Diseño y Evaluación de un Algoritmo Adaptable Control de Concurrencia

Wenceslao Enrique Palma Muñoz
wenceslao.palma@ucv.cl
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso
Fax: (56)(32) 274097
Valparaíso, Chile

Raúl Monge Anwandter
rmonge@inf.utfsm.cl
Departamento de Informatica
Universidad Tecnica Federico Santa Maria
Fax: (56)(32) 797513
Casilla 110-V, Valparaíso, Chile

Resumen

El rendimiento de un Sistema de Procesamiento de Transacciones puede ser afectado, entre otras cosas, por el Método de Control de Concurrencia utilizado, donde los esquemas optimistas ofrecen una alternativa a métodos basados en bloqueo. Ambos métodos poseen un mejor desempeño bajo condiciones operacionales opuestas. Sin embargo, dado que en general el comportamiento operativo es difícil de predecir, usualmente los servicios de procesamiento de transacciones optan por uno de ellos. Este trabajo plantea la posibilidad de diseñar un método de control de concurrencia adaptable, el cual puede decidir sobre el mejor método disponible a utilizar ante los distintos escenarios operacionales a los cuales se vería enfrentado el sistema y mejorar el rendimiento del Sistema de Procesamiento de Transacciones. La metodología utilizada para la obtención de resultados corresponde a simulación de eventos discretos adoptándose un enfoque orientado a los procesos, el cual beneficia, dado su modelo de programación, la simulación de sistemas complejos. Mediante el análisis de los resultados se determina bajo qué condiciones es aconsejable conmutar de método. Los resultados muestran que bajo ciertos niveles de contención es preferible un proceso de conmutación y que posteriormente los tiempos de respuesta mejoran manteniéndose muy cercanos a un *ideal*.

Palabras clave: Control de Concurrencia, Transacciones, WISBD.

1. Introducción

El control de concurrencia (CC) [1] es la actividad de coordinación de acciones pertenecientes a procesos que operan en paralelo, acceden a datos compartidos y que potencialmente pueden interferir mutuamente. Básicamente las interferencias entre procesos que acceden a datos compartidos se manifiestan mediante operaciones (tipicamente read/write) que pueden dejar a un sistema en un estado inconsistente. De acuerdo con [2], existen 3 maneras mediante las cuales operaciones de distintos procesos que acceden a los mismos datos pueden ocasionar problemas de consistencia y que pueden ser resumidos con la siguiente afirmacion [3]: se dice que 2 operaciones estan en conflicto si ellas acceden al mismo dato, son usadas por diferentes procesos y una o ambas operaciones corresponden a una escritura.

Ante lo anterior, es deseable que los procesos se ejecuten de manera atómica. Para cumplir con dicha condición las operaciones que componen un proceso se agrupan en lo que se denomina una transacción. Cuando dos o más transacciones se ejecutan de manera concurrente, sus operaciones se ejecutan en forma intercalada. De este modo se pueden interferir y producir un estado inconsistente en el repositorio de datos [2].

Una manera de evitar problemas es no permitir ejecuciones intercaladas, así las transacciones se ejecutan en forma serial, es decir, para cada par de transacciones todas las operaciones de una se ejecutan antes que cualquier operación de la otra. Sin embargo, la falta de concurrencia involucra una mala utilización de los recursos y malos

tiempos de respuesta, por lo cual es necesario ampliar la clase de ejecuciones permitidas de manera que se permita concurrencia y el efecto de la ejecución sea el mismo que una serial. Tales ejecuciones se denominan serializables.

Con todo, ya que las ejecuciones seriales son correctas y como una ejecución serializable tiene el mismo efecto que una ejecución serial entonces esta última también es correcta [3].

Debido a que existen algoritmos de CC que poseen buen desempeño bajo condiciones operacionales opuestas y además el escenario es difícil de predecir, esta trabajo plantea la posibilidad de estudiar y diseñar un algoritmo de CC adaptable el cual permita a un sistema de procesamiento de transacciones tomar la decisión, en tiempo de ejecución, sobre cuál es el algoritmo que mejor se ajusta al escenario operacional actual [12], teniendo como objetivo mejorar su desempeño. La metodología usada para validar la hipótesis corresponde a simulación de eventos discretos adoptándose un enfoque orientado a los procesos, el cual beneficia, dado su modelo de programación, la simulación de sistemas complejos.

Los resultados muestran que, luego del proceso de conmutación, es posible lograr una mejora en los tiempos de respuesta obtenidos. Sin embargo, esto no sucede para todos los niveles de contención, observándose que el mayor beneficio se obtiene ante la presencia de escenarios de gran contención y alto grado de concurrencia. El presente trabajo continua de la siguiente manera, en la sección 2 se muestran los modelos de adaptabilidad, los algoritmos escogidos y sus consideraciones al someterlos a un proceso de adaptación. En la sección 3 se presentarán el modelo del sistema a simular, sus parámetros relevantes y cargas de trabajo. En la sección 4 se presentan y analizan los resultados de la simulación, para en la sección 5 terminar con conclusiones y perspectivas de trabajo futuro.

2. Algoritmo Adaptable de Control de Concurrencia

Inicialmente en [4] se plantea la posibilidad de un mecanismo de CC adaptable basado en lo que se denomina la temperatura de un objeto, lo cual provoca que distintos objetos sean accedidos por distintos métodos, sin embargo nuestra preocupación se enfocará hacia un mecanismo que manifieste una adaptación por transacción.

Lograr que luego del proceso de adaptación se dé origen a ejecuciones serializables es lo más importante en un algoritmo adaptable de control de CC, ya que de este modo es posible asegurar que el sistema pasa de un estado consistente a otro.

2.1. Modelos de Adaptabilidad

Existe un modelo formal para Sistemas Adaptables de Procesamiento de Transacciones [5]. En él se distinguen básicamente tres maneras de realizarlo:

1. Adaptación de estado genérico. Bajo este esquema es necesario desarrollar una estructura de datos común a todos los planificadores de modo que todos usen la misma estructura de datos, la cual contiene información relevante para ambos (ver figura 1). Tiene como ventaja su simplicidad. Sin embargo los algoritmos pueden ser menos eficientes usando una estructura de datos común.

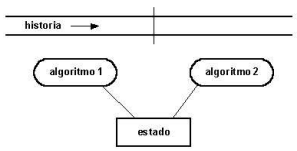


Figura 1: Estado genérico

2. Adaptación con conversión de estado. Los datos mantenidos por diferentes planificadores contienen la misma información en diferentes formas, si bien en algunos casos no es posible que distintos algoritmos puedan

compartir la misma estructura de datos, si es posible convertir datos de uno a otro. Esto sugiere la existencia de un proceso adaptable que se lleva a cabo mediante una rutina de conversión de la información de estado requerida por el nuevo algoritmo (ver Figura 2).Tiene como ventaja la utilización de su propia, natural y eficiente estructura de datos por parte de cada algoritmo. Sin embargo, es necesario un algoritmo de conversión para cada par de algoritmos.

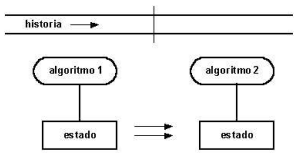


Figura 2: Conversión de estado

3. Adaptación de estado sufijo-suficiente. La idea básica de este enfoque se centra en permitir que durante el proceso adaptable se permite que las transacciones sigan con sus operaciones siempre y cuando el "nuevo" y el "viejo" algoritmo lo permitan. Entonces existirán las siguientes partes de la historia serializable (ver Figura 3).

   - Una historia "vieja"que es aceptable por el método A (H_A).
   - H_{AB} es una parte común tanto, para la historia "vieja" como para la "nueva" y que es aceptada por ambos métodos.
   - H_B es la "nueva"historia aceptada por B.

   Tiene como ventaja que el traslape entre los algoritmos permite un alto de grado de concurrencia.Sin embargo es difícil determinar cuando H_{AB} terminará.

2.2. Algoritmos de Control de Concurrencia escogidos

Pensando en la conmutación de algoritmos de CC, se escogieron algoritmos que pertenecen a un enfoque pesimista y a uno optimista. Se considera que la única forma en que una transacción aborte se debe a la presencia de deadlock (en el caso del algoritmo pesimista) y la falla en el proceso de validación (en el caso del algoritmo optimista). Cuando una transacción aborta su ejecución se reinicia con el mismo conjunto de datos.

Two phase locking (2PL): La filosofía del algoritmo [3] asocia un lock con cada operación. En caso de que la transacción obtenga un lock sobre el dato, la operación procede, de lo contrario debe esperar hasta que el dato sea liberado. Básicamente todo sistema que trabaje bajo la filosofía de bloqueo puede caer en una situación de deadlock, lo cual no es deseable ya que impide el término de los procesos. En la simulación la estrategia de detección de deadlock es precisa [6] y se basa en la utilización de wait-for graph (WFG).

La estrategia para la detección de deadlock se conoce como detección continua, es decir, cada vez que se solicita un lock se verifica la existencia de deadlock. Además se considera el evitar upgrade locks en una transacción, ya que se sabe que es una situacion que provoca deadlock. Por último se implementó una variante de 2PL conocida como 2PL Estricto ya que posee propiedades más deseables ante la presencia de fallas.

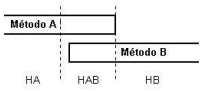


Figura 3: Estado sufijo-suficiente

OCC (OPT): corresponde a un algoritmo optimista [8] que serializa la ejecución intercalada de transacciones en base a una marca de tiempo que asigna el Administrador de Transacciones cuando recibe una petición de validación. Este método es libre de deadlock, sin embargo las transacciones pueden abortar cuando dan origen a ejecuciones no serializables.

2.3. Algoritmo Adaptable

El algoritmo adaptable construido está basado en el método de conversión de estado, de este modo cada algoritmo trabaja con sus estructuras de datos y en el momento de la conversión se consideran los siguientes criterios:

- 2PL a Optimista. Para convertir es necesario considerar que en este caso no son útiles los datos escritos por transacciones ya comprometidas bajo 2PL, ya que las que actualmente tienen un lock de lectura sobre dichos datos lo han obtenido luego de que la transacción que poseía el lock se ha comprometido. Luego es necesario considerar sólo las transacciones que están activas.
  Para esto se utiliza :

```
for D in Lock_Table {
   release_lock(D);
   release_waiting();
}
```

  Donde :

  - release_lock(D). libera el lock sobre el dato D.
  - release_waiting(). permite que las transacciones que esperaban por el dato salgan de su estado de espera. Y además construye el *readset* y *writeset* para cada una de éstas.

- Optimista a 2PL. Este proceso considera sólo a las transacciones activas, es decir aún no comprometidas, sin embargo se puede dar el caso que una transacción no comprometida ya pasó su etapa de validación. Esto es muy importante ya que transacciones que están activas, pero en etapa de validación, no se consideran para conmutar a 2PL debido a que están a punto de terminar su ejecución. Además, dado que se conmuta hacia un algoritmo más restrictivo es necesario verificar en este proceso que se están generando ejecuciones serializables, por lo tanto es posible que algunas transacciones que violan dicha condición sean abortadas.
  Para esto se utiliza :

```
for T in Active_Transactions {
  if (T.phase != VALIDATE)
  for P in T.operations {
     if (!Lock_table.get_lock(P))
        restart(T);
  }
}
```

  Luego de cada proceso de conmutacion, las transacciones realizan sus operaciones bajo el nuevo algoritmo sin enterarse de dicho cambio. Sólo envían sus operaciones y el planificador decide qué filosofía de CC utilizar.

2.4. Proceso de conmutación.

Para determinar cuándo es pertinente una conmutación de algoritmo de CC, el sistema evalúa a intervalos el tiempo de respuesta promedio de las transacciones. Inicialmente se considera un tiempo de respuesta deseable, el cual puede cambiar durante la ejecución del sistema luego de un proceso de conmutación. Se consideró el tiempo de respuesta como dato relevante ya que permite cuantificar cuál es el efecto que tiene la ejecución del sistema sobre los tiempos de espera de los usuarios.
Para evaluar el comportamiento del sistema y determinar si se justifica un proceso de conmutación se utiliza la medida llamada $performance\_index$ [13], la cual se calcula de la siguiente manera:
$performance\_index = tiempo\_respuesta\_actual/tiempo\_respuesta\_deseable$

Por lo anterior si:

- $performance\_index > 1$: indica que el tiempo de respuesta está por sobre lo esperado. Por lo tanto es necesario, de acuerdo al escenario operacional actual, un proceso de conmutación de algoritmo de CC con el objetivo de mejorar los tiempo de respuesta.
- $performance\_index = 1$: indica que el tiempo de respuesta promedio actual corresponde al óptimo.
- $performance\_index < 1$: indica que el tiempo de respuesta promedio es mejor que lo esperado.

3. Simulación

Para la simulación del sistema se construyó un simulador de eventos discretos [9] [10] y el enfoque utilizado está basado en objetos activos. Dicho enfoque fue escogido ya que se obtiene una correspondencia muy cercana entre el código de la simulación y el sistema real [11], al destacar preferentemente la lógica de los componentes del sistema. Junto al modelo del sistema se muestran los parámetros relevantes y las cargas de trabajo diseñadas.

3.1. Modelo del Sistema

Desde el punto de vista conceptual un sistema de procesamiento de transacciones [3] consta de los siguientes componentes :
Administrador de Transacciones: recibe las operaciones pertenecientes a las transacciones y las envía al planificador de CC.
Planificador: este componente materializa un algoritmo de CC controlando la ejecución concurrente de las transacciones, de modo que se produzcan ejecuciones serializables y recuperables ante fallas. Luego de recibir una operación, el planificador puede tomar una de las siguientes acciones :

1. Ejecutar : si no existe problema desde el punto de vista de la serialización, la operación recibida se ejecuta, lo cual involucra el acceso al dato mediante el Administrador de Datos.
2. Rechazar : si existe algun tipo de problema, como la presencia de dealock o bien una falla en el proceso de validación o compromiso, la operación se rechaza y eventualmente la transacción debe ser reiniciada.
3. Retrasar : en filosofías de CC basadas en bloqueo el acceso al dato puede esperar en una cola hasta el momento de su posible ejecución o rechazo.

Administrador de Datos: es el encargado de acceder a los datos y de este modo retornar a la transacción mediante el planificador el valor leído o escrito por ésta.

Además desde el punto de vista de la simulación, este modelo se ve aumentado con los siguientes componentes (ver Figura 4):

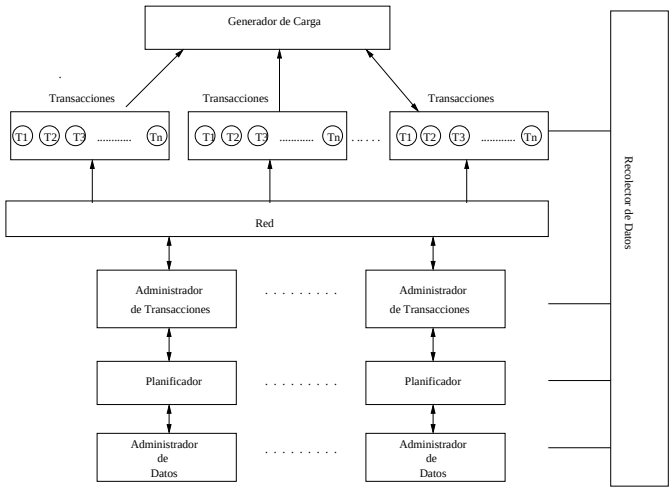


Figura 4: Modelo del Sistema a simular

Generador de carga: Permite configurar el sistema de acuerdo a parámetros que se consideran relevantes para caracterizar el sistema desde el punto de vista de la carga de trabajo.
Recolector de Datos: permite que cualquier componente del sistema registre su actividad en un archivo de log, lo cual es útil al momento de evaluar el comportamiento.

3.2. Parámetros relevantes y Cargas de Trabajo

Cada transacción consiste de una secuencia de operaciones de lectura y escritura determinadas por un generador de carga, el cual permite simular diferentes patrones de acceso y niveles de contención.

Cada vez que se genera una transacción es necesario determinar cuántos y cuáles son los datos a los cuales accede y además a qué operación está relacionada el acceso.También es necesario definir con que grado de concurrencia (MPL) se ejecutará la simulación. Con todo, es necesario definir una serie de parámetros que permitan configurar un escenario de simulación.

Se determinó realizar la siulación bajo un esquema de múltiples clientes y un servidor, con esto no es necesario un protocolo de compromiso (2PC). Suponemos que independente del algoritmo de CC el 2PC agrega la misma cantidad de mensajes por lo tanto no tiene un impacto significativo en el desempeño del sistema.

3.2.1. Carga de Trabajo

Una carga de trabajo permite capturar patrones de acceso y así asociar un escenario de simulación a un escenario real. Considerando los trbajo ya realizados en el área de CC y de acuerdo a los propuesto en [7], se ha decidido implementar las cargas de trabajo PRIVATE, HOTCOLD y HICON. Los elementos comunes a cada carga de trabajo corresponden a lo que se denomina un 80-20, es decir, los accesos asociados a una carga de trabajo corresponden a un 80 % de un mismo tipo y el 20 % restante a otro. Este tipo de patrones de acceso es común a muchos trabajos [7] y se ha convertido en un estándar *de facto* para el estudio de algoritmos de control de concurrencia.
La carga de trabajo PRIVATE se utiliza para simular el comportamiento de una aplicación en la cual los usuarios trabajan sobre zonas disjuntas; en cada zona el proprietario realiza actualizaciones y eventualmente accede a una zona compartida del tipo sólo lectura. Por otra parte la carga de trabajo HOTCOLD permite simular un comportamiento en el cual los procesos que acceden a lo datos tiene algún grado de intersección en el acceso, lo que puede provocar accesos compartidos en forma moderada y asi generar un escenario de baja contención. La carga de trabajo HICON está diseñada para simular un escenario de gran contención en donde los procesos no tienen una

zona exclusiva de acceso, sino que es compartida.
Cada carga de trabajo posee parámetros que permiten configurar niveles de contención. El detalle de cada una es el siguiente:

PRIVATE : Cada transacción posee una región privada que accede el 80 % de las veces y el 20 % restante accede a una región compartida de sólo lectura (ver Figura 5).

HOTCOLD : Cada transacción posee una región privada que accede el 80 % de las veces, el resto de la región incluye regiones privadas de otras transacciones y es accedida el 20 % restante (ver Figura 6).

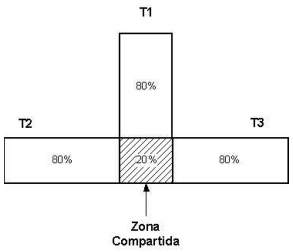


Figura 5: Carga de Trabajo PRIVATE

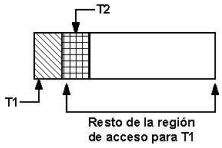


Figura 6: Carga de Trabajo HOTCOLD

HICON : Aquí no existen regiones privadas a cada transacción. Toda la región de accesos se divide en 2, una subregión que corresponde al 20 % de la región total se denomina HOT y es accedida un 80 % de las veces por cada transacción, y la otra subregión que corresponde al 80 % de la región total se denomina COLD y es accedida un 20 % de las veces por cada transacción (ver Figura 7).

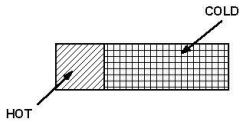


Figura 7: Carga de Trabajo HICON

3.3. Preparación del experimento

Para cada carga de trabajo se generan datos experimentales considerando un determinado rango de MPL. Para cada punto de un gráfico se consideró el promedio perteneciente a un intervalo de confianza del 95 %.
Para determinar los efectos de la latencia se utilizó el tiempo de respuesta y la cantidad de conflictos que se generan; básicamente un conflicto se asocia al reinicio de una transacción debido a deadlock o bien a una falla en la etapa de validación.
Por otro lado, se mostrará cuál es la real mejora que se puede obtener, para ello se utilizó el throughput, en transacciones por segundo (TPS), alcanzado por cada algoritmo. Con esto es posible cuantificar la ganancia de la conmutacion de algoritmos. Si consideramos *throughput_a* y *throughput_b*, para dos algoritmos *a* y *b* respectivamente, las siguientes expresiones determinarán la conveniencia de conmutar de algoritmo:

- si *throughput_a* >*throughput_b*, la mejora estará dada por
  (throughput_a - throughput_b)/throughput_b
- si *throughput_b* >*throughput_a*, la mejora estará dada por
  (throughput_b - throughput_a)/throughput_a

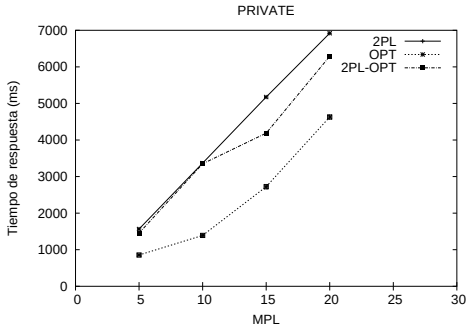


Figura 8: Tiempo de Respuesta para la carga PRIVATE

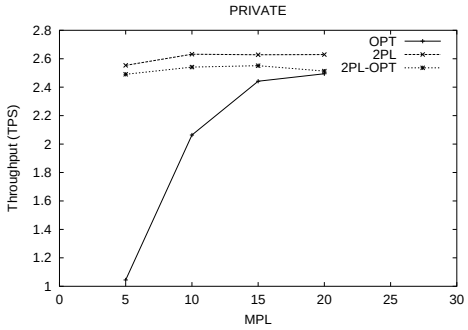


Figura 9: Throughput para la carga PRIVATE

Por último, el indicador que permite decidir cuál es el momento oportuno para realizar un proceso de conmutación de algoritmo de CC corresponde a $performance\_index$. Vale la pena destacar que mediante este indicador es posible manejar ciertos niveles de tolerancia en cuanto a la degradación del tiempo de respuesta, sin embargo el valor que se utilizó para determinar el momento oportuno de la conmutación fue > 1. En los gráficos correspondientes se muestra cuál debe ser el valor ideal de este indicador.

4. Resultados de la simulación y análisis

4.1. Carga de Trabajo PRIVATE

En este caso el tiempo de respuesta asociado a cada algoritmo se ve influido por los retardos inherentes a su filosofía de control de concurrencia. En la figura 8 se puede apreciar que 2PL posee tiempos de respuesta que son superiores a los presentados por OPT y que al realizar una conmutación del tipo 2PL-OPT los tiempos de respuesta mejoran respecto a 2PL, sin embargo el throughput es levemente inferior luego de la conmutación de algoritmos (ver figura 9).

Si consideramos la ganancia real ver figura 10 podemos apreciar que bajo esta carga, si bien los tiempos de respuesta mejoran no resulta tan conveniente la conmutacion 2PL-OPT.

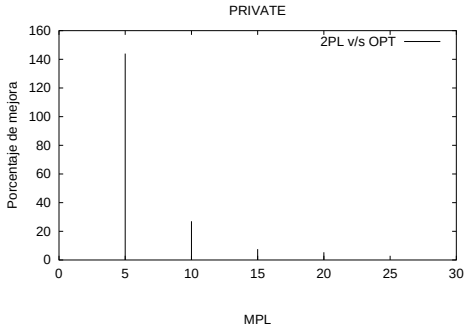


Figura 10: Mejora para la carga PRIVATE

4.2. Carga de Trabajo HOTCOLD

Claramente los tiempos de respuesta mejoran luego de aplicar la conmutación de algoritmo (ver figura 11), lo cual es respaldado por la figura 13 en donde se aprecia una disminución en la cantidad de conflictos que se generan durante su ejecución. Sin embargo el real aporte de la conmutación de algoritmo se concreta con un MPL

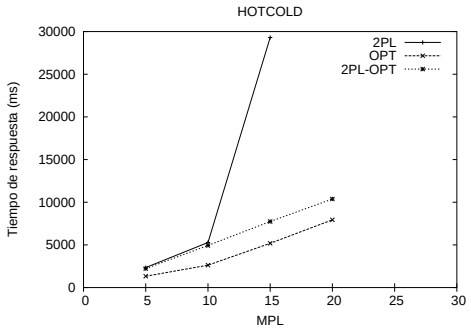


Figura 11: Tiempo de Respuesta para la carga HOTCOLD

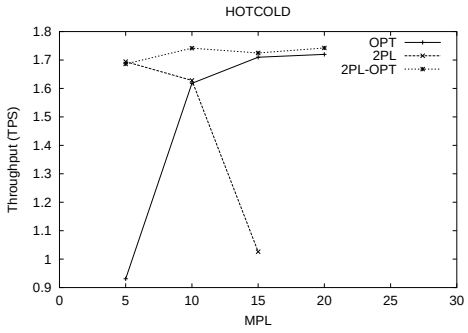


Figura 12: Throughput para la carga HOTCOLD

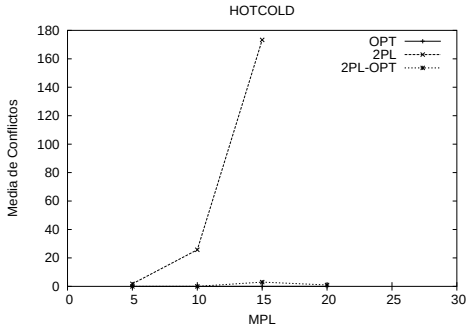


Figura 13: Conflictos para la carga HOTCOLD

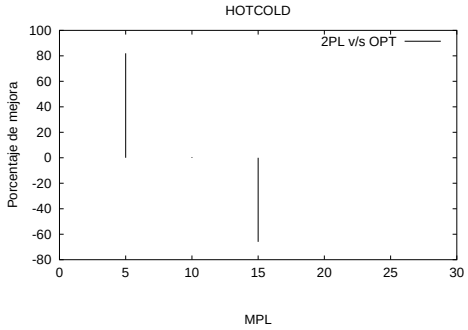


Figura 14: Mejora para la carga HOTCOLD

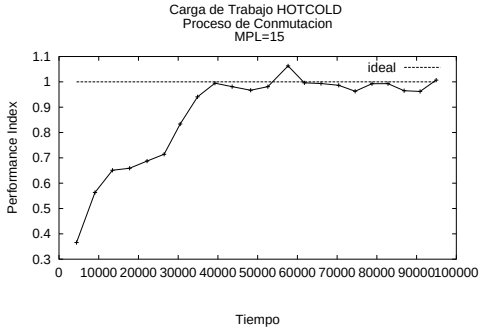


Figura 15: Proceso de conmutación para la carga HOTCOLD

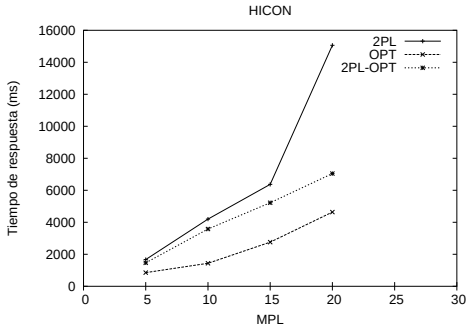


Figura 16: Tiempo de Respuesta para la carga HICON

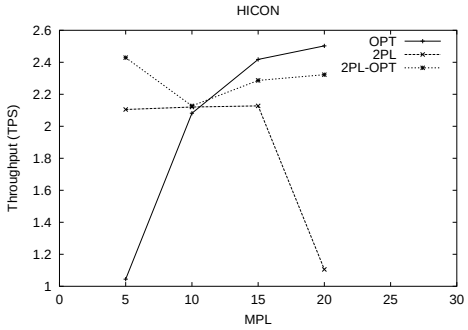


Figura 17: Throughput para la carga HICON

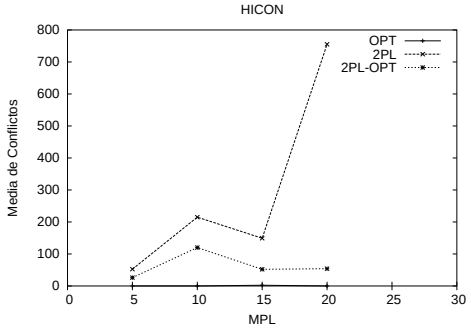


Figura 18: Conflictos para la carga HICON

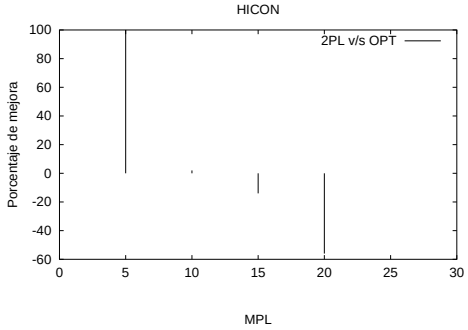
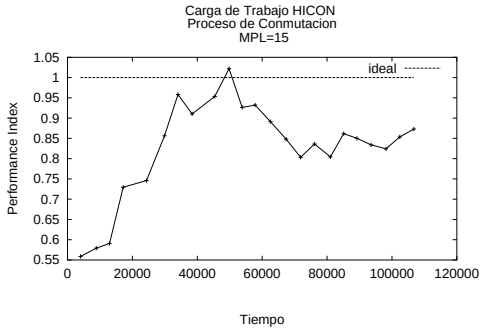


Figura 19: Mejora para la carga HICON



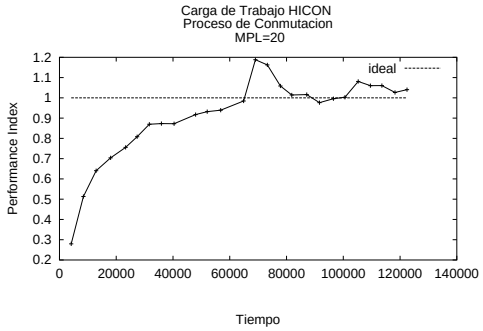


Figura 20: Proceso de Conmutación para la carga HICONFigura 21: Proceso de Conmutación para la carga HICON

de valor 15, en donde el throughtput presenta una mejora (ver figura 12. En la figura 14 se aprecia que el aporte de 2PL es negativo respecto de OPT (con MPL 15), por lo tanto la conmutación ayudará. El momento preciso de la conmutación se muesta en la figura 15, allí se aprecia que luego de la conmutación el nuevo algoritmo tiende a dejar los tiempos de respuesta muy cerca de su valor ideal.

4.3. Carga de Trabajo HICON

Los tiempos de respuesta así como la cantidad de conflictos disminuyen luego de la conmutación de algoritmo (ver figuras 16 y 18 respectivamente). Lo importante es destacar que el real aporte de la conmutación se realiza cuando el nivel de concurrencia es más alto, es decir mayores valores asociados a MPL, allí según el grafico de mejoras (ver figura 19), el aporte se realiza para MPL de valor 15 y 20, lo cual se respalda con el aumento del throughput (figura 17) luego de la conmutación. El momento preciso de la conmutacion se puede apreciar en las figuras 20 y 21. En ambos casos existe una consistencia con lo que muestra la figura 19, ya que luego de conmutar

de algoritmo los tiempos de respuesta se mantienen muy cerca del ideal.

5. Conclusiones y trabajo futuro

Se ha presentado un algoritmo de control de concurrencia y un análisis de los distintos escenarios operacionales a los cuales se puede ver enfrentado un sistema de procesamiento de transacciones. Básicamente lo que se mostró fue cuál es el escenario operacional que mejor calza con una filosofía adaptable de control de concurrencia, basándose en el tiempo de respuesta, conflictos y throughput.
Claramente bajo ambientes de gran contención resulta deseable un algoritmo adaptable de CC, sin duda que esto refleja para el par de algoritmos escogidos que el reinicio de transacciones que abortan debido a deadlock provoca un empeoramiento en el desempeño del sistema; la causa puede tener su origen en la detección de deadlock, la cual es continua. Sabemos que detecciones de otro tipo como las basadas en timeout sufrirán de otras desventajas, tales como el afinamiento del timeout. Por otro lado, el reinicio a causa de dealock provoca que la transacción debe liberar locks adquiridos hasta el momento. Con todo, el simple reinicio que presenta un algoritmo del tipo OPT provocará mejoras en el desempeño del sistema.
En escenarios de gran contención el retardo en el reinicio de una transacción es un parámetro relevante, ya que de este modo se evita que la misma transacción se reinicie continuamente. En los casos en los cuales no es razonable realizar una conmutación del tipo 2PL-OPT, es posible plantear que una conmutación del tipo OPT-2PL mejorará el desempeño del sistema lo cual sólo se puede realizar cuando en la conmutacion existe un aumento del throughput.
Luego del proceso de conmutación los tiempos de respuesta se mantiene muy cerca del tiempo de respuesta ideal, sin embargo como se mostró en ambientes de gran contención es perfectamente posible que durante un instante de la ejecución los tiempos de respuesta se encuentren por sobre lo ideal. Esto se debe a que con el nuevo algoritmo de CC transacciones que estaban sin finalizar y que por lo tanto ya tiene un tiempo de respuesta alto puedan hacerlo rápidamente.
Para realizar una implementación en un sistema real de lo propuesto en este trabajo se debe tener en cuenta que:

- No todos los escenarios operacionales son adecuados para implementar un algoritmo adaptable de control de concurrencia.
- Dado el modelo de conmutación adoptado existe una desventaja ya que es necesario implementar un algoritmo adaptable para cada par de algoritmos de control de concurrencia escogidos.
- Una implementación que adhiera al paradigma de programación orientada a objetos es importante ya que permite garantizar transparencia en la comunicación de los distintos componentes de un Sistema de Procesamiento de Transacciones.

Desde el punto de vista del desarrollo de la simulación es destacable que el modelo de programación basado en objetos activos entrega un mayor beneficio al destacar la lógica de los objetos que se simulan por sobre los aspectos de la simulación. Sin embargo esto tiene un costo, ya que cada objeto a simular es un objeto activo (thread), esto afecta claramente el tiempo de ejecución de la simulación, lo cual se comprobó durante la captura de resultados para las distintas cargas de trabajo. Se trató de mejorar este tiempo con compilación JIT, pero esto no mejoró mas allá del 10 % el tiempo de ejecución.

5.1. Trabajo Futuro

Reinicio adaptable, atributos en tiempo de ejecución. En ambientes de gran contención el retardo en el reinicio de una transacción es un parámetro dificil de manejar, sin embargo es posible plantear un esquema en el cual el tiempo de retardo sea adaptable, considerando como elementos de juicio los atributos que adquiere una transacción durante su tiempo de ejecución.
Ajuste del tiempo de respuesta deseado. Para el proceso de conmutación se utilizó un *tiempo de respuesta deseado*,

el cual se mantiene fijo durante la ejecución. No se consideró en este trabajo la posibilidad de un ajuste de dicho tiempo en la medida que se realizan procesos de conmutación. Si bien en los resultados obtenidos muestran que el tiempo de respuesta, luego de la conmutación, se mantiene muy cerca del ideal es perfectamente posible obtener tiempos de respuesta menores.

Administración de objetos activos El tiempo de simulación para ambientes de gran contención es bastante alto, lo cual demora la obtención de resultados. La mejora en este caso debido al enfoque de simulación utilizado puede ser el resultado de una mejora al esquema de adiministración de los objetos activos. Considerando que tan sólo unos pocos objetos se encuentran en ejecución durante el mismo tiempo, un esquema del tipo pool-threading puede ser conveniente.

Referencias

[1] Bharat Bhargava. *Concurrency Control In Databasa Systems*. Transaction on Knowledge and Data Engineering, Vol. 11, No 1,3-16,January 1999.

[2] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques,* Morgan-Kaufmann,, San Mateo, Calif., 1992.

[3] A. Bernstein, V. Hadzilacos and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] Robert Gruber. *Temperature-Based Concurrency Control*. MIT Laboratory for Computer Science, 1995.

[5] Bharat Bhargava, Riedl John. *A model for Adaptable Systems for Transaction Processing*. Transactions on Knowledge and Data Engineering, Vol 1. N 4, Diciembre 1989.

[6] Anita van Engen, Michael Bradshaw, Nathan Oostendorp. *Extending Java to Support Shared Resource Protecion an d Deadlock Detection in Threads Programming*. ACM CrossRoad Magazine.

[7] Robert Gruber. *Optimism vs Locking : A Study of Concurrency Control for Client-Server Object-Oriented Databases*. PhD Dissertation, MIT, Febrero 1997.

[8] A. Adya, R. Gruber, B. Liskov , and U. Maheshwari. *Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks*. In ACM SIGMOD, Mayo 1995.

[9] José M. Garrido. *Discrete-Event Simulation with Java. A Practical Introduction*. Kluwer Academic, Series in Computer Science,2001.

[10] Thomas J. Schriber, Daniel T. Brunner. *Inside Discrete-Event Simulation Software: How it Works and Why It Matters*. Proceedings of the 1998 Winter Simulation Conference.

[11] Richard Kilgore. *Object-Oriented Simulation with Java*. Proceedings of the 2002 Winter Simulation Conference.

[12] Wenceslao Palma, Raúl Monge. *Hacia un Sistema Adaptable de Procesamiento de Transacciones*. IV Worskshop de Sistemas Distribuidos y Paralelismo, 2000.

[13] J. Aman, C.K. Eilert, D. Emmes, P. Yocom, D. Dillenberger. *Adaptive Algorithms for managing a distributed data processing workload*. IBM Systems Journal, Volume 36, N 2, 1997.