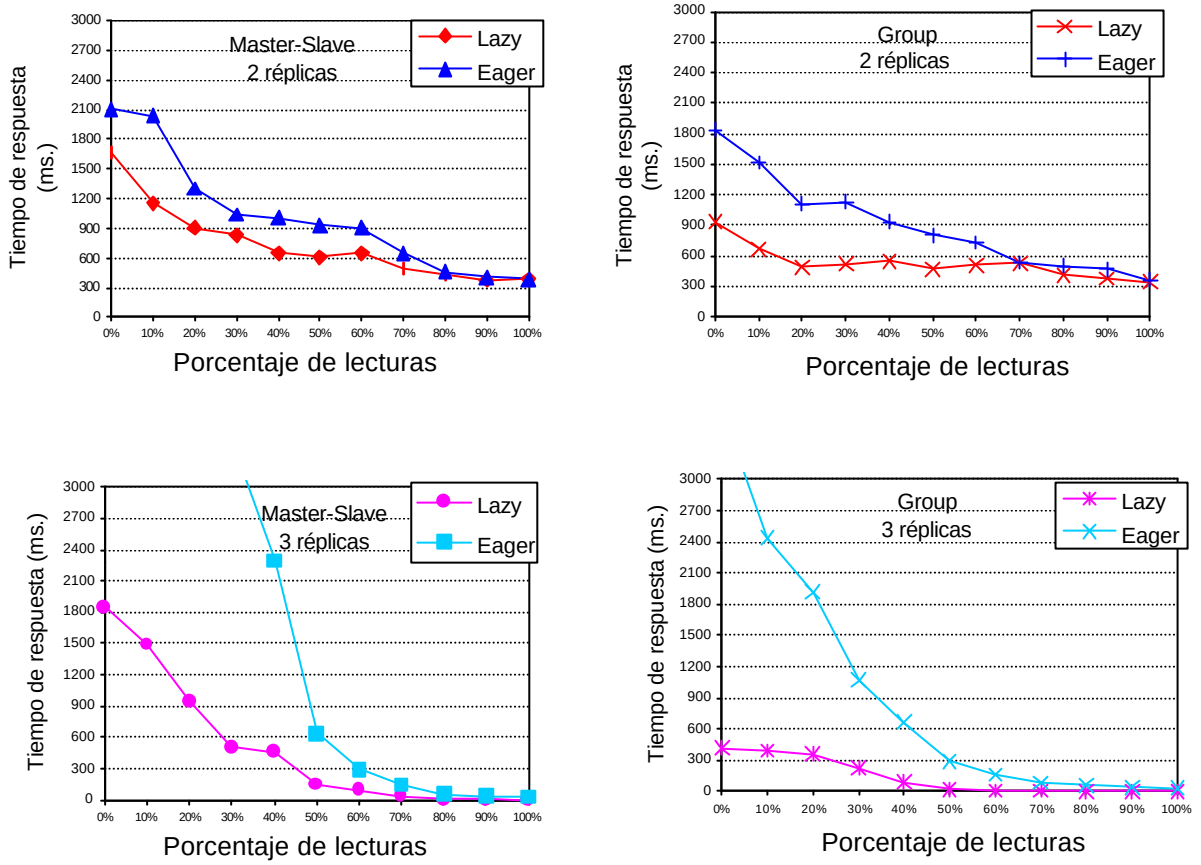






Por idénticas razones se ve afectado el tiempo de respuesta, que disminuye a media que el porcentaje de lecturas aumenta. Sin embargo, en términos relativos la disminución es mucho mayor en el caso de la replicación total, debido a la ejecución de operaciones de lectura de manera local. También se puede observar que los esquemas Lazy tienen un mejor tiempo de respuesta que los sistemas Eager. Esto se hace más evidente al aumentar el grado de replicación y la tasa de actualizaciones debido a que los

esquemas lazy no esperan la sincronización, primero se retorna una respuesta para luego realizar la propagación. En general, los esquemas Group presentaron mejores tiempos de respuesta que los esquemas Master-Slave, debido que los estos últimos necesitan realizar los requerimientos de actualización sobre la copia Master, provocando accesos remotos que incrementan el tiempo de respuesta y producen posibles cuellos de botella. Los resultados se presentan gráficamente a continuación:



Escalabilidad

A mayor número de réplicas mayor es la disponibilidad del sistema, puede ocurrir una sobrecarga en algunos esquemas debido a un aumento de la sincronización entre las mismas. Además, esta sobrecarga puede ser compensada por el balance de carga. La escalabilidad de los esquemas fue analizada variando el número de réplicas, observando los resultados obtenidos tanto con recuperación de información como la actualización de la misma.

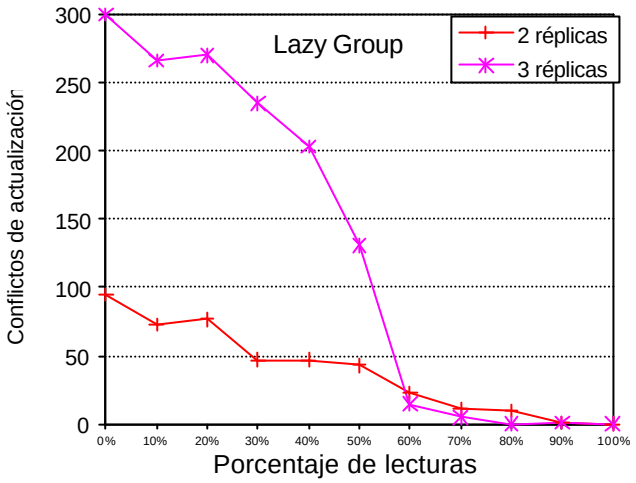
Agregar más réplicas significa aumentar la disponibilidad del sistema, esto redunda en menor tiempo de respuesta para consulta de información. El cuadro anterior presenta un análisis parcial con 2 y 3 réplicas, respectivamente y se observa una baja en el tiempo de respuesta al aumentar el porcentaje de lecturas. En el caso extremo, todas lecturas sin escrituras, la replicación total es indudablemente la mejor aproximación ya que todas las operaciones son locales y no es necesario sincronizar intercambio de mensajes.

Si analizamos ahora el porcentaje de actualizaciones, replicar los datos decrementa la performance; en general, como se desprende de los gráficos anteriores, los tiempos de respuesta son mayores y la

productividad disminuye notoriamente. Además, se observa que los esquemas Eager presentan resultados peores a los Lazy por al forma intrínseca de trabajo.

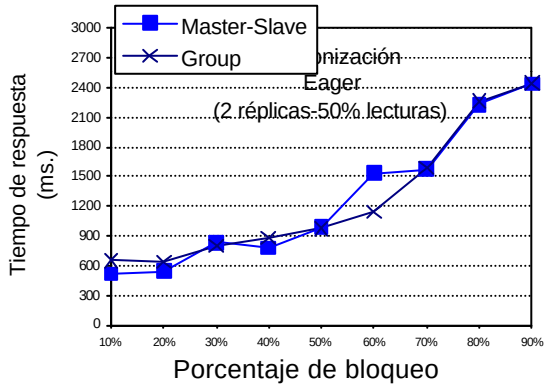
Conflictos de actualización

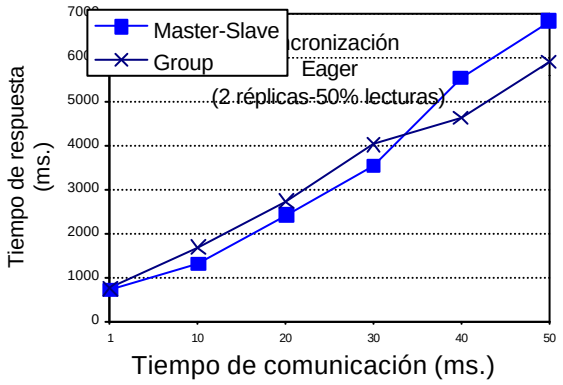
Otro índice que permite evaluar particularmente la escalabilidad de los esquemas Lazy Group es la proporción de conflictos de actualización. Esta proporción disminuye cuando aumenta el porcentaje de lecturas debido a que solo se producen conflictos con las operaciones de actualización. También se observa que los conflictos tienden a incrementarse abruptamente cuando el número de réplicas aumentan debido a la mayor descentralización del procesamiento que se produce.



Sincronización en los esquemas Eager

Se analizan aquí, los tiempos de repuesta de los esquemas Eager, variando los parámetros utilizados para simular la sincronización de los mismos. Ellos son el tiempo de comunicación entre localidades y la probabilidad que un dato en particular se encuentre bloqueado al momento de ser accedido. Los resultados, que se presentan a continuación, indican que tanto al aumentar el tiempo de comunicación entre las localidades como la probabilidad de bloqueos, se produce una degradación de performance para cualquier esquema Eager empleado.





Se han realizado otra serie de pruebas y mediciones sobre el modelo de simulación que no se presentan en este artículo. [11]

Conclusiones

La *Replicación sobre Bases de Datos Distribuidas* es un tema que está aumentando día a día con respecto a su importancia. Los ambientes modernos de computación demandan soluciones innovadoras y mecanismos flexibles que puedan soportar diferentes formas de replicación, para lograr mejoras de disponibilidad y performance en sus sistemas de bases de datos.

Por razones de performance, la mayoría de los productos comerciales e investigaciones han direccionado hacia la utilización de esquemas Lazy o Asincrónicos, reduciendo por supuesto la posibilidad que en un futuro los productos puedan soportar replicación Eager. Sin embargo, esta mejora de performance, se ve empañada por la posibilidad de trabajar con datos inconsistentes, hecho que no ocurre en los esquemas Eager.

En este proyecto, hemos considerado y comparado cuatro esquemas de regulación y propagación de actualizaciones. Direccionamos el problema de actualización de réplicas de datos presentando un modelo para los esquemas de regulación (Group y Master-Slave) y propagación (Eager y Lazy). El comportamiento de esas estrategias fue analizado mediante experimentos prácticos mostrando que la decisión de replicar depende potencialmente de la proporción de lecturas y actualizaciones, pudiendo afectar casi todos los algoritmos y funciones de control de un DBMS distribuido.

Este modelo nos permitió comparar y estudiar las características de performance de cuatro técnicas de replicación basadas en diferentes esquemas de regulación y propagación de actualizaciones sobre las réplicas.

Entre los indicadores de performance, consideramos el tiempo de respuesta, la productividad y los conflictos de actualizaciones. Los experimentos se realizaron variando las proporciones de lecturas y escrituras realizadas en los mismos.

Los resultados muestran que la replicación Lazy tiene una mejor performance que los esquemas Eager y demuestra ser una opción con mayores posibilidades de escalabilidad. Los esquemas replicados brindan mayor disponibilidad y balanceo de las cargas que los centralizados logrando una mayor productividad; y ante la presencia de proporciones bajas de actualizaciones, también otorgan una mejor performance que el caso particionado.

Todos los resultados nos llevan a concluir que el principal beneficio ganado en la replicación de datos está en la disponibilidad para realizar operaciones de lectura localmente, sin ningún tipo de comunicación. Incrementar el número de réplicas tiene el efecto negativo esperado sobre la performance ante altas tasas de actualizaciones y uno positivo ante intensivas lecturas.

Trabajos Futuros

Los trabajos futuros se dividen en dos grupos. El primero de ellos y en función de los resultados obtenidos consisten en realizar nuevas experiencias modificando parámetros de simulación. Se mantiene consistencia floja sobre los datos para el esquema Lazy Master-Slave, y eventual para el Lazy Group, es interesante experimentar grados de consistencias más fuertes entre los datos replicados. En el caso de los esquemas Eager, los trabajos futuros deberán implementar los protocolos de bloqueo que sincronizan la actualización de las réplicas.

También se podría simular un escenario donde ocurran fallas. Una falla puede significar la caída de una o más réplicas y por consiguiente la posible suspensión repentina del procesamiento sobre los datos. Si bien se realizaron algunos trabajos donde se involucró estudio de fallos, es interesante compatibilizar ambas simulaciones y estudiar los resultados. [12] [13]

El segundo grupo de trabajos futuros consiste en compatibilizar este ambiente de simulación con otro existente que permite evaluar diferentes entornos distribuidos para encontrar el mejor esquema de trabajo. Este módulo de simulación permitirá evaluar el tipo de actualización de réplicas incorporando este elemento a los ya existentes en el modelo. [14]

Por último, y en conjunto con otras líneas de investigación, es interesante evaluar un esquema que maneje replicación dinámica de datos y observar el comportamiento de los métodos de actualización en ese entorno.[15]

Bibliografía

[1] M. Buretta, Data Replication. Tools and Techniques for Managing Distributed Information. John Wiley & Sons, Inc. 1997

[2] M. Tamer Özsu, P. Valduriez, Principles of Distributed Database Systems, Second Edition, Prentice Hill, Inc. 1991

[3] D. Burleson, Managing Distributed Databases. Building Bridges between Database Islands, John Wiley & Sons, Inc. 1996

[4] D. Bell, J. Grimson, Distributed Database Systems, Addison Wesley Publishing Company, Inc. 1992

[5] A. Bobak, Distributed and Multi-Database Systems, Artech House, Inc. 1996.

[6] B.Kemme, G.Alonso, A new approach to developing and implementing eager database replication protocols, ACM Trans. Database Syst. 25, 3 (Sep. 2000), Pages 333 – 379

[7] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. IEEE, April 2000.

[8] J. N. Gray, P. Helland, P. O'Neil and D. Shasha. The dangers of replication and a solution. ACM, June 1996.

[9] O.Wolfson, S.Jajodia, Y.Huang, An adaptive data replication algorithm, ACM Trans. Database Syst. 22, 2 (Jun. 1997), Pages 255 – 314

[10] R.Lenz, Adaptive distributed data management with weak consistent replicated data, Official program of the 1996 ACM symposium on Applied Computing, 1996, Pages 178 – 185

[11] Len, S, Bases de Datos Distribuidas. Estudio de Actualización de réplicas de datos. Tesis de Grado. Facultad de Informática. UNLP. 2001.

[12] S. Ruscuni, Estudio de Recuperación de errores en BDD, Thesis, Facultad de Informática, 2000.

[13] S. Ruscuni, R. Bertone, Ambiente para la simulación de distintos ACP y recuperación de errores en Bases de Datos Distribuidas, CACIC 2000, Congreso Argentino de Ciencias de la Computación, Ushuaia, Argentina, Octubre 2000. CD.

[14] S. Ruscuni, R.Bertone, A. Mauriello. Distributed Processing in Replicated Image Data Bases. Efficiency Analysis. ISAS SCI 2001. Orlando USA Julio 2001. Pag 276-280

[15] M.Zanconi, J.Ardenghi, Un protocolo para Replicación Dinámica de Datos, Proccedings IV CACIC, Neuquen, Octubre 1998, pp.797,806.