Clusters de Datos en Ambientes Móviles

Marcelo Zanconi Jorge Ardenghi
Laboratorio de Investigación en Sistemas Distribuídos
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Av. Alem 1253 - 8000 - Bahía Blanca - ARGENTINA
{mnz,jra}@cs.uns.edu.ar

Resumen

Los sistemas móviles son aquellos sistemas distribuídos donde se incorpora el uso de dispositivos de trabajo que migran de locación. Los sistemas trabajan autónomamente hasta que requieren el acceso a datos compartidos. Estos datos pueden ser alcanzables con relativa facilidad, (por ejemplo en sistemas fijos, permanentemente conectados) o deben usarse datos en cache. La localización de los datos debe ser una tarea sencilla y que aproveche al máximo el escaso tiempo de conexión de las estaciones móviles. En este trabajo se presentan algunas especificaciones y soluciones para estrategias y técnicas discutidas en un trabajo anterior, [12]. Concretamente, se presentaran alternativas para el control de la concurrencia en un ambiente distribuído móvil, manejo de replicación de datos y modos de trabajo (conectado, debilmente conectado y desconectado), a través de la introducción de una nueva idea que es el *cluster de datos*.

Palabras claves: sistemas distribuídos, ambientes móviles, replicación, control de concurrencia.

1 Introducción

En esta sección daremos algunos conceptos sobre ambientes móviles, basados principalmente en los trabajos de [8], [1] y [12].

Los sistemas móviles son aquellos sistemas distribuídos donde se incorpora el uso de dispositivos de trabajo que migran de locación; los sistemas distribuídos clásicos contienen un numero de estaciones de trabajo fijas cuya locación puede ser prevista por algún software *à la DNS*. Por el contrario, en estos sistemas se incorporan estaciones poderosas, tales como laptops que demandan servicios del mismo modo que las estaciones fijas, con algunos puntos en contra tales como: desconectividad frecuente, inaccesibilidad, ancho de banda reducido, locación migratoria, costo de comunicación. La idea final es brindar la misma calidad de servicio, independientemente del modo de conexión a la red.

Los sistemas móviles se conectan en forma no cableada o inalámbrica a la red, principalmente a través de conexiones satelitales o comunicación telefónica. Necesitan trabajar autónomamente para soportar operaciones en forma desconectada y a su vez dependen fuertemente de la infraestructura de la red.

Para el acceso a los datos, ya se ha establecido que el esquema apropiado es el de replicación combinado con múltiples versiones, [1] que soporta autonomía, implica propagación de las actualizaciones cuando el sistema está en modo conectado y aumenta la performance y la disponibilidad de los datos, [6].

El manejo de replicacion de datos ha sido extensamente estudiado. En [12] se expresa que los mecanismos apropiados para el manejo de replicación eran aquellos basados en quorums variables o dinámicos. Los primeros fijan un número de accesos menor al número total de copias, mientras que los segundos varían el quorum de acuerdo a la evolución del sistema.

Los sistemas móviles funcionan muy bien cuando se los ve como clientes de algunos servidores, pero una estación móvil está dedicada principalmente a un único usuario y tal vez sea ésta su única máquina, por ello, es deseable asumir a estos usuarios como servidores, por ejemplo, de algunos datos propios o de aplicaciones específicas.

En este orden de ideas, llegamos a la conclusión de que el costo de acceso en un ambiente móvil no es simétrico. El costo para una unidad fija es no solo el costo de comunicación, sino que se suma el costo de la búsqueda de ese dato. Para una unidad móvil, se reduce al costo de comunicación.

Si una unidad móvil es utilizada como servidor, alguno de sus datos no estarán disponibles mientras esté desconectada. Estas pérdidas tienen que ser manejadas con cuidado, dado que no se preve un tiempo de reconexión, tal como se hace normalmente en los sistemas distribuídos tradicionales, donde se presume un timeout de reconexión. Para limitar el número de pérdidas, es importante seleccionar cuáles serán los datos almacenados localmente y explotar al máximo el tiempo de conexión para recuperar datos.

2 Un algoritmo de replicación y movilidad

Presentaremos a continuación las consideraciones generales para establecer un algoritmo de replicación en un ambiente móvil.

En primer lugar debemos tener en cuenta que todo mecanismo de replicación contempla dos subalgoritmos: uno de control de concurrencia y otro de consistencia. El primero asegurará que las operaciones conflictivas no se realicen simultáneamente y que el efecto final sea

una planificación serializable, [2]. El segundo que en todo momento todas las copias tengan el mismo valor.

A continuación se ofrece una revisión de estos algoritmos que provienen del area de sistemas distribuídos fijos, a los fines de adaptarlos a sistemas distribuídos móviles.

2.1 Concurrencia

El algoritmo de replicación tiene que pertenecer a la categoría de aquellos con quorum dinámico o variable. Si algún nodo móvil pertenece al quorum, es posible que éste esté largo tiempo desconectado, por lo tanto no puede ser el único en el que resida un dato. Por lo menos, debemos encontrar dos copias del dato, una de ellas en un nodo fijo. El esquema debe ser *adaptivo al conjunto de aplicaciones vigente*. Un sistema es adaptivo sensible a la aplicación[1] si provee un balance entre la transparencia de aplicación, donde el sistema esconde la movilidad a las aplicaciones y otro que simplemente deja hacer sin soporte del sistema, [1]. En este sentido las aplicaciones advierten la disponibilidad de recursos en un cierto instante. Esta aparente dicotomía entre transparencia y adaptabilidad es un concepto nuevo aplicable a sistema con movilidad. Si usáramos simplemente transparencia, una aplicación podría ser abortada por no disponibilidad de datos, posiblemente por falla de una estación móvil, con lo cual, el sistema no progresaría y la transparencia desaparecería, ya que los usuarios reconocerían esta situación. En el otro extremo, no hacer nada también puede dejar al sistema sin progreso, ante la carencia de algún recurso. Por ello, solo si *todos* los recursos no están disponibles entonces el sistema podrá fallar, lo cual parece razonable.

El punto que se propone aqui es que la aplicación reconozca la situación y actúe consecuentemente. Por ejemplo, recuperando alguna versión de un dato, tal vez mas vieja o con cierta degradación que igual le permita progresar, aunque no en las condiciones ideales. Pensemos en una aplicación que necesita una foto para trabajar; ésta tiene una gran variedad de presentaciones, (blanco y negro vs color, alta vs baja resolución, reducida vs tamaño real). Es posible que una aplicación requiera esta foto, pero solo esté disponible una copia de la misma en alguna calidad menor a la optima, ¿puede la aplicación conformarse con este dato? En otras circunstancias, este análisis no es posible. Por ejemplo, si un agente móvil desea hacer un retiro o transferencia de su cuenta bancaria no puede "conformarse" con algún saldo, necesita tener el valor preciso.

Para proveer esta interacción, el sistema debe proveer una cooperación con la aplicación. Se reconocen tres formas:

1. Uso de algoritmos dinámicamente instalables, denominados genéricamente *estrategias*.
2. Monitoreo del ambiente, registro de incorporaciones y bajas.
3. Pistas del usuario o la aplicación.

Describiremos brevemente estos puntos, aunque el último queda fuera del alcance de este artículo.

[1] application awareness

2.1.1 Estrategias

Las estrategias de nuestro algoritmo de replicación se refieren al manejo de las versiones, asegurar la propagación de actualizaciones, detectar los conflictos y resolución automática de las disponibilidades. Estos son mecanismos de bajo nivel independientes de la aplicación y es parte de un *middleware* provisto por el sistema, en la forma de librerías. También podemos encontrar estrategias para el control de acceso, consistencia, filtrado, compresión de la información, etc.

Las aplicaciones en sí pueden hacer uso de estas estrategias o bien proveer las propias, más adaptadas a la semántica de la aplicación. Los cambios del ambiente pueden afectar la performance de una estrategia, por ello puede ser dinámica y proveer alternativas, (por ejemplo, consistencia fuerte para ambientes conectados o consistencia débil para un ámbito móvil).

Las estrategias pueden ser definidas como clases, con sus objetos, funciones y métodos. Por ejemplo, en un ámbito de replicación, el objetivo es determinar si un objeto puede ser copiado o no. El objetivo no es copiar el objeto; esto será determinado por otra estrategia que descubrirá la copia de acuerdo al monitoreo, (ver la seccion siguiente).

El control de replicación puede decidir copiar siempre un objeto, no copiarlo nunca o permitirlo en caso de que haya menos de n copias. Esta función de control está adosada a la clase a la que pertenece el objeto.

Una alternativa a la replicación que hemos analizado en [12] es la de incorporar dos tipos de copias de datos: copias de lectura y de escritura. Las primeras son de acceso rapido y estan liberadas del control de la concurrencia, las segundas sirven obviamente para modificar un dato y quedan bajo el control del protocolo de replicación. Para el manejo de la consistencia, debemos actualizar las copias de lectura, pero esta operacion puede hacerse en forma asincrónica o en una política perezosa.

Esta distinción puede ser particularmente útil para las estaciones móviles que pueden conformarse con una copia para trabajar, que tal vez no sea aquella últimamente actualizada.

2.1.2 Monitoreo

El monitoreo del ambiente va a estar abocado principalmente a las estaciones móviles que presentarán el mayor dinamismo dentro del sistema. Cada objeto tendrá asociado un conjunto de referencias, similar a aquel propuesto por [3].

Cada objeto está asociado a un cluster de grupo. Cada cluster de grupo es un conjunto de estaciones (fijas y móviles) que replican al objeto. El cluster provee un esquema de replicación adaptable, es decir, permite que el dato sea modificado mientras existan suficientes estaciones en él. Cada estación móvil que se retira del sistema (ya sea en forma "gentil" o abrupta) es eliminada del cluster y no será considerada para futuros accesos del objeto. Cada estación fija que deja de servir (típicamente en forma abrupta), será eliminada tambien del cluster. Cuando un objeto queda en un cluster con una única estación entra en estado de alerta, pues el mecanismo de replicación tiene tolerancia cero a las fallas.

Cuando ocurre un estado como el anterior, el sistema buscará otra estación para replicar los objetos del cluster y asi incorporar mayor seguridad al sistema. Los sistemas deben ser cooperativos y se pueden establecer prioridades para encontrar otro servidor apropiado.

Cuando una estación se reincorpora al sistema, averigua el estado de sus clusters, -típicamente una estación móvil tendrá objetos en un único cluster para acelerar este proceso-,

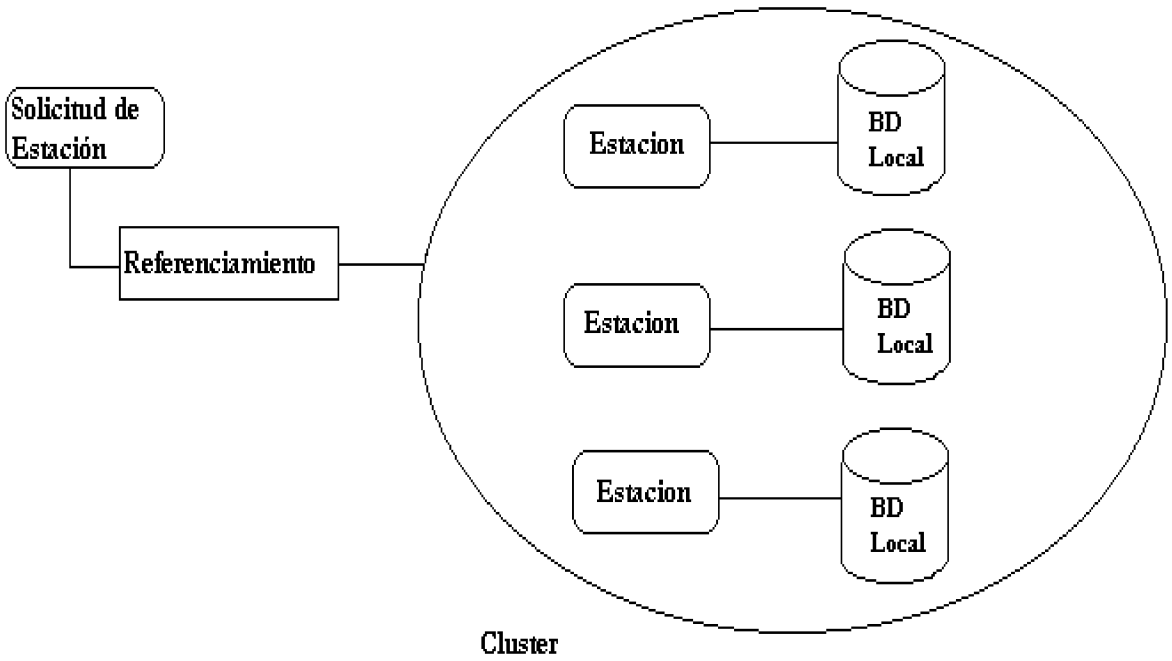

Figura 1: Arquitectura General de Clusters

informando de su estado de conectividad, copiando los objetos actualizados de su cluster y nuevamente sirviendo al sistema.

Observemos como la localidad de un objeto es ahora *migratoria* pues puede ocurrir que la localización efectiva haya cambiado en el tiempo en que una estación estuvo desconectada. Sin embargo, ello no impide que el sistema continúe disponible y brindando servicio, pues la *referencia de cluster no ha cambiado.* Este mecanismo dinámico, adaptable a la performance del sistema es nuevo y distinto a aquellos mecanismos usados en sistemas distribuídos físicos, donde la localidad de un objeto era fija. El cluster constituye entonces un guardián de los datos.

Naturalmente, esto tiene un costo de comunicación y transferencia de información, pero el beneficio es evidente: todo sistema distribuído debe continuar brindando servicio a pesar de la caída de alguna de sus componentes. Es decir, el sistema distribuído es de naturaleza gestháltica, el todo es mucho mayor que la suma de la partes. En la figura, 1 se muestra la arquitectura de este sistema.

Otro mecanismo que se puede utilizar para el manejo de réplicas con seguridad en un ámbito distribuído es aquel denominado de *ligadura dinámica*, [1]. Aqui el objeto es una referencia móvil, con un protocolo de direccionamiento. Por ejemplo, una referencia a un archivo puede estar ligada a un servidor dedicado cuando la estación está fuertemente conectada, a una copia cache, mientras la estación está en movimiento o no conectada y a una copia secundaria cuando la estación está débilmente conectada.

El criterio para utilizar la ligadura dinámica puede ser dependiente del sistema, por ejemplo, puede contemplar la copia más cercana, el servidor menos cargado, el estado de la conexion, la red disponible, la copia disponible, alguna estrategia propuesta por la aplicación o por la calidad del servicio.

Otros parámetros que también deben ser monitoreados, y que eventualmente afectan al mecanismo de replicación, son el ancho de banda de la red, la latencia, el estado de conexion, el costo de la comunicación, el espacio en disco, el tiempo disponible de batería, la posicion geográfica. Estos parámetros son tambien independientes de la aplicación y pueden ser provistos por el middleware de estrategias.

2.2 El Servicio de Copias

Cada cliente fijo tendrá asociado una serie de clusters donde se administrarán los datos compartidos. Como ya dijimos, un cliente móvil no es recomendable que posea más de un cluster, debido al servicio restringido que puede ofrecer. Mientras un cliente móvil se halle fuertemente conectado su percepción del sistema será idéntica a aquella de un cliente fijo. Si el sistema monitorea que la performance de comunicación se degrada, entonces el protocolo eligirá trabajar en modo débilmente conectado y para ello creará localmente un caché de las datos que requiere y modifica. Posteriormente, cuando se reestablezca la comunicación reintegrará sus modificaciones al sistema.

La replicación, como ya dijimos, será administrada por clusters, lo que nos lleva a un control único de referenciamiento de datos. Obviamente, cuando se necesita un dato, se lo solicitamos al cluster correpondiente. Este administrador, como ya dijimos está distribuído entre varias localidades, preferentemente fijas. Si existe acuerdo, entonces podemos acceder a la copia.

Veamos formas de implementar este servicio.

2.2.1 Esquema con Copia Primaria

Una alternativa, mencionada en [9], se denomina *server-based write* y es una extensión del mecanismo de copia primaria y copias secundarias de réplicas en sistemas estáticos.

Cuando un cliente C pretende propagar las modificaciones de los datos, es decir, realizar el proceso de reintegración correspondiente, puede forzar la escritura. Esto es muy útil para vaciar la cache de C que puede estar llena debido a un período de alta actividad de actualizaciones en modo desconectado.

El servidor P, una vez que recepta esta escritura forzada, graba las actualizaciones en memoria estable y luego propaga las actualizaciones a los n servidores secundarios S, a través de un multicast. Cada servidor S envía un reconocimiento y salva las actualizaciones en su propia memoria estable. En este punto es interesante aclarar que para cada dato habrá un conjunto de servidores secundarios distintos, no todos los datos estarán almacenados en todos los servidores. Cada servidor primario P tendrá un conjunto de servidores secundarios S asociados con un dato particular, es decir suponemos una replicación parcial de la base de datos.

Cuando P recibe los n reconocimientos, informa a C que las actualizaciones pueden ser descartadas de su cache. Hasta ese momento, C mantenía las copias caché. Esta es la única obligación de C para el servicio, inclusive C puede hasta "olvidar" el nombre de P, siendo que entonces debe aguardar hasta la próxima reintegración para saber cual servidor leer.

Además, se debe notar que P debe esperar los n reconocimientos para poder enviar a C el mensaje de que puede liberar su cache, es decir el servicio es tolerante a n fallas. El esquema se muestra en la figura 2.

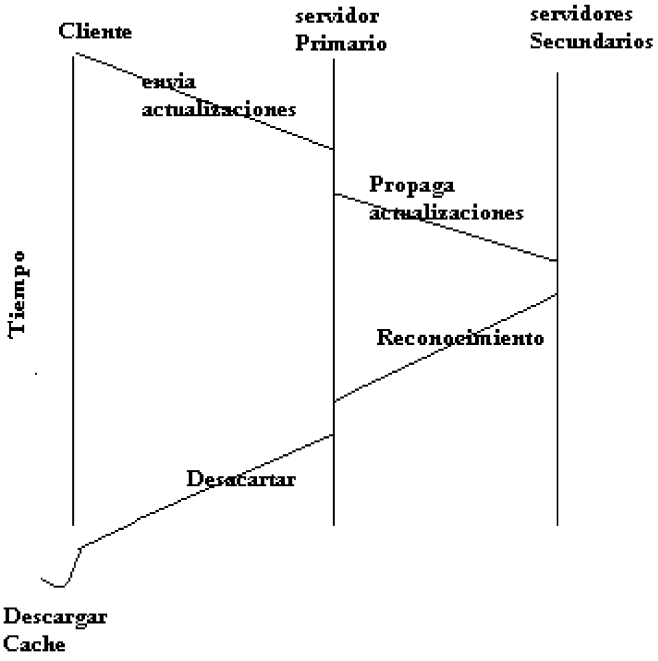

Figura 2: Esquema Copia Primaria

2.2.2 Manejo de Token

El problema del cluster y la consistencia puede realizarse mediante el uso de un token que expresa el permiso. Este esquema es importante porque introduce consistencia en n réplicas sin requerir n permisos. Como ya dijimos cada dato pertenece a un cluster. Para un cliente móvil, cuyos datos están agrupados en un único cluster, podemos introducir la idea de token, resignando una granularidad más fina, pero introduciendo la idea de *localidad de acceso*, propia de los sistemas operativos cuando hacen caché de datos.

Si un cliente tiene un token t para un dato va a hallar la versión actual con el mismo costo que una lectura estricta. Veamos los pasos:

1. Chequear la cache del cliente: si un cliente tiene un token t, esto significa que mostró el deseo de realizar una lectura estricta por lo que en caso de tener una copia en su cache, con seguridad es la más actual.

2. Si no se halla la copia, chequear el servidor de cluster, P. En caso de ser hallada se la transfiere a la cache del cliente, junto con la autorización de retener t sobre ese dato (o cluster, depende de la granularidad elegida).

Como se ha visto, el servidor de cluster P es simplemente aquella referencia de localidad del cluster. El sistema puede haber cambiado la localidad física del cluster. El fallo en el cliente móvil que tenía información vieja es absorbido por el sistema quien, junto con el nuevo dato, enviará la nueva localidad física de P, actualizando de este modo su DNS.

Cualquier copia de algún miembro del cluster P está actualizada. El servidor de cluster se encargará de actualizar todas las copias en algún momento. Para una lectura estricta, t no necesita ser transferido a P o a C, sin embargo, si el cliente solicita una operación de este tipo, es muy probable que luego requiera una escritura y por lo tanto, si el servidor S no requiere a t puede transferirlo a P, de modo que esta potencial operación sea atendida brevemente en el futuro. Esta transferencia de permiso de escritura puede ser inútil, y por ello P puede entregar el token t a otro servidor S que asi lo requiera antes que C.

En este caso, corremos el riesgo de que la transacción en C sea abortada en el momento de la reintegración por problemas de serializabilidad. Sin embargo, si retenemos el token t en forma inútil, degradaremos la accesibilidad a ese dato. Finalmente, si imponemos un acceso estricto de lectura-escritura, caemos en un mecanismo tipo lock que tiene muy baja disponibilidad de datos, aunque garantizamos trivialmente la serializabilidad.

Un cliente recibe un token con el dato requerido en respuesta a una lectura-estricta solo si es el único "escritor consistente potencial" leyendo el dato. En contraste, cuando una lectura estricta la realiza un cliente que no tiene token se contacta cada servidor del dato y éstos deben retornar la copia del dato.

La comparación entre las copias puede manipularse a través de un "vector de versión´´, [5], el servidor primario compara el vector de versión y da al cliente la copia más reciente y le informa si dicha copia tiene una indicación de token. Este vector de versiones puede manejarse más simplemente que en la literatura propuesta, pues esperamos que los servidores fijos tengan los mismos valores para un dato, mientras que para los servidores móviles el problema puede haber sido un largo tiempo de desconexion y por lo tanto una desactualización de copias. Si permitimos que los móviles sean servidores de datos, el vector puede ser una estrategia para homogeneizar los valores, mientras que para los servidores fijos podemos adoptar una estrategia tipo espejo, (*mirror*).

Los token son liberados cuando ocurre al menos una de las siguientes situaciones:

- Uno o más clientes tienen estricta lectura de datos pero no tienen permiso de escritura.
- Un cliente que tiene permiso de escritura y lectura-estricta tiene un dato y un segundo cliente en las mismas condiciones pretende leer un dato. Si el cliente no lo requiere, entonces lo entrega a este último.
- El servidor primario P es incapaz de contactar el cliente por un largo periodo de tiempo, en este caso el servidor descarta su registro del token del cliente, evitando así el bloqueo de larga duración (o eventualmente infinito si el cliente móvil no logra recuperarse).

Se debe tener en cuenta que es absoluta responsabilidad del cliente el solicitar lecturas estrictas cuando se sabe que el dato va a ser compartido. Por otro lado el servidor puede opcionalmente retener en memoria no solo a los clientes que hacen lecturas estrictas sino también aquellos que realizan lecturas relajadas. Así, el servidor podría detectar potenciales interferencias entre las lecturas estrictas y relajadas.

2.2.3 Consistencia Débil

En algunos contextos, sobre todo en aquellos de aplicaciones web, puede pensarse que un recurso, si bien es compartido, tiene un *servidor de origen*, [4], al que se accede mediante una dirección única, tipo URL. Como el documento es compartido, cuando un cliente solicita el

documento, el servidor se lo envía junto con una estampilla de tiempo. Existen proveedores intermedios, tales como proxies, espejos, sistemas de archivos, etc, que se denominan nodos de replicación y se crean asi diferentes puntos de acceso. A su vez, cuando un cliente recupera una versión del dato de un servidor, se transforma en un nodo primario réplica de esa versión.

En esta arquitectura se proveen agentes que permiten registrar en el sistema distribuído los diferentes datos que cada nodo posee o los que deja de poseer, en caso de remover un dato, por carencia de espacio, por ejemplo.

3 Aspectos semánticos

Hasta este punto se han analizado, adaptado y extendido mecanismos clásicos de control de concurrencia, basados principalmente en mecanismos sintácticos, de acuerdo a las operaciones de escritura y lectura. Este enfoque resulta tradicional y naturalmente respeta la serializabilidad.

Sin embargo, son más restrictivos que algunas planificaciones serializables. Con esto queremos decir que toda planificación que respete alguno de los protocolos anteriores es serializable, pero existen planificaciones serializables que no necesariamente respetan estos protocolos.

Este hecho nos lleva a pensar en la posibilidad de analizar otros protocolos que tal vez sean más adaptables al ambiente distribuído y sobre todo en un contexto móvil.

El primer hecho importante es el concepto de transacción. Una transacción es, desde el punto de vista de la aplicación, una operación atómica. La primera pregunta que nos hacemos es ¿podemos resignar esta propiedad? En algunos contextos es posible. Supongamos una consulta de una unidad móvil a internet solicitando información sobre bibliografía. Normalmente de todos los nodos disponibles es posible que alguno no sea alcanzable (por razones de comunicación, o porque este nodo no está activo o porque el ancho de banda no es suficiente para transmitir la información en un tiempo adecuado). Si esta consulta se toma como una transacción, entonces la misma deberá ser abortada, sin embargo, podríamos conformarnos con una visión más pequeña que el todo e igualmente poder proseguir con la operación.

Otro punto importante con respecto a la definición de una transacción es la granularidad de la misma. Cuanto más pequeña sea una transacción tenemos mayores posibilidades de éxito pues será más rapidamente ejecutada, requerirá menor tiempo de comunicación e interface y accederá a menor cantidad de datos. Además si usamos un algoritmo de token, por ejemplo, el tiempo de retención de un recurso será mucho menor. En este sentido, las transacciones de larga duración no son aptas para sistemas móviles.

El concepto de serializabilidad semántica nos lleva a pensar en un criterio de serializabilidad que dependa de la aplicación. Es posible que se defina para un sistema una especificación particular de correctitud. Por ejemplo, en una aplicación estadística o en una auditoría de sistemas, puede no tenerse en cuenta el orden serial ya que el resultado es simplmente una foto del estado del sistema en un determinado momento. Está claro que el sistema es dinámico y cambiará constantemente por lo cual no se puede impedir proseguir a las otras transacciones. Para este tipo de aplicaciones, el criterio serial puede ser muy restrictivo. Igual criterio se puede seguir a una aplicación sobre la Web.

4 Conclusiones

En este trabajo se han presentado diversas alternativas al problema de los datos distribuídos en un ambiente móvil. La movilidad requiere técnicas específicas que permitan aprovechar al máximo el tiempo de conexión y que soporte las desventajas de un modo desconectado.

En un sistema tradicional, cuando un nodo se halla incomunicado, estamos en presencia de una falla. El sistema se protege contra esta falla, continúa operando y cuando el nodo se reestablece, se reincorpora al sistema. En un ambiente distribuído móvil, no existe una falla al haber incomunicación pues éste es uno de los estados posibles del sistema. Al igual que en un ambiente fijo, el nodo movil, cuando reestablezca la comunicación deberá actualizar sus datos.

Aparentemente, ambos sistemas son similares, pero existe una diferencia muy fuerte, la incomunicación de un nodo móvil no puede afectar la performance del sistema. Por esta razón, los sistemas se preparan para contingencias de este tipo, básicamente replicando los datos en diferentes estaciones e incorporando mecanismos de control de concurrencia dinámicos o adaptivos.

En este sentido, se han presentado varios algoritmos, algunos de ellos variantes de algoritmos encontrados en la literatura de sistemas distribuídos fijos y también algoritmos originales propios de este entorno.

La idea básica es agrupar los datos en clusters de datos. El cluster es un tipo de dato abstracto que agrupa las localidades en las que reside un dato. El cluster es de administración distribuída y se halla identificado por un nombre (como si fuera un URL), pero no tiene una localidad fija. Esto permite que el cluster migre en el sistema, o lo que es lo mismo, que tenga una existencia virtual. El cluster se autoadminstra como si fuera un agente, creando nuevas copias del dato, si observa una potencial situación de riesgo.

En este orden de ideas, se presentó una implementación básica de cluster, apoyado en las ideas de copia primaria. Aqui, el sistema no tolera demasiadas fallas y se dividen las copias en primaria y en espejadas. Aquellas espejadas serán actualizadas oportunamente por el adminstrador primario. Éste tiene un rol distinto al de los nodos secundarios.

Se presentó una implementación de cluster basada en la idea de token, con un manejo aumentado por una cache local y un proceso de reintegración a las otras copias. Aqui hay una horizontalidad de las estaciones, cada una de ellas actuando en diferentes roles según las necesidades del sistema.

Finalmente, se incorporó la idea de consistencia débil, es decir, se relajó el criterio de serializabilidad, principio básico de todos nuestros algoritmos. Este area debe seguir una exploración más profunda. Existen aplicaciones de consistencia débil, para ciertas estructuras de datos o para sistemas específicos, pero aún no hay patrones genéricos.

Referencias

[1] A. Baggio. *Design and Early Implementation of the Cadmium Mobile and Disconnectable Middleware Support.* Projet SOR. Rapport de recherche N. 3515. Octubre 1998. INRIA.

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

[4] Mesaac Makpangou, Guillaume Pierre, Christian Khoury, Neilze Dorta. *Replicated Directory Service for Weakly Consistent Distribuited Caches.* Rapport de Recherche, INRIA, N. 3514. Octubre 1998.

[5] K. Salem, H. García Molina, R. Alonso. *Altruistic Locking: A strategy for coping with long-lived transactions.* Proceedings of the 2nd International Workshop on High Performance Transaction Systems. pp 19.1-19.24, Septiembre 1987.

[6] L.B. Mummert. *Exploiting Weak Connectivity in a Distributed File System.* Carnegie Mellon University. Dic. 1996. http://www.cs.cmu.edu/afs/cs/project/coda/web/docs-doca.html, CMU-CS-96-195.

[7] Sampath Rangarajan, Sanjeev Setia, Satish K. Tripathi. *A Fault-Tolerant Algorithm for Replicated Data Management.* IEEE Transaction on Parallel and Distributed Systems, vol6, No. 12, Diciembre 1995.

[8] J. Kistler, M. Stayanarayanan. *Disconnected Operation in the Coda File System.* ACM Transaction on Computer Systems, N. 10, Vol. 1. Feb 1992.

[9] . C.D. Tait y D. Duchamp. *Service Interface and Replica Management for Mobile File System Clients.* Columbia University. Proceedings of the Eleventh International Conference On distributed Computing System, pages 2-9. IEEE. Mayo 1991

[10] M. Zanconi, J. Ardenghi. *Sobre el Manejo de Réplicas.* III Congreso Argentino de Ciencias de la Computación. Universidad Nacional de La Plata. Facultad de Ciencias Exactas. Departamento de Informática. La Plata. Octubre 1997.

[11] M. Zanconi, J. Ardenghi. *Un protocolo para réplicación dinámica de Datos.* IV Congreso Argentino de Ciencias de la Computación. Universidad Nacional del Comahue. Facultad de Economía y Administración. Departamento de Informática y Estadística. Neuquén. Octubre 1998.

[12] M. Zanconi, J. Ardenghi. *Replicacion de Datos en Ambientes Móviles.* V Congreso Argentino de Ciencias de la Computación. Universidad Nacional del Centro. Octubre 1999.