TRANSITANDO HACIA LAS BASES DE DATOS DE TIEMPO REAL

Carlos E. Buckle, José M. Urriza, Francisco Paez

Facultad de Ingeniería, Departamento de Informática
Universidad Nacional de La Patagonia San Juan Bosco - Puerto Madryn, Argentina
cbuckle@unpata.edu.ar, josemurriza@unp.edu.ar, franpaez@gmail.com

Resumen: Los Sistemas de Tiempo Real enfrentan la necesidad de trabajar con Bases de Datos de forma estructurada y organizada. Los Sistemas de Bases de Datos empresariales comienzan a requerir un rendimiento predecible con transacciones sobre grandes volúmenes de datos y tiempos de respuesta certeros. En los sistemas de automatización industrial se combinan necesidades de Tiempo Real y alta tasa de transacciones con datos persistentes. Atendiendo a estas demandas surgen los Sistemas de Bases de Datos de Tiempo Real, los cuales en la actualidad son más que una definición teórica. Se han desarrollado desde hace tiempo numerosas investigaciones al respecto y un conjunto de experiencias que intentan dar solución a problemas cada vez más frecuentes. Este trabajo está dirigido a investigadores y profesionales del área de Tiempo Real interesados en problemáticas de Bases de Datos. Tiene como principal objetivo introducir a la temática, exponer conceptos fundamentales, identificar problemas específicos y presentar un relevamiento de líneas de investigación e implementaciones concretas, tanto experimentales como comerciales.

Palabras Claves: Bases de Datos, Sistemas de Tiempo Real, Sistemas de Bases de Datos de Tiempo Real. *RTDBS*, *SGBD*, Consistencia Temporal, Transacciones de Tiempo Real, Transacciones con vencimiento.

1 Introducción

Los Sistemas de Base de Datos (*SBD*) se han perfeccionado apuntando a determinados aspectos como: performance, capacidad de procesamiento de transacciones, propiedades *ACID* (*Atomicidad, Consistencia, Aislamiento y Durabilidad*) de las transacciones y disponibilidad de los datos. Este desarrollo ha tomado forma a medida que las empresas fueron aumentando las exigencias hacia los sistemas que manejan sus datos, debido a que en el presente, gestionar datos es gestionar uno de los principales activos de las compañías. A estos sistemas se les deriva cada vez más responsabilidades de negocio y son un aspecto clave en la calidad de servicio ofrecida a clientes como: datos seguros, correctos y disponibles siempre.

Sin embargo, en algunos sistemas hay ciertos aspectos que siguen presentando incógnitas. Por ejemplo: *¿Es posible identificar objetos sobre la cinta transportadora de una fábrica comparando patrones guardados en un base de datos y reaccionar puntualmente para guiarlos a diferentes destinos?, ¿Es posible establecer en un contrato de servicios el compromiso de que las transacciones electrónicas serán resueltas en un determinado tiempo máximo?, ¿Se puede garantizar que el sistema de monitoreo obtiene información instantánea, deriva información instantánea y muestra comportamientos de datos confiables?*

Estas incógnitas están vinculadas entre sí por problemas comunes, que son los estrictos requerimientos en el tiempo de respuesta para el procesamiento de las transacciones y las consultas sobre las bases de datos. Estos requerimientos son constricciones de tiempo, las cuales debe ser consciente el planificador de transacciones. Consecuentemente, las disciplinas de Bases de Datos (*BD*) y Tiempo Real (*TR*) trabajan en conjunto para abarcar este tipo de situaciones.

A continuación se determinaran diversos tipos de requerimientos. En la sección 2 se presenta los sistemas de bases de datos de tiempo real. En la sección 3 características y problemas de un *SBDTR.* En la sección 4 se presenta el avance de las soluciones en *SBDTR*. En la sección 5 se exhiben las conclusiones. Finalmente en la sección 6 se presentan los trabajos futuros.

1.1 Sistemas de Bases de Datos con Requerimientos de Tiempo Real

Un *SBD* maneja datos *persistentes* garantizando durabilidad y consistencia a través de gestión de transacciones. Una transacción es un conjunto de operaciones que implementan una función lógica indivisible dentro de un *SBD.* En [1] se define a una transacción como: *"Una parte de un programa que accede y posiblemente actualiza varios datos con un fin específico."*. Todo conjunto de acciones ordenadas que debe llevarse a cabo para lograr un resultado en una *BD* está comprendido dentro de una transacción. Las transacciones se definen con cuatro propiedades fundamentales, conocidas como propiedades *ACID*:

- Atomicidad: Una transacción se ejecuta por completo o no se ejecuta. Confirma todas sus operaciones satisfactoriamente (*commit*) o deshace sus operaciones si no pudo completarse (*rollback*).
- Consistencia: Una transacción que altera datos garantiza que la *BD* pasa de un estado consistente a otro, cumpliendo las reglas de integridad.
- Aislamiento: Las actualizaciones parciales de los datos no son visibles por otras transacciones hasta que la transacción se confirme.
- Durabilidad: Una vez que una transacción se ha confirmado el resultado debe persistir en la *BD* aunque se produzcan fallas posteriores.

En un ambiente de multiprogramación se garantiza además que la ejecución concurrente de transacciones no ponga en riesgo la consistencia. Permite controlar las secuencias de operaciones que acceden a datos compartidos para que sean ejecutadas en serie y no concurrentes en caso de conflictos. A esto se llama *serialización*.

Además de la calidad de los datos, un *SBD* tiene objetivos de rendimiento que apuntan a reducir el *tiempo de respuesta promedio* de las transacciones del sistema.

Pero las características comentadas no son suficientes para dar solución a los problemas planteados en el punto anterior, en los cuales el *SBD* debe garantizar tiempos de ejecución y de respuesta estrictos. Por ejemplo para el ámbito de las aplicaciones web, un sitio de subasta electrónica, donde varios, tal vez miles de participantes subastan simultáneamente un producto, la actualización de la mejor oferta debe ser publicada a todos los interesados al mismo tiempo para no dar ventajas. A su vez, las ofertas tienen un tiempo límite o vencimiento. El ranking de ofertas se debe realizar en tiempo real, entre todas las ofertas, antes de que éstas envejezcan. Además, se debe considerar que el sistema debe atender concurrentemente múltiples subastas electrónicas con las mismas condiciones de exigencia.

En estos casos el *SBD* debe tener en cuenta el tiempo límite en el cual cada transacción debe responder y en función de ello implementar una política de planificación de transacciones que permita garantizarlos. Estos requerimientos de *TR* se transforman paulatinamente en una exigencia para los *SBD*.

1.2 Sistemas de Tiempo Real con Requerimientos de Bases de Datos

Algunos *Sistemas de Tiempo Real* (*STR*) tradicionales manejan sus datos en estructuras dependientes de la aplicación, en ocasiones porque no manejan gran cantidad de datos o bien porque no tienen importantes requerimientos de persistencia. Pero también hay *STR* con una fuerte orientación a los datos, que trabajan con modelos complejos y de importante volumen.

Supongamos en el ámbito industrial, un ejemplo de un sistema que debe identificar y guiar un objeto sobre una cinta transportadora de una fábrica. Debe capturar la posición del objeto e imágenes periódicas del mismo, compararlas con patrones almacenados en una *BD*, identificarlo y, de acuerdo al tipo de objeto, tomar la acción que corresponda registrando en la *BD* la decisión tomada y el tipo de objeto identificado de manera que el sistema de monitorización muestre el estado de la cinta y la cantidad de objetos guiados. Esto se muestra en la Figura 1.

En estos casos el *STR* necesita contar con una herramienta que gestione la definición, almacenamiento, el procesamiento y la consistencia de los datos. La herramienta recomendada es un *Sistema Gestor de Bases de Datos* (*SGBD*).

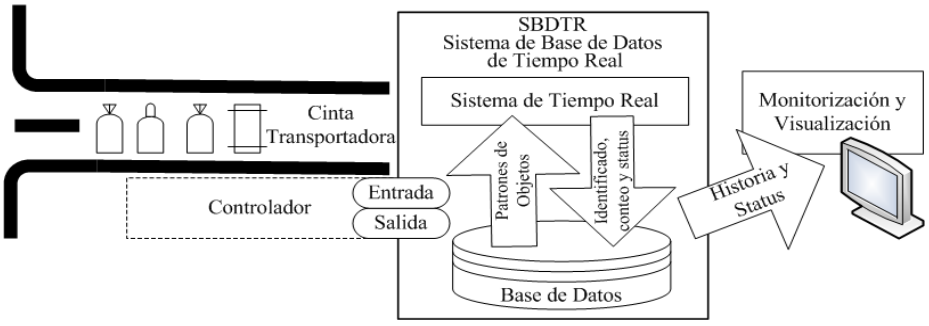

Figura 1. Ejemplo de *STR* que guía objetos sobre cinta transportadora de una fábrica

Los *SGBD* convencionales tienen diferentes puntos de vista respecto de lo que significa performance y correctitud. Que un *SGBD* convencional sea eficiente y rápido no significa que pueda cumplir siempre con las constricciones de tiempo requeridas en un *STR*. Los *SGBD* convencionales difieren de los *STR* pues planifican de manera diferente sus actividades.

Implementar un *STR* haciendo uso de un *SGBD* convencional pone seriamente en riesgo la *puntualidad* y la *predictibilidad* de este, y esto generalmente lo hace inaplicable. Se entiende como *puntualidad* a la garantía de cumplir con los tiempos de vencimiento de las transacciones y como *predictibilidad* a la capacidad de poder asegurar de antemano que todas las transacciones con vencimiento estén garantizadas.

2 Sistemas de Bases de Datos de Tiempo Real

En las situaciones planteadas se puede ver que hay *STR* con requerimientos de *BD* y *SBD* con requerimientos de *TR*. Ambos escenarios han sido estudiados por investigadores de *TR* y de *BD* dando forma a la sub-disciplina denominada *Sistemas de Bases de Datos de Tiempo Real* (*SBDTR*), en la cual, se conjugan las ventajas que aportan los *SGBD* con la predictibilidad y puntualidad de los *STR*.

Existes numerosas ventajas que aportan los *SGBD*, principalmente: colaboran en eliminar redundancia de datos, garantizan integridad y durabilidad mediante transacciones *ACID*, aseguran la ejecución consistente en ambientes de concurrencia, logran buena eficiencia en el acceso a datos, soportan tolerancia a los fallos, aportan independencia de datos mediante capas de abstracción que separan la estructura lógica conceptual de la implementación física de la *BD*, los datos están auto-descriptos en un diccionario, implementan seguridad y protección, etc.

El procesamiento de transacciones en un *SGBD* convencional garantiza resultados lógica y aritméticamente correctos, pero en *TR* este resultado solo es correcto si además se produce antes del vencimiento de la transacción [2]. Esto ha llevado a estudiar puntos de vista alternativos para determinados problemas, por ejemplo: en la planificación se deben considerar alternativas para la serialización de transacciones en busca de cumplir con los vencimientos, el aislamiento de transacciones no siempre es crítico, la puntualidad es más importante que la correctitud en muchos casos, algunas transacciones son periódicas análogas a tareas de *TR*, etc.

En base a estos replanteos se han ido caracterizando los *SBDTR*. Existen confusiones comunes con respecto a su significado, tales como: el *SBDTR* debe ser pequeño, debe caber en memoria, debe ser rápido, trabajar con poca cantidad de datos, etc. Las personas que desconocen la disciplina *TR*, son propensas a pensar que se trata de usar un *SGBD* comercial muy rápido.

En [3], *Stankovic* identifica varios conceptos erróneos acerca de los *SBDTR* y que es conveniente aclarar, para guiar las ideas y definiciones. Numerosas confusiones provienen de considerar que una mayor capacidad de cómputo soluciona los problemas de *TR*, ya sea por avances en el *hardware* o en los *SGBD*, y en [2] explica que un alto desempeño no es suficiente para garantizar tiempos estrictos. También existe la idea errónea de que la *BD* debe estar siempre en memoria en un *STR*. Si bien es una mejora importante, la *BD* en memoria por sí sola no garantiza planificación de

transacciones y además sobre grandes volúmenes de datos puede ser inviable. Sin embargo, se debe determinar si es posible administrar accesos a disco con requerimientos de *TR*, como se presenta en [4].

Un *SBDTR* y un *SGBD* convencional tienen concepciones diferentes principalmente porque, en su diseño interno los *SBDTR* consideran el vencimiento (*deadline*). Consecuentemente, las transacciones con vencimiento, al igual que las tareas en un *STR*, pueden tener distintos comportamientos respecto del cumplimiento o no de dicho tiempo límite.

En los sistemas críticos, con *vencimientos duros*, las transacciones deben finalizar antes de su vencimiento, y el sistema debe poder garantizarlo ni bien se conocen los detalles de las transacciones duras a ejecutar, caso contrario se considera *no planificable*.

En sistemas con *vencimientos blandos*, se intenta cumplir con los vencimientos, pero si no es posible y la demora es aceptable, el sistema puede continuar con la ejecución de la transacción. Los resultados son óptimos si se cumplen antes del vencimiento, son aceptables en instantes inmediatos al vencimiento y a medida que transcurre el tiempo los resultados van perdiendo valor.

Con *vencimientos firmes* se respeta el vencimiento, pero no es crítico si no se llega a cumplir. A diferencia de los vencimientos blandos, no tiene sentido seguir ejecutando la transacción después del vencimiento. Un protocolo de implementación es planificar y avanzar la transacción, y si no finaliza antes del vencimiento se aborta.

De esta manera, la *puntualidad* de los *SBDTR* está asociada a dos principales características que poseen las transacciones: el vencimiento y su criticidad. La característica de la *predictibilidad* está asociada a las garantías de planificación. Para poder predecir, se debe poder planificar el conjunto completo de todas las transacciones, considerando si tienen o no vencimiento y la criticidad del mismo.

3 Características y Problemas de un *SBDTR*

En esta sección se tratará sobre las principales características de los *SBDTR* y que han determinado las principales líneas de investigación en el área.

3.1 Consistencia Temporal

Los *SBD* utilizados en *TR* procesan en sus transacciones, datos que están relacionados con el entorno de operación del sistema. En el ejemplo presentado en la sección 1.2, para la identificación de un objeto que se viene aproximando, es necesario obtener periódicamente la posición desde el ambiente. El sistema trata con variables que en el exterior se están actualizando permanentemente y el *SBDTR* debe ser consciente de dichos cambios. Para esto es necesario que cada dato externo pueda ser procesado oportunamente, de manera que el valor de la *SBDTR* sea equivalente respecto de su contraparte en el ambiente. A esto se lo conoce como *consistencia temporal absoluta* o *consistencia externa* ([5]).

Otro requerimiento a considerar, es que si hay varios de estos datos en una transacción de *TR,* todos deben pertenecer a la misma imagen instantánea del

ambiente. Un grupo de datos accedidos para un mismo fin debe reflejar el estado de sus correspondientes objetos externos dentro de un intervalo cercano de tiempo. A este requerimiento se lo conoce como *consistencia temporal relativa*.

En los *SBDTR* se necesita poder validar la *consistencia tempora*l (como lo define *Ramamritham* en [6]) tanto *absoluta* como *relativa* y para esto se necesita registrar una marca de tiempo (*timestamp*) de lectura que recuerde la última actualización de cada dato respecto de su contraparte en el exterior.

Para validar la *consistencia temporal absoluta* de un dato *d* es necesario conocer el período de validez durante el cual, el valor interno de dicho dato es aceptable. A ese período se lo llama *intervalo de validez absoluta* (*avi*). De esta manera *d* cumple con las condiciones de consistencia absoluta si su *timestamp* cumple con:

$$(t_{actual} - timestamp) \leq avi$$

Para validar la *Consistencia Temporal Relativa* de un conjunto de datos Ω es necesario conocer cuál es el tiempo máximo de diferencia que puede existir entre los refrescos de cada dato del conjunto. Esto implica establecer el tiempo máximo de diferencia que sea aceptable entre las lecturas de los datos de un mismo conjunto Ω. A este período se lo llama *intervalo de validez relativa* (*rvi*). Luego, Ω_{rvi} es el *rvi* del conjunto de datos Ω. En principio, obtener el *rvi* de un conjunto con diferentes *avi* para cada dato significa que prevalezca el *avi* más pequeño. De esta manera, el *rvi* del conjunto es, a lo sumo, el *avi* más pequeño.

$$\Omega_{rvi} \leq \min_{i=1}^{n} avi_i$$

Para validar la consistencia relativa se debe chequear los *timestamp* de lectura de todos los datos involucrados. Sea el ts_i el *timestamp* del dato d_i de un conjunto Ω con *n* datos, se dice que cumple con las condiciones de consistencia relativa si:

$$\forall d_i \in \Omega \quad \max_{j=1}^{n} |ts_i - ts_j| \leq \Omega_{rvi} \quad \text{con} \quad i \neq j$$

Esto incrementa las exigencias sobre las transacciones, ya que para ser *correctas* no solo tienen que cumplir con la consistencia lógica sino también con la consistencia temporal.

3.2 Periodicidad de Transacciones

Cumplir con la *consistencia temporal* de los datos y con los requerimientos de tiempo de reacción del sistema conlleva contemplar transacciones periódicas, de esta manera, es posible garantizar requisitos funcionales como: cada 1 seg. leer y grabar la posición de un objeto en movimiento, ó cada 1.5 seg. actualizar dichos valores en un monitor.

El *avi* de los datos puede determinar el período de las transacciones. Por ejemplo, en el caso del objeto en movimiento supongamos que el dato *posición* posee un *avi* de 1 seg. y que cada un período *P* se debe ejecutar una instancia de transacción que lo utiliza y cuyo peor caso de tiempo de ejecución (*WCET*) es *e*. Considerando un planificador por períodos monotónicos crecientes, si una instancia de la transacción comienza en tiempo *t*, la próxima instancia debe iniciar, a lo sumo, en (*t + 2*P – e*).

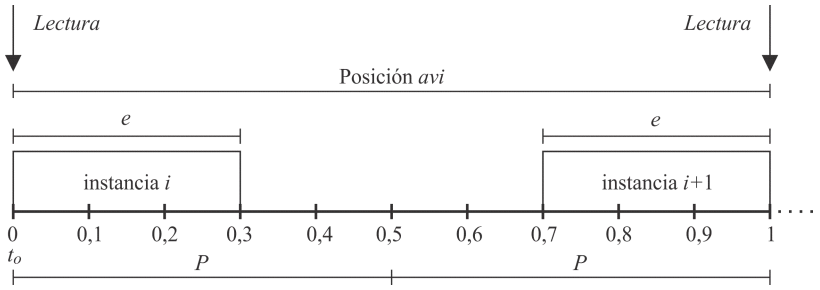

Figura 2. Ejemplo de Consistencia Temporal Absoluta.

De esto se deriva, como se indica en [7], que:

$$P \leq \frac{posición_{avi}}{2}$$

Para garantizar el *avi* de un dato de la transacción, el período *P* no puede ser mayor a la mitad del *avi*. En el ejemplo, *P = 0.5 seg*. como se muestra en Figura 2.

3.3 Criticidad del Vencimiento

Como fue comentado en el punto 2, las transacciones de un *SBDTR* pueden tener diferentes niveles de criticidad en su vencimiento. Estos niveles de criticidad influyen en las decisiones que toma el sistema al momento de planificar transacciones y por ese motivo es fundamental su implementación.

Una implementación frecuente es la basada en el concepto de *Función Valor* introducido por Jensen en [8]. La *función valor* de una transacción permite medir en cada instante de tiempo cuan critico es para el sistema el hecho de finalizar la transacción antes o después del vencimiento. El principio de funcionamiento es que la finalización de una transacción entrega un *valor* al sistema, el cual depende del instante de tiempo en el que se concrete dicha finalización. Los valores positivos son deseables y los negativos indeseables.

Esta *función valor* puede presentar una discontinuidad en el vencimiento de la transacción, dependiendo de la cual se identifican vencimientos duros, blandos o firmes, como se muestra en la Figura 3.

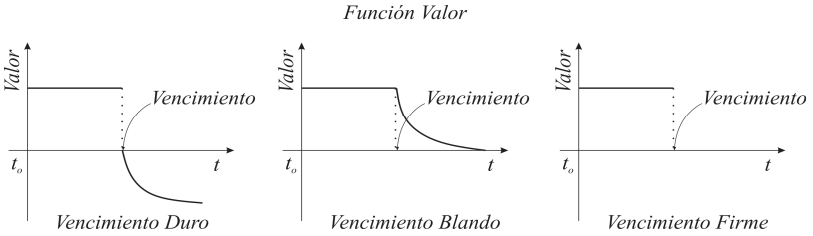

Figura 3. Función Valor.

En los *vencimientos duros*, la *función valor* es negativa inmediatamente después del vencimiento. En los *vencimientos blandos*, la *función valor* es positiva luego del

vencimiento, pero decrece a medida que pasa el tiempo. Finalmente, en los *vencimientos firmes*, la *función valor* es 0 a partir del vencimiento.

3.4 Control de Concurrencia

En un ambiente de multiprogramación, sin una planificación apropiada, la ejecución concurrente de transacciones puede arrojar resultados indeseables. La ejecución de una única transacción *ACID* no presenta inconvenientes. Sin embargo puede suceder que al ser ejecutada concurrentemente con otras transacciones con las que comparte datos, se generen resultados incorrectos. En estos casos, el *SGBD* opta por no intercalar operaciones en zonas de conflicto con recursos compartidos, sino que las mismas son ejecutadas en serie (*serializadas*). Naturalmente esto disminuye el grado de multiprogramación, pero garantiza la consistencia. Para asegurar la serialización las transacciones deben implementar protocolos de control de concurrencia. Un tipo de protocolo es el *basado en bloqueos*, el cual implementa exclusión mutua entre las transacciones en conflicto. Otro tipo de protocolos son los *optimistas* en los cuales la detección y resolución de conflictos se realiza al finalizar la transacción. Si se detecta un conflicto se deshace la transacción y se reintenta (*restart*) tantas veces como resulte necesario hasta que se logre confirmar.

En un *SBDTR* ambos protocolos generan incertidumbre. La variabilidad de tiempos de espera generados por bloqueos y/o las situaciones de reintentos de transacciones dificultan la estimación del *WCET*.

3.5 Administración de Discos y de Memoria Intermedia

Las transacciones en una *BD* involucran persistencia de datos dentro de su secuencia de ejecución y esto significa lecturas y escrituras en disco. Este intercambio de datos con el disco introduce varios puntos de incertidumbre. Por esto muchos *STR* que utilizan *BD* ofrecen la posibilidad de manejar los datos en memoria. Lamentablemente, en grandes *BD* esto no siempre es viable de emplear.

Resulta necesario entonces ampliar los estudios respecto a las políticas de planificación y administración de disco, en las cuales es necesario introducir el concepto de constricciones temporales y así brindar mayores precisiones a las transacciones que requieran lecturas y/o escrituras.

Por otro lado, la gestión de memoria intermedia (*buffer*) en *SGBD* convencionales no es la misma que en *SBDTR*, conjuntamente con las políticas de contención/reemplazo de páginas intermedias. Reemplazar una página con datos de una transacción en curso, significa una demora en esta con el posible riesgo de perder el vencimiento. Por eso es muy importante el estudio de nuevos algoritmos de reemplazo y gestión de *buffers* que contemplen las constricciones temporales de las transacciones involucradas.

3.6 Planificación

Los *SGBD* convencionales difieren de los *STR* en la planificación. En un *SGBD* convencional la unidad de planificación es la transacción, cuya única información temporal con la que cuenta es el instante de arribo. Los registros que necesita procesar se adquieren dinámicamente, lo que dificulta predecir tiempos de espera por bloqueos.

Un *SBDTR* debe dar garantías de puntualidad y predictibilidad. Para esto debe poder planificar considerando más información sobre las transacciones como: la periodicidad, el vencimiento, la criticidad del vencimiento, el *WCET* y los posibles conflictos con otras transacciones concurrentes.

Ser predecible y tener posibilidades de planificación permite conocer antes de iniciar una transacción, si se podrá completar antes del vencimiento. Esto permite garantizar transacciones con vencimientos *críticos*, evita tener que deshacer transacciones con vencimientos *firmes* y permite saber si es aceptable ejecutar transacciones con vencimientos *blandos*.

Como se ha presentado en secciones anteriores hay varios aspectos complejos en la estimación del *WCET*. Para muchos de ellos se presentan alternativas. Por ejemplo: Para un control de concurrencia más predecible considerar la pre-alocación de recursos como alternativa. Para una administración de disco y *buffers* más predecible, utilizar *BD* en memoria o bien utilizar nuevos algoritmos de manejo de memoria intermedia y de planificación de disco que contemplen vencimiento de las transacciones. Para un hilo de ejecución más predecible evitar el uso de bucles cerrados en el código y evitar el uso de estructuras de datos recursivas y dinámicas.

Una técnica que permite convivir con las condiciones de impredictibilidad y estimar el *WCET* de una transacción en un determinado instante, es el *Pre-Fetch*. La misma consiste en que la transacción se ejecuta una vez sin escribir datos. Esto permite medir la demanda computacional y los datos que va a requerir. El problema del *Pre-Fetch* es el costo computacional. Sin embargo se puede optimizar utilizando alternativas como las presentados en [9].

En el estudio de políticas de planificación para *SBDTR* se han tomado como referencia muchos de los planificadores de *STR* basados en prioridades. Se pueden aplicar políticas que se basan en *Rate Monotonic* (*RM*) ([10]), ó *Earliest-Deadline-First* (*EDF*) ([10]). Otras consideran la *función valor* de cada transacción como *Highest-Value-First* ([8]), ó el porcentaje de avance de una transacción como es *Work-Proportional-Approach* ([7]). Existen políticas basadas en el tiempo que se permiten retrasar una transacción sin perder su vencimiento como *Least Slack* ([11]).

A su vez, estas técnicas deben considerar problemas que surgen con el control de concurrencia, por ejemplo, al utilizar protocolos basados en bloqueos se puede presentar el caso de *inversión de prioridades* cuando una transacción de mayor prioridad debe esperar a que una transacción de menor prioridad libere el bloqueo sobre el conjunto de datos en conflicto. Para afrontar estos casos el protocolo más utilizado es el de *herencia de prioridades* por el cual la transacción bloqueante obtiene la prioridad de la transacción bloqueada hasta que se libere el conflicto.

3.7 Correctitud o Puntualidad

Un punto de compromiso en *SBDTR* es definir la relación de importancia entre tres características que se deben garantizar: ser predecible, ser puntual y ser correcto lógicamente. Cuando las restricciones de tiempo son muy exigentes, vale la pena considerar que es mejor producir un resultado parcial antes del vencimiento en lugar del resultado completo después del mismo. Para esto, hay que definir cuáles son los grados de imprecisión o inconsistencia permisibles.

Para abordar este tema es importante considerar los modelos de *computación imprecisa,* los cuales tratan de optimizar el rendimiento del sistema dividiendo a las tareas en una sección obligatoria y otra opcional. Una vez que se realiza la ejecución de la sección obligatoria de todas las tareas se evalúa que parte opcional es la más conveniente ejecutar. La elección se realiza a través del cálculo de una *función valor* o una *recompensa* determinada ([12, 13]).

4 El Avance de las Soluciones en *SBDTR*

El resultado de muchos esfuerzos de investigadores y compañías, ha permitido brindar soluciones a las *BD* en ambientes de *TR* y ha permitido materializar muchos de los conceptos presentados en los puntos anteriores.

Producto de este esfuerzo, se han generado un conjunto de prototipos y de productos comerciales que han marcado el camino y han permitido detectar importantes necesidades del área. Los prototipos experimentales han sido concebidos siguiendo las teorías elaboradas en diversas investigaciones. Los productos comerciales tienen una concepción basada en la demanda del medio, pues han necesitado brindar soluciones a problemas reales, concretos y apremiantes.

El diseño de las soluciones que se han generado se puede clasificar principalmente por dos aspectos: el modelo de ejecución y el modelo de almacenamiento del *SGDB*.

Los *modelos de ejecución* que se han planteado proponen diferentes diseños de procesos en el *STR*. Por un lado el modelo *cliente/servidor* es un modelo de procesos separados, en el cual el *SGBD* es una aplicación independiente con características de *servidor* que interactúa con las tareas *cliente* de *TR* bajo algún protocolo de *solicitud/respuesta* generalmente basado en *SQL*. Quien planifica las transacciones es el servidor *SGBD*. En este modelo el cliente puede ser cualquier aplicación que implemente el protocolo de comunicación. Se usa generalmente en soluciones empresariales, en arquitecturas donde se desconocen las posibles aplicaciones cliente, o en sistemas que manejan gran volumen de datos.

Otro modelo es el de *SGBD incorporado* ó *SGBD incrustado* a las tareas del *STR*. En este caso el *SGBD*, implementado en forma de librerías, está vinculado con la aplicación y corre en el mismo espacio de ejecución que cada una de las tareas. Las solicitudes al *SGBD* son llamadas a rutinas y las transacciones son tareas del *STR*. Este tipo de solución permite acotar la estimación del *WCET* al ámbito de cada tarea y es apropiado para utilizar en sistemas de propósito dedicado. Es uno de los modelos preferidos actualmente para *TR* con vencimientos duros. Las alternativas se muestran en la Figura 4.

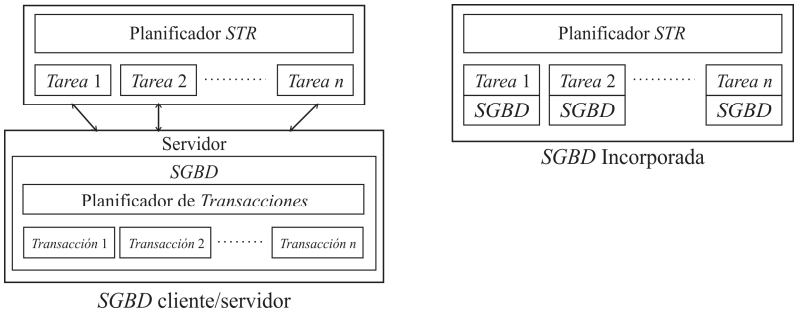

Figura 4. Modelos de Ejecución.

Los *modelos de almacenamiento* de *SGBD* que se han planteado distinguen entre aquellos que administran la *BD* en memoria (*BDM*), en disco ó mixtos. Las *BDM* mejoran la performance y eliminan las incertidumbres generadas por la interfaz de entrada/salida del disco duro. Además es posible garantizar durabilidad en disco u otros medios de persistencia utilizando replicaciones u otras técnicas. Es el modelo de almacenamiento más utilizado en sistemas con tiempos de respuesta reducidos y en los cuales se requiere ser predecible para implementar planificaciones duras o firmes. Dependiendo del volumen de datos a tratar este modelo puede ser muy costoso. Por ello se han planteado variantes mixtas de *BDM* y *BD* en Disco (*BDD*), las cuales permiten diseñar sistemas balanceados entre performance y costo.

Los modelos de almacenamiento presentados se pueden combinar con cualquiera de los modelos de ejecución descritos al principio.

4.1 Los prototipos Generados a Partir de Investigaciones

Se presenta en la Tabla-1 una lista de prototipos que han permitido experimentar soluciones a los diferentes problemas presentados en este trabajo. En dicha tabla se mencionan los prototipos y se referencian los trabajos donde fueron publicados. Para cada uno se describe el criterio para manejar los vencimientos en transacciones (duros, firmes, blandos), el modelo de datos adoptado (Modelo relacional ú orientado a objetos), si tiene características de sistema distribuido, el modelo de almacenamiento (BDM o BDD) y una lista de aspectos que destacan al prototipo.

Prototipo	Vencimi-entos	Modelo Datos	Distribui-do	Almacena-miento	Innovación y/ó Aspectos Destacados
BEEHIVE [14]	Blandos	Orientado a Objetos	Si	Mixto BDM + BDD	Datos con Vencimiento Calidad de Servicio Tolerancia a Fallos Datos Multimedia
STARBASE [15]	Firmes	Relacional	No	BDD	Agrega Funcionalidad de *TR* a un SGBD convencional. Soporte TR en *SO* (RT-Match)
STRIP [16, 17]	Blandos	Relacional	Si	BDM	Abierto y Estándar (Posix y SQL) BD

					Activa (triggers)
DeeDS [18]	Duros y Blandos	Relacional	Si	BDM	BD Activa
REACH [19]	Blandos	Orientado a Objetos	No	BDD	BD Activa

Tabla 1. Algunos de los prototipos surgidos en investigaciones

4.2 Transitando hacia los Productos Comerciales

Existe una variedad de productos comerciales, que quizás no puedan ser considerados *SBDTR* puros desde el punto de vista de las formalidades establecidas en las secciones anteriores. Sin embargo, presentan significativos avances y son las soluciones que se disponen actualmente. En la Tabla-2 se enumeran los productos mas destacados. Todos los productos pueden ser utilizados en sistemas embebidos. Para cada producto se destaca el modelo de procesos que adopta, el modelo de almacenamiento, los sistemas operativos sobre los que ejecuta, los principales clientes que lo usan y las características destacadas:

Producto	Ejecución	Almacena-miento	Sistemas Operativos	Proyectos	Características Destacadas
RDM [20] *Raima Database Manager*	Ambas: C/S e Incrustado	BDM y Mixto	QNX, VxWorks, Linux, WinCE	HP, Lucent, Boeing, 3Com	Alta Disponibilidad Distribuciones adecuadas para sistemas de uso general o dedicado
Polyhedra [21]	C/S	BDM	VxWorks, LinuxWk, WinCE	Lucent, Boeing, Nokia, Motorola	Alternativa com Flash Memory Estándar ODBC / JDBC
eXtremeDB [22]	Incrustado	BDM y Mixto	QNX, VxWorks, LinuxWk, WinCE	JVC, BAE, DirecTV, Siemens	Alta Disponibilidad Alternativa *Kernel Mode*: Se integra al kernel del *SO*.
Oracle TimesTen [23]	C/S	BDM	Linux, Windows, HP-Ux	Lucent, Cisco, Ericsson, NEC	Aplicaciones de gran escala. OLTP (on-line transact.-processing) Posibilidad de Caché de Oracle Server 10g
IBM SolidDB [24]	C/S	BDM y Mixto	AIX, Linux, Windows	Lucent, Cisco, HP, Nokia, Siemens	Aplicaciones de gran escala. OLTP (on-line transact.-processing) Posibilidad de Caché de DB2, Oracle, otros.

Tabla 2. Algunos de los productos comerciales utilizados en ambientes *BD+TR*

Las soluciones presentadas pueden seleccionarse de acuerdo a la necesidad. Para sistemas embebidos con vencimientos exigentes se prefieren los *SGBD* más pequeños y en memoria, como RDM, eXtremeDB y Polyhedra. En sistemas mas generales,

orientados a OLTP (on-line transact.-processing) con alta tasa de transacciones y necesidades de alto rendimiento se prefieren servidores de gran escala como Oracle Times-Ten e IBM SolidDB los cuales además permiten funcionar como caché de servidores SGBD ya en producción mejorando su performance hasta 10 veces.

4.3 Áreas para Desarrollar Investigaciones

En esta sección se identifican algunas líneas de investigación en el área de *SBDTR* las cuales están activas y son potenciales espacios de desarrollo para investigadores que estén formándose en el tema.

- ***Planificación y Predictibilidad*:** En este aspecto hay varios puntos a estudiar como: consistencia temporal de transacciones, planificación para vencimientos críticos, firmes y blandos, técnicas de estimación de *WCET*, modelos de control de concurrencia, políticas para manejo de *buffer* de transacciones con vencimiento, planificación de disco con vencimientos, puntualidad vs. correctitud, etc.
- ***SBDTR en Sistemas Distribuidos***: Existe una creciente demanda sobre *BD* distribuidas con requerimientos de *TR*. Es necesario avanzar sobre las problemáticas de transacciones distribuidas que garanticen la puntualidad punto a punto.
- ***Seguridad**:* Las *SBDTR* necesitan incorporar otras propiedades además de las de *TR*, por ejemplo: tolerancia a los fallos, disponibilidad, seguridad, etc.
- ***Otros aspectos**:* También hay otros aspectos como: lenguajes de *BD* con expresiones de restricciones de tiempo, modelado conceptual con requerimientos temporales, *BD* activas (evento-condición-acción) que consideren constricciones temporales, *QoS* de transacciones, etc.

5 Conclusiones

La creciente complejidad de las aplicaciones de *TR* requiere del uso de *BD* en forma estructurada y organizada. Por otro lado, las aplicaciones empresariales comienzan a imponer requerimientos de respuesta críticos a las transacciones sobre grandes volúmenes de datos. En los sistemas industriales se combinan ambas necesidades debido a que se requiere una alta tasa de transacciones sobre la *BD* en ambientes de TR. Los *SBDTR* son en la actualidad una necesidad existente y en respuesta a esto se han desarrollado numerosas investigaciones al respecto, identificando los problemas existentes, y proponiendo soluciones verificadas por medio de diversos prototipos experimentales. Sin embargo los productos comerciales sólo recientemente están comenzando a implementar estos resultados.

Este trabajo cumple con el objetivo propuesto de introducir al lector con conocimientos de *STR* en los *SBDTR*, y relevar el estado de avance de soluciones concretas, confiando que facilitará el desarrollo de futuras investigaciones.

6 Trabajos Futuros

El carácter de este trabajo ha sido introductorio y panorámico como tránsito necesario hacia el desarrollo de investigaciones más específicas. En el futuro se prevén trabajos orientados hacia los problemas de la planificación de transacciones complejas, de larga duración y con componentes opcionales. Estos casos se presentan por ejemplo en *SBD* empresariales que actúan sobre grandes volúmenes de datos y emplean hilos de ejecución complejos, con varios caminos alternativos, que podrían recorrerse en paralelo y otros que podrían considerarse opcionales sin afectar a los requerimientos mínimos de consistencia.

Una posibilidad es dividir el trabajo de transacciones complejas en sub-transacciones simples. Considerando entonces a la transacción como un flujo de ejecución (*workflow*) de sub-transacciones, con relaciones de precedencia y posibilidades de paralelismo. Esta visión permite un incremento de la concurrencia dentro de la transacción, permite relajar algunas de las propiedades *ACID* y posibilita que una parte de la transacción se ejecute solo si queda tiempo disponible (parte opcional). La confirmación (*commit*) de sub-transacciones depende del estado de la transacción padre. Si una sub-transacción falla pueden tomarse dos acciones: o propagar la falla para que falle todo el conjunto o deshacerse en forma individual (*rollback parcial*). También podrían definirse sub-transacciones autónomas, con *commits* independientes.

En este escenario se propone el relevamiento de algoritmos de planificación de *TR* basados en prioridades que puedan ser aplicables para poder organizar la ejecución de una transacción de *TR* compleja en sub-tareas periódicas concurrentes, cada una con vencimiento individual.

7 Referencias

[1] A. Silberschatz*, et al.*, *Database System Concepts, Sixth Edition* 2010.

[2] J. A. Stankovic, "Misconceptions About Real-Time Computing: A Serius Problem for Next-Generations Systems," *IEEE Computer,* vol. Octubre, pp. 10-19, 1988.

[3] J. A. Stankovic*, et al.*, "Misconceptions About Real-Time Databases," *IEEE Computer,* vol. 32, pp. 29-36, 1998.

[4] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data," in *Proceedings of the 15th VLDB Conference*, 1989.

[5] P. Yu*, et al.*, "On Real-Time Databases: Concurrency Control and Scheduling," in *Special Issue on Real-Time Systems*, Proceedings of IEEE, 1994, pp. 140-157.

[6] K. Ramamritham, "Real Time Databases," *International Journal of Distributed and Parallel Databases,* vol. 1, pp. 199-226, 1993.

[7] M. Xiong*, et al.*, "Maintaining Temporal Consistency: Issues and Algorithms," in *Proceedings of International Workshop on Real-Time Database Systems*, 1996, pp. 2-7.

[8] E. D. Jensen*, et al.*, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of Real-Time Systems Symposium,* pp. 112-122, 1985.

[9] P. O'Neil, *et al.*, "Towards Predictable Transaction Executions in Real-Time Database Systems," University of Massachusetts1992.
[10] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM,* vol. 20, pp. 46-61, 1973.
[11] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," in *Proceedings of the 14th VLDB Conference*, 1988.
[12] J. M. Urriza, *et al.*, "Optimización on-line de Sistemas de Tiempo Real con Computación Imprecisa Basados en Recompensas.," in *32 JAIIO AST2003*, Buenos Aires, Argentina, 2003.
[13] R. M. Santos, *et al.*, "Heuristic use of Singularities for On-Line Scheduling of Real-Time Mandatory/Reward-Based Optional Systems," in *14th Euromicro Conference on Real-Time Systems*, Vienna, Austria, 2002, pp. 103-110.
[14] J. Stankovic, *et al.*, "BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications," in *in Real-Time Databases and Information Systems*, 1997, pp. 409--422.
[15] Y.-K. Kim and S. H. Son, "Developing A Real-Time Database: The Starbase Experience," 1997.
[16] B. Adelberg, *et al.*, "Overview of the Stanford Real-Time Information Processor (STRIP)," *ACM SIGMOD Record,* vol. 25, pp. 34-37, 1996.
[17] B. Adelberg and R. Pease, "Strip: A Soft Real-Time Main Memory Database for Open Systems," 1997.
[18] A. H. Eriksson, *et al.*, "DeeDS Towards a Distributed and Active Real-Time Database System," *ACM SIGMOD Record,* p. 25, 1996.
[19] A. P. Buchmann, *et al.*, "The REACH active OODBMS," in *Conference on Management of Data*, Proceedings of the 1995 ACM SIGMOD international conference on Management of data 1995.
[20] Raima. Abril 2010). *RDM* Available: http://www.raima.com/products
[21] Enea. Abril 2010). *Polyhedra*. Available: http://www.enea.com/polyhedra
[22] McObject. Abril 2010). *eXtremeDB* Available: http://www.mcobject.com/
[23] Oracle. Abril 2010). *Oracle TimesTen*. Available: http://www.oracle.com/database/timesten.html
[24] IBM. Abril 2010). *IBM Solid DB*. Available: http://www-01.ibm.com/software/data/soliddb/