Replicación Dinámica de Datos

Marcelo Zanconi* Jorge Ardenghi†
Grupo de Investigación en Sistemas Distribuídos
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Av. Alem 1253 - 8000 - Bahía Blanca - ARGENTINA

1 Resumen

En los sistemas de manejo de datos distribuídos, SMDD, se requieren algoritmos de manejo de copias o réplicas, que de alguna manera aceleren el acceso a los datos. En un extremo de replicación se presenta la **replicación total**, donde la *base de datos* está replicada integramente en cada uno de los nodos intervinientes; en otro extremos se presenta el manejo centralizado de datos, tipo cliente-servidor, donde un nodo administra todos los datos. El primero de los casos es poco práctico de implementar. Por otro lado, el manejo centralizado viola la idea de un SMDD donde los nodos tienen no solo independencia de cómputo, sino de datos.

En el medio de estos dos extremos, encontramos la *replicación parcial*, donde un administrador decide que cuerpo de datos van a replicarse y en que nodos lo harán. A partir de este punto, todos los nodos intervinientes conocen el esquema de replicación y se comportan siguiendo algún algoritmo de control de concurrencia tal como copia primaria, token, rowa, etc, [1].

Por otro lado, debemos recordar que todo mecanismo de replicación tendrá éxito comercial si el número de copias es bajo, la performance es buena, (comparada con el manejo centralizado y el almacenamiento redundante), y el número de lecturas se reduce a una copia, tal como se especifica en [7], [2]

Estos métodos son *estáticos*, es decir no cambian con la ejecución de transacciones. Sin embargo, es posible implementar algoritmos *dinámicos* que de acuerdo a la carga del sistema, decidan crear nuevas réplicas o eliminar algunas. Típicamente, cada nodo se encargará de administrar los datos que le son propios, es decir se parte de un esquema de replicación parcial. A medida que el sistema crece, si una copia o residente en un nodo s es requerida remotamente y en forma intensa, desde otro nodo s' es preferible incorporar o a s', de modo que el acceso se haga en forma local. Por el contrario, si una copia o es modificada remotamente y en forma intensa desde otro nodo s', es preferible eliminar del nodo s la adminstración de o, de modo de facilitar el esquema de concurrencia.

En este trabajo, se presenta un algoritmo de control dinámico de réplicas, basado en un trabajo de Huang, [5]

*cczanc@criba.edu.ar
†jra@criba.edu.ar

2 Introducción

La distribución de datos en un SMDD es un decisión crucial de diseño que implica establecer en principio para cada nodo el cuerpo de datos que va a administrar. Normalmente esta decisión de fragmentación lógica y física de los datos, se lleva adelante antes de implementar el SMBDD o bien a medida que se incorporan nuevos nodos a la red en un SMBDD heterogéneo. En el primero de los casos se busca un balance natural del procesamiento entendiendo que una mayoría de transacciones accederán solamente a datos locales y por lo tanto parece acertado que dichos datos estén fisicamente en el mismo nodo. En el caso de sistemas heterogéneos es natural pensar que los nodos que se van incorporando a una red no cederán la administración y control de los datos que mantenían antes de dicha incorporación.

Por otro lado, si la responsabilidad de administración de un dato recae unicamente sobre un nodo, se dispone de un *sistema de procesamiento distribuído*, pero no de un verdadero SMBDD. En este caso se busca *confiabilidad*, *accesibilidad* y *disponibilidad*, [10]. Por lo tanto la idea de replicación es decir de una repetición orgánica de los datos, surge naturalmente. Los mecanismos de replicación apuntan a alcanzar algunos de los objetivos mencionados anteriormente y han sido extensamente estudiados en la literatura, [2], [3], [4], [6].

La solución dada al manejo de réplicas es la de decidir cuales nodos mantendrán simultáneamente la administración compartida de algunos datos. Esta decisión se realiza en forma estática al momento del diseño o de la ampliación de la red, pero no interviene la performance real del sistema, sino que se basa en el comportamiento previsto para las transacciones.

En [5], se presenta un algoritmo de replicación dinámico, es decir uno en el que las copias van a ser creadas y destruidas basadas en la performance del sistema. El objetivo básico será mantener una alta tasa de accesibilidad y un bajo tráfico en la red, entendiendo esto último como el mínimo acceso a copias remoto posible.

Se presentan en los sistemas de cómputo actuales, varias aplicaciones donde la replicación parece ser una buena alternativa: el acceso a páginas de Internet, -objetivo de confiabilidad, sistemas bancarios, -objetivo de confiabilidad y accesibilidad, sistemas de tiempo real, -objetivo de confiabilidad y disponibilidad.

El algoritmo que presentaremos a continuación es una combinación de dos ideas: el manejo de páginas caché, -idea tomada de los sistemas operativos-, para minimizar el acceso a disco y el manejo dinámico propuesto por Huang y Wolfson, [5].

Conceptualmente, el algoritmo funciona midiendo la performance del sistema a través de las operaciones de lectura y escritura. En este sentido, decimos que el algoritmo se basa en un criterio sintáctico para determinar dicha performance. Para ello:

- Si durante un período de tiempo se observa que se inician en un nodo operaciones de lectura de datos remotos, es más efectivo replicar el dato localmente. Es decir, privilegiamos el costo de acceso sobre el costo de almacenamiento, tratando de bajar el tráfico en la red, a costa de mayor espacio de almacenamiento local.

- Si durante un período de tiempo se observa que llegan operaciones remotas de escritura a una copia que se mantiene localmente, es más efectivo eliminar dicha copia. En este caso, privilegiamos el costo de acceso y el costo de almacenamiento.

Concretamente, si pensamos que una réplica que mantiene con el objetivo de lograr mayor confiabilidad y accesibilidad, también debemos pensar que esta operación tiene su costo. Por un lado, el esquema estático impone que todas las réplicas se modifiquen ante una escritura,

pero si pensamos que dicha copia existe para que localmente se facilite la lectura, entonces observamos que:

- Si el número de accesos locales a un dato es bajo o nulo, -lo que equivale a decir que habrá varios accesos remotos de dicho dato para garantizar la consistencia de copias-, no tiene sentido mantener esta réplica y
- Si el número de accesos remotos es grande, -lo que equivale a decir que habrá un intenso tráfico en la red-, vale la pena generar localmente una copia.

De este modo, el algoritmo va cambiando el esquema de replicación, sujeto a que nunca debe reducirse el número de copias por debajo de un dado valor k, -típicamente 2. Tengamos en cuenta que este valor k es un multiplicador del espacio de almacenamiento y no existen demasiadas BDD que resistan una replicación más intensa.

Las tablas de replicación cambiarán dinámicamente, al tiempo que el esquema de replicación se modifica. Para ello, cuando un nodo s decida crear una copia en otro nodo s', se compromete a propagar sus escrituras hacia s'. Inversamente, cuando un nodo s decida eliminar una copia de un dato, informará al resto de los nodos de esta novedad.

Observemos que el algoritmo viola en algún sentido el principio de autonomía de los nodos al ordenar a otro que cree una copia. Esta violación se ve compensada por una mejor performance del sistema.

3 El algoritmo

Los nodos intervinientes en el esquema de replicación pueden ser clasificados como *nodos de datos* y *nodos ajenos a los datos*. Los primeros guardan réplicas de los datos, mientras que los segundos no. Debe tenerse en cuenta que un nodo es de datos para un grupo de datos, pues hablamos de replicación parcial y no total.

De los nodos de datos, se escoge un nodo p como *coordinador*, de modo que si un nodo efectua una operación *r(x)*, será atendida localmente, -si el nodo es de datos, o remotamente por p; si se efectúa una operación *w(x)*, ésta es atendida por p y luego se propaga al resto de los nodos de datos.

Cada nodo mantiene varios datos internos:

- *Ed*: un bit para saber si es un nodo de dato, (1), o nodo ajeno, (0), para un dato x.
- *r-counter*: contador de las operaciones de lecturas emitidas por el nodo.
- *w-counter*: contador de las operaciones remotas de escritura que recibe el nodo.
- El nodo coordinador mantiene los contadores para los nodos ajenos j: *r-counter*$_j$ y *w-counter*$_j$ y Ed_j, permitiéndole incorporarlos al esquema de replicación.

El nodo coordinador tomará esta información para sumar un nodo ajeno j al esquema si recibe muchas operaciones de lectura de j mientras que si recibe muchas operaciones de escritura de nodos distintos que j, eliminará a j del esquema de replicación.

Los pasos del algoritmo son los siguientes, (se asume un parámetro k que expresa la frecuencia de actualización del esquema de replicación):

1. Cuando un nodo de datos i, emite una operación de lectura incrementa su *r-counter*. Si éste alcanza un valor k, se reinicializan los *r-counter* y *w-counter* a 0,

2. Cuando un nodo ajeno j emite una operación de lectura, el nodo coordinador p incrementa el $r\text{-}counter_j$ y si dicho contador alcanza el valor k, p envía una copia del dato a j y un mensaje invitándolo a sumarse al esquema de replicación. Para ello cambia Ed_j de 0 a 1, (para indicar que ahora j es un nodo de datos).

3. Cuando un nodo de datos i recibe una operación de escritura de p, incrementa su *w-counter*. Si éste llega a k, i decide eliminarse del esquema de replicación siempre y cuando el número de nodos en el esquema sea no menor a m; en este caso, inicializa su Ed a 0 e indica a p que cree contadores para i.

4. Cuando el nodo coordinador p recibe una escritura de un nodo i, propaga este valor a todos los nodos de datos e incrementa el $w\text{-}counter_i$. Si éste alcanza el valor k, p reinicia estos contadores a 0. Si el propio contador *w-counter* de p alcanza el valor k, p decide eliminarse del esquema de replicación, eligiendo un nuevo nodo:

   (a) Si existen al menos m nodos en el esquema de replicación, p se elimina y elige a otro s para cumplir esas funciones.

   (b) De otro modo si existen al menos m nodos en el esquema de replicación, contando a i, p indica a i que asuma sus funciones.

   (c) Elegido el nuevo nodo coordinador, s, p cambia Ed a 0 e indica a s que inicie nuevos contadores para p, enviándole todos los contadores que mantenía.

   (d) p envía un broadcast a todos los nodos en la red para indicar que s es el nuevo coordinador.

El parámetro k mide el número de accesos al dato. Si un nodo N emite k accesos remotos de lectura, el algoritmo decide incorporar el nodo N al esquema de replicación. Si se accede a N para completar k accesos remotos de escritura, entonces el algoritmo decide eliminar N del esquema de replicación.

El parámetro m, -que en realidad podría ser una constante definida en el momento de diseño-, indica el número de nodos en el esquema de replicación. Por razones de confiabilidad, -resistencia a las fallas-, se requiere que $m > 1$.

4 Extensión del Algoritmo

En esta sección extenderemos el algoritmo anterior para manipular un esquema de replicación basado en el modelo ROWA, read-one, write-all, [1]. En este modelo, las operaciones de lectura se realizan a un nodo, mientras que las operaciones de escritura se propagan[1] a todos los nodos.

En lugar de centralizar el control del dato en un nodo coordinador, distribuiremos el control de replicación entre todos los nodos. Cada nodo deberá mantener los siguientes datos:

1. Los nodos que intervienen en el esquema de replicación, N_i

[1] broadcast

2. Contadores *r-counter* y *w-r-counter* y *w-l-counter*. El primero registra las operaciones de lectura remotas; el segundo las operaciones de escrituras remotas[2], y el tercero las operaciones de escritura locales[3].

El algoritmo sigue los siguientes pasos:

1. Cuando un nodo de datos i, emite una operación de lectura incrementa su *r-counter*. Si éste alcanza un valor k, se reinician los *r-counter*, *w-r-counter* y *w-l-counter* a 0.

2. Cuando un nodo ajeno j emite una operación de lectura, incrementa su *r-counter*; si dicho contador alcanza el valor k, entonces j se suma al esquema de replicación:

   (a) Solicita una copia a algún nodo en N,

   (b) Cuando accede a ella, cambia N_j a 1 y propaga esta novedad al resto de los nodos N_i. Inicia sus contadores a 0.

3. Cuando un nodo de datos i emite una operación de escritura, incrementa su *w-l-counter*. Si éste llega a k, lo reinicializa en 0.

4. Cuando un nodo de datos i recibe una operación de escritura remota, incrementa su *w-r-counter*. Si éste llega a k, y *r-counter* y *w-l-counter* están en 0, i decide eliminarse del esquema de replicación siempre y cuando el número de nodos activos en N sea no menor a m; en este caso, inicializa su N_i a 0, propaga esta novedad al resto de los nodos en N e inicializa sus *w-r-counter* a 0.

En este algoritmo se establecen relaciones entre los contadores. Si una copia existe solo a los fines del protocolo de replicación, se la decide eliminar siempre y cuando queden suficientes copias como para que el sistema continúe confiable. Por ello si el *w-r-counter* crece y no se actualizan el *r-counter* y *w-l-counter*, este nodo se elimina del sistema.

Si por otro lado, un nodo requiere accesos remotos insistentemente, decidirá sumarse al protocolo de replicación.

5 Manejo de Concurrencia

El control de concurrencia para el primer algoritmo puede realizarse facilmente exigiendo que toda operación de escritura se apropie exclusivamente, *x-lock*, de todas las copias disponibles de un dato y cada operación de lectura se apropie en forma compartida, *s-lock*, de la copia primaria o local de un dato.

Este esquema ROWA preserva la propiedad de "serializabilidad de una copia", [1]. Observemos que cuando una transacción retiene un lock sobre un dato, la visión del esquema de replicación se congela desde ese punto y hasta la terminación de la transacción, *punto de commitment* o hasta la emisión de la operación *unlock* respectiva, pues ninguna otra transacción activa puede cambiar este esquema.

En el segundo algoritmo, la aplicación de este esquema ROWA puede ser basta deficiente desde el punto de vista de mensajes. Resulta más apropiado un esquema de quorum de lectura y escritura, [9], que permite un manejo dinámico de las copias necesarias. Básicamente, debe

[2] write remote counter
[3] write local counter

tenerse en cuenta que el concepto de "quorum" es variable de acuerdo al número de nodos en el esquema de replicación, (dado por la estructura N_i, mantenida por cada nodo), pero este dato es *constante durante la ejecución de una transacción*, pues ninguna transacción puede incorporarse o eliminarse del sistema de replicación.

6 Conclusiones

Se han presentado algoritmos para el control de réplicas dinámico. Estos algoritmos mejoran el protocolo estático de creación de copias a medida que el sistema lo requiere. Se busca confiabilidad y menor tráfico en la red.

El primer algoritmo dinámico presenta las desventajas de todo sistema centralizado en un ambiente dinámico: cuello de botella al coordinador, intenso tráfico de mensajes hacia un nodo, baja resistencia a las fallas. Es fácil de implementar y adaptable a diferentes protocolos de manejo de concurrencia.

El segundo algoritmo aumenta la resistencia a las fallas, aunque no es económico, en el sentido de que existe un alto tráfico en la red, cuando un nodo se incorpora o elimina del protocolo de replicación, sin embargo existe una compensación al favorecer la creación de copias locales, disminuyendo la transferencia de datos, -tengamos en cuenta que normalmente un mensaje de control es mucho más pequeño que un mensaje de datos, sobre todo en las BD actuales con tipos de datos no estándares, [8].

Por otro lado, este algoritmo complica el manejo de concurrencia al requerir un mayor número de copias a mantener en forma consistente, además de aumentar el espacio de almacenamiento local. Los protocolos más apropiados son los del tipo votos en mayoría.

Referencias

[1] P.A. Bernstein, N.Goodman. *Concurrency control and recovery in database systems.* Addison-Wesley. 1987.

[2] P.A. Bernstein, N. Goodman. *An algorithm for concurrency control and recovery in replicated distributed databases.* ACM-TODS, Vol 9, N. 4, pg 596-615. Diciembre 1984.

[3] M. Carey, M. Stonebraker. *The Performance of Concurrency Control Algorithms for Database Management Systems.* Proceedings of the 10th VLDB Conference. Singapure. Agosto 1984.

[4] El Abbadi, D. Skeen, F. Cristian. *An efficiente, Fault-Tolerant Protocol for Replicated Data Management.* Proceedings of the 4th ACM Symposium on Principles of Database Systems. Portland. Marzo 1985.

[5] Y. Huang, O. Wolfson. *A competitive Dynamic data replication algorithm* Proceedings of the 9th International Conference on Data Engineering. Austria. Abril 1993.

[6] M. Stonebraker. *Concurrency Control and Consistency of Multiple Copies of Data in Distributed* INGRES. IEEE Transactions on Software Engineering. Vol. 5, N. 3. Mayo 1979.

[7] M. Stonebraker. *Readings in Database Systems.* Morgan and Kaufman Publishers. 1984.

[8] J. Ullman. *A First Course in Database Systems*. Prentice Hall. 1997.

[9] R. Alonso, D. Barbar, H. García Molina. *Quasi-copies: efficient data sharing for information retrieval systems*. Proc. of the EDBT'88, Springer-Verlag.

[10] M. Zanconi *Análisis de Replicación en un Sistema Distribuido*. Tesis de Magister. Departamento de Ciencias de la Computación. Universidad Nacional del Sur. Diciembre 1996.