Un Protocolo para 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
{cczanc,jra}criba.edu.ar

1 Resumen

La replicación es un mecanismo utilizado para favorecer la tolerancia a las fallas y la disponibilidad de los sistemas de manejo de datos distribuídos, SMDD. Se puede tener **replicación total**, donde la *base de datos* está replicada integramente en cada uno de los nodos intervinientes; en otro extremo se presenta el **manejo centralizado de datos**, tipo arquitectura cliente-servidor, donde un nodo administra todos los datos. El primero de los casos es poco práctico de implementar dado el alto costo de actualización que tiene. Por otro lado, manejar los datos enn forma centralizadas hace al SMDD muy vulnerable y tiene un alto grado de estrangulamiento.

Entre estos dos extremos, se tienen la **replicación parcial**, donde un administrador decide qué conjunto de datos van a replicarse y en que sitios 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, entre otros, [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] y en [2]. Pero 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 sitio 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 C residente en un nodo s es requerida remotamente y en forma intensa, desde otro nodo s' es preferible incorporar C a s', de modo que el acceso se haga en forma local. Por el contrario, si una copia C es modificada remotamente y en forma intensa desde otro nodo s', es preferible eliminar del nodo s la administración de C, de modo de facilitar el esquema de concurrencia.

Por otro lado, los agentes permiten la toma de una decisión basada en la percepción del sistema; en este caso no interesa un agente que tenga una arquitectura especial de percepción, (como se especifica en la literatura, [12]), sino que el agente pueda captar el estado del sistema de acuerdo a ciertos parámetros predefinidos. Concretamente, el agente actuará como un controlador de las versiones dinámicas de un dato, en un modo similar al expuesto en [11], pero con mayor interacción con el ambiente distribuído.

Palabras claves: replicación, replicación dinámica, sistemas distribuídos.

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 Sistema de Manejo de Bases de Datos Distribuídas, 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, donde las copias son creadas y destruidas en base al desempeño del sistema. El objetivo básico es mantener una alta tasa de accesibilidad y un bajo tráfico en la red, entendiendo esto último como el mínimo acceso a copias remoto posible.

En los sistemas de cómputo actuales se presentan 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).

En [11] se presentó un algoritmo de replicación dinámica que permitía evaluar la necesidad de creación, mantenimiento y eliminación de réplicas basado en la evolución del sistema. Este algoritmo distribuído tenía una componente local, (para resolver la necesidad de réplicas) y una componente distribuída, (para resolver la consistencia de las réplicas).

En este trabajo presentamos un protocolo basado en tres ideas: el manejo de páginas caché, -idea tomada de los sistemas operativos-, para minimizar el acceso a disco y el manejo dinámico propuesto en [11] y la distribución de cargas de un sistema.

El protocolo está basado en el desempeño del sistema a través de las operaciones de lectura y escritura, es decir, se trata de un protocolo sintáctico. 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, se privilegia 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 es-

critura a una copia que se mantiene localmente, es más efectivo eliminar dicha copia. En este caso, se privilegia 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. Se tiene 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.

Se observa 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 un mejor desempeño del sistema.

3 Descripción del Protocolo

Tal como se definiera en [11], el sistema está compuesto por nodos que bajo la óptica de la 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 el esquema obedece a 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: $r\text{-}counter_j$ y $w\text{-}counter_j$ 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 Protocolo

El protocolo anterior tiene una desventaja que es el uso de un coordinador para el manejo de copias. Este tipo de arquitectura no resulta apropiada para las fallas ni para la accesibilidad, es decir desde el punto de vista de la comunicación se carga demasiado el sistema sobre un nodo y desde el punto de vista de confiabilidad no se toleran fallas.

Para solucionar estos inconvenientes, se presenta una extensión a dicho protocolo donde los propios nodos intervinientes deciden acoplarse o eliminarse del esquema de replicación. Para tomar estas decisiones, cada nodo cuenta con un *agente de replicación* que mantiene datos sobre la evolución del sistema local y naturalmente sirve de nexo con el sistema distribuído.

El agente maneja una serie de parámetros e información local sobre el sistema:

1. Los nodos que intervienen en el esquema de replicación, N_i
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, (write remote counter), y el tercero las operaciones de escritura locales, (write local counter).
3. Los parámetros
   (a) ξ: que representa la longitud de la copia,
   (b) τ: que representa el tamaño del paquete de transmisión de datos,
   (c) los umbrales u_{min} y u_{max} que representan la tolerencia del agente para crear o eliminar copias.
   (d) t: unidad de tiempo.

El protocolo sigue los siguientes pasos:

1. Cuando un nodo de datos i, emite una operación de lectura para un dato x en el nodo, incrementa su *r-counter(x)*. Si éste alcanza un valor k, se reinician los *r-counter(x)*, *w-r-counter(x)* y *w-l-counter(x)* a 0.
2. Cuando un nodo ajeno j emite una operación de lectura sobre un dato x, incrementa su *r-counter(x)*. Se testea la condición

$$\frac{\text{r-counter} \times \xi}{t} > 1$$

   que establece el criterio para crear una copia. Típicamente, si el número de lecturas para ese dato es muy grande, con respecto al tiempo, es conveniente crear una copia local. Si el número de lecturas no es grande, pero el tamaño del dato es importante, entonces el tráfico en la red aumenta y por lo tanto también es conveniente crear una nueva copia local. En cualquiera de los casos, el agente de replicación en j decide sumarse 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 para un dato x, 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(x)*. Si éste llega a k, realiza el siguiente test:

$$\frac{\text{w-r-counter} \times \frac{\xi}{\tau} \times |N|}{t} < u_{max}$$

   y comprueba asimismo que el número de nodos activos en N sea no menor a m. Si ambas condiciones se cumplen el nodo i decide eliminarse del sistema y para ello:
   (a) Inicializa su N_i a 0, propaga esta novedad al resto de los nodos en N,
   (b) Inicializa sus *w-r-counter* a 0.

   Este test tiene en cuenta el número de accesos al dato, el costo de transmisión, (especificado en número de paquetes de mensaje) y naturalmente, el número de copias para dicho dato.

5. Cuando un nodo ajeno j emite una operación de escritura para un dato x, el agente de replicación en j realiza el siguiente test:

$$\frac{\text{w-r-counter} \times \frac{\xi}{\tau} \times |N|}{t} > u_{min}$$

Si es test es positivo, entonces el nodo j decide acoplarse al esquema de replicación, realizando las tareas mencionadas en los apartados 2a y 2b.

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.

Si por otro lado, un nodo requiere accesos remotos insistentemente en el tiempo, decidirá sumarse al protocolo de replicación. Sin embargo, es posible que un proceso requiera al principio de su procesamiento varias operaciones de escritura, con la consecuente modificación del *w-r-counter(x)* y luego el sistema no requiera dicho dato. Si no se ajusta convenientemente este parámetro es posible que se tomen decisiones no apropiadas. Para ellos, los contadores sufrirán un mecanismo de envejecimiento[1] tal como es común en los sistemas distribuídos, de modo que si un dato no se actualiza por mucho tiempo, su valor vaya decrementándose.

Los umbrales definidos de tolerencia a la replicación pueden ser fijos en el sistema, (tal como lo hemos supuesto en este esquema), es decir que quedan especificados estáticamente en base a las capacidades de cada uno de los sistemas intervinientes o bien pueden ser dinámicos de acuerdo a criterios tales como espacio disponible, medición del tráfico en la red, costo de las comunicaciones, entre otros.

El protocolo presentado trata de medir y balancear dos costos, que en cierta medida son dicotómicos: *costo de comunicación* y *costo de procesamiento caché*.

5 Manejo de Concurrencia

Resta definir a continuación como actuará el planificador de concurrencia para mantener la consistencia de las copias. No debe confundirse en esta arquitectura el rol del agente, que actuará sobre la necesidad de crear o eliminar copias y la del planificador que estará encargado de manejar concurrentemente las réplicas de un dato.

Naturalmente, ambos sistemas interactúan. El agente afecta al planificador al crear o destruir copias y por lo tanto deberá informar al manejador de estas novedades para que éste adapte sus algoritmos de consistencia. A su vez, el planificador afectará al agente, pues una vez que el primero haya satisfecho el test de consistencia, podrá el agente actuar preventivamente creando o eliminando copias.

Básicamente, los sistemas pueden actuar independientemente compartiendo las estructuras comunes. En primer lugar, el planificador de concurrencia debe mantener un esquema de replicación, que en lugar de ser estático como en la mayoría de los sistemas provistos comercialemente, será dinámico. Esta estructura será compartida con el agente. Cuando una transacción solicita una operación sobre un dato x, el scheduler consultará la tabla de esquema de replicación:

1. Si el nodo es local para ese dato x y el test de concurrencia es satisfecho, se accede al dato, modificando el permiso adecuadamente en este nodo y en todos aquellos donde exista una réplica.

[1] aging

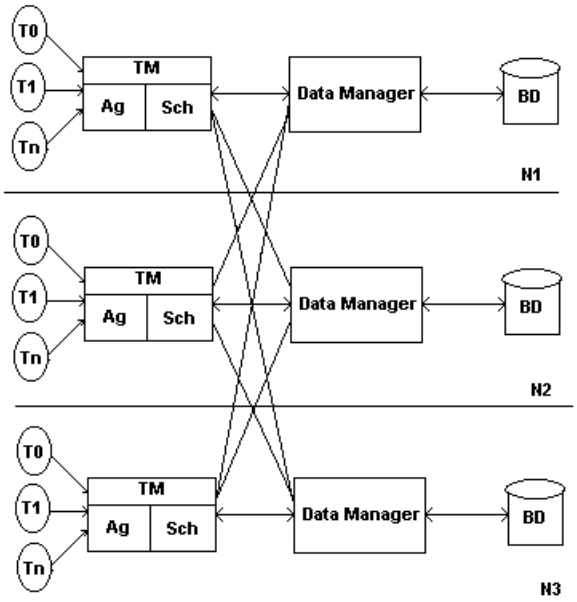

Figura 1: Arquitectura General del Sistema

2. Si el nodo es ajeno para x y el test de concurrencia es satisfecho, el agente podrá analizar la posibilidad de crear una nueva copia. En este caso, se ajustará la tabla esquema de replicación.

El test de concurrencia puede ser cualquier mecanismo apropiado para mantener consistencia en varias copias tal como: ROWA, [1], esquema de votos en mayoría o un esquema de quorum de lectura y escritura, [9], que permite un manejo dinámico de las copias necesarias.

El esquema ROWA preserva la propiedad de "serializabilidad de una copia", [1], basado en *locks*, donde se distinguen *locks compartidos*, (para simultáneas operaciones de lectura) y *locks exclusivos*, (para operaciones de escritura). En el primer caso es suficiente con retener una copia en forma compartida y para el último es necesario retener simultáneamente todas las copias en forma exclusiva.

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 cuanto al mecanismo basado en "quorum", debe tenerse en cuenta que este concepto 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.

En la figura 1 se muestra la arquitectura general del sistema y la interacción entre el agente, el scheduler y los nodos del sistema.

El manejador de transacciones TM, ha sido enriquecido un el software para implementar el protocolo del agente. Las estructuras de datos comunes son mantenidas por este manejador. Cada TM interactúa con el administrador de datos DM local y los foráneos. Esto es así debido a que una transacción generada en un nodo s puede acceder a datos replicados en otro nodo. A su vez, los DM deben interactuar con los TM por resolución del agente que decide crear una copia, en cuyo caso deberá propagar el dato desde un nodo foráneo al local. Finalmente, la BD es accedida unicamente a través del DM local, con lo

cual se preserva la autonomía, (existe un unico nodo responsable del mantenimiento de un dato), pero también se alcanza la cooperación, (ya que otros nodos pueden demandar un cambio en un dato).

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. Barbará, 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.

[11] M. Zanconi, J. Ardenghi. *Réplicación Dinámica de Datos*. CACIC 97. Universidad Nacional de La Plata. Departamento de Informática.

[12] Russell, S y Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice-Hall. 1995.