Lottery Scheduling: Flexible Proportional-Share Resource Management

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Lottery scheduling es un mecanismo aleatorio de asignación de recursos. Los permisos de los recursos se representan por medio de tiquetes de loteria(*lottery tickets*). Cada asignación se determina por medio de la realización de una lotería: el recurso de otorga al cliente con el tiquete ganador. Esto permite la asignación efectiva de los recursos a los recursos competidores en proporción al número de tiquetes que tengan. *Resource Rights:* los tiquetes de lotería encapsulan los permisos de los recursos que son (1) Abstractos porque cuantifican los permisos de los recursos independientemente de los detalles de la máquina. (2) Relativos porque la fracción de un recurso que ellos representan varía dinámicamente en proporción a la conexión para ese recurso. (3) Uniformes porque los permisos para recursos heterogéneos pueden ser representados de forma homogénea como tiquetes. **Lotteries:** *Scheduling* por lotería es probabilísticamente justo. La asignación esperada de recursos a los clientes es proporcional al número de tiquetes que tienen. Dado que el algoritmo de *scheduling* es aleatorio, las proporciones asignadas reales no son garantizadas a coincidir las proporciones esperadas. El número de loterías ganadas por un cliente tiene un distribución binomial. La probabilidad p de que un cliente que tiene t tiquetes vaya a ganar un lotería de T tiquetes es $p = t/T$. Luego de n loterías idénticas, el número esperado de ganes w es $E[w] = np$, con varianza $\sigma_w^2 = np(1-p)$. El coeficiente de variación para la proporción observada de ganes es $\sigma_w/E[w] = \sqrt{(1-p)/np}$. De esta forma, el *throughput* de un cliente es proporcional a su asignación de tiquete, con precisión de que mejora con $\sqrt{n}$. El número esperado de loterías n que un cliente tiene que esperar antes de su primer gane es $E[n] = 1/p$ con varianza $\sigma_n^2 = (1-p)/p^2$. De esta forma, el tiempo de repuesta promedio de un cliente es inversamente proporcional a su asignación de tiquete.

Modular Resource Management. Técnicas para implementar políticas gestión de recursos con *lottery tickets:* *Ticket Transfers:* son transferencias explícitas de tiquetes de un cliente a otro. Pueden ser usados en cualquier situación donde un cliente bloquee debido a alguna dependencia. Los clientes también tienen la habilidad de dividir transferencia de tiquetes a través de múltiples servidores en donde ellos pueden estar esperando. *Ticket Inflation:* es una alternativa a transferencia de tiquetes explícitos en donde un cliente puede escalar los permisos de sus recursos por medio de la creación de nuevos tiquetes. Esta inflación debería de ser rechaza debido que viola propiedades desebles de aislamiento y carga pero aún asi, esta puede ser muy útil entre clientes mutuamente confiables: inflación y deflación puede usarse para ajustar la asignación de recursos sin comunicación explícita. *Ticket Currencies:* una moneda (*currency*) única es utilizada para denominar tiquetes dentro de cada límite de confianza. Cada moneda está respaldada o "financiada" (*funded*) por tiquetes que se denominan en monedas más primitivas. Los efectos de la inflación pueden ser contenidos de forma local al mantener un tipo de cambio(*exchange rate*) entre cada moneda y un una moneda *base* (*base currency*) la cual se conserva. La abstracción de moneda es

útil para brindar flexibilidad, dar nombre, compartir y proteger los permisos de los recursos. *Compensation tickets*: a un cliente que solo consume una fracción f de su *quantum* de recurso asignado se le puede otorgar un tiquete de compensación(*compensation ticket*) el cual infla su valor en $1/f$ hasta que el cliente inicie su próximo *quantum*. Esto asegura que el consumo de recursos de cada cliente, igual a f veces de su probabilidad p de ganar una lotería, sea ajustada en $1/f$ para igualar su parte asignada p. Sin tiquetes de compensación, un cliente que no consume su asignación completa de *quantum* recibiría menos que su parte otorgada de procesador.

Implementation. Se implementó un prototipo de *lottery scheduling* modificando un microkernel Mach 3.0. *Random Numbers:* se requiere una forma rápida de generar números aleatorios uniformemente distribuidos. La implementación usa un generador de número pseudo aleatorios basado en el algoritmo Park-Miller. *Lotteries:* la forma más simple para implementar *lottery ticket* centralizado es seleccionando aletatoriamente un tiquete ganador y luego buscar en una lista de clientes para localizar el poseedor del tiquete ganador. Esto requiere de generación de números aleatorios y $O(n)$ operaciones para recorrer una lista de clientes de tamaño n, acumulando la suma de boletos hasta que se alcance al valor ganador. Optimizaciones: (1) si la distrubución de tiquetes a los clientes es desigual, ordenar los clientes por conteo de tiquetes en forma decreciente puede reducir el tiempo promedio de la búsqueda. (2) Para n largos usar un árbol de sumas parciales de tiquetes. Los clientes son las hojas y para localizar el poseedor del tiquete ganador se tiene que recorrer desde la raíz hasta la hoja con el tiquete ganador. Requiere $O(\lg n)$. *Mach Kernel Interfase:* Una interface de *lottery scheduling* mínima es exportada por el microkernel. Consiste de operaciones para crear/destruir tiquetes y monedas, para *fund/unfund* una moneda y para calcular el valor actual de tiquetes y monedas en unidades base. La política de *lottery scheduling* coexiste con el tiempo compartido y políticas de prioridades fijas. *Ticket Currencies:* El prototipo usa un esquema simple para convertir las cantidades de los tiquetes en unidades base. Cada moneda mantiene la suma de la cantidad activa de todos los tiquetes que ha emitido. Un tiquete está activo mientras es usado por un hilo(*thread*) para competir en una lotería. Cuando el hilo es removido de la cola de ejecución, sus tiquetes son desactivados. Estos son reactivados cuando un hilo vuelve a unirse a la cola de ejecución. Si la desactivación de un tiquete cambia la cantidad activa de una moneda a cero, la desactivación se propaga a cada uno de sus tiquetes que respalda. De igual forma si la activación de un tiquete cambia la cantidad activa de la moneda de cero, la activación se propaga a cada uno de los tiquetes que respalda la moneda. El valor de una moneda se calcula al sumar tl valos de todos los tiquetes que respalda. El valor de un tiquete se calcula al multiplicar el valor de la moneda en la que el tiquete está denominado por su parte de cantidad activa emitida por la moneda. El valor de un tiquete denominado en la moneda base ese define como su *face value* (valor nominal). *Ticket Transfers:* El llamado al sistema `mach_msg` fue modificado para transferir temporalmente tiquetes de un cliente a un servidor por medio de llamados RPC[1] sincrónicos. Esto automáticamente redirecciona los permisos de un cliente bloqueado a un servidor el cual hace los cálculos en su nombre. *User Interfase:* monedas y tiquetes puede ser manipulados por medio de comandos. Crear y destruir tiquetes: `mktkt,rmtkt,mkcur,rmcur`, para *fund/unfund* monedas: `fund,unfund`, obtener información: `lstkt,lscur` y para ejecutar un comando con un financiamiento(*funding*) específico: `fundx`.

Experiments. Ver desarrollo en respuesta a pregunta 5 "¿Qué tan exitosa es esta solución?"

Managing Diverse Resources. En general, una lotería puede ser usada para asignar recursos en donde una cola de espera es necesaria para acceso a recursos. *Synchronization Resources:* Contención por sincronización puede afectar substancialmente las tasas de computación. *Lottery scheduling* puede ser usado para controlar los tiempos de espera relativos de hilos compitiendo por acceso a *locks*.. Se extendió la librería CThreads de Mach para soportar mutex de tipo *lottery-scheduled* además de la implementación de mutex estándar. Un *lottery-scheduled* mutex tiene asociado un *mutex currency* y un *inheritance ticket* emitido por una moneda. Todos los hilos que

[1]Remote Procedure Call

están bloqueados esperando a adquirir el mutex realizan transferencias de tiquetes para financiar (*fund*) el *mutex currency*. El mutex transfiere su *inheritance ticket* al hilo el cual posee el mutex actualmente. El efecto en la red de estas transferencias: un hilo que adquiere el mutex ejecuta su propio *funding* más el *funding* de todos los hilos en espera. Estos soluciona el problema de inversión de prioridad[2] Cuando un hilo libera un *lottery-scheduled* mutex, lleva a cabo una lotería entre los hilos en espera para determinar el próximo propietario del mutex. Entonces, el hilo se mueve el *inheritance ticket* mutex al ganador y cede el procesador. El próximo hilo a ejecutarse puede ser el *waiter* seleccionado o algún otro hilo que no necesite el mutex; la lotería normal de procesador elegirá de manera justa basado en su *funding*. *Space-Shared Resources:* Las loterías son útiles para asignar recursos tiempo compartido indivisibles. Una variante de *lotery scheduling* puede proveer eficientemente el mismo tipo de garantías en cuotas proporcionale probabilísticas para recursos de espacio compartido divisibles como la memoria. La idea básica is la de usar una lotería inversa(*inverse lottery*), en donde un "perdedor" es elegido para que renuncie a una unidad del recurso que posee. Realizar una lotería inversa es similar a poseer un tiquete de lotería normal excepto que probabilidades inversas son utilizadas. *Multiple resources:* dado que permisos para numerosos recursos se representan de forma uniforme por medio de tiquetes de lotería, los clientes puede usar comparaciones cuantitativas para hacer decisiones involucrando ventajas/desventajas entre diferentes recursos. Esto lanza algunas preguntas interesantes en torno a las políticas de *funding* de una aplicación en ambientes con múltiples recursos. Por ejemplo, cuándo tiene sentido cambiar el *funding* de un recurso a otro? Qué tan frecuentemente las asignaciones de *funding* deberían de ser reconsideradas? Una forma de abstraer la evaluación de las opciones de gestión de recursos es asociar un hilo administrador(*manager*) por separado a cada aplicación. El hilo administrador podría ser asignado a un pequeño porcentaje fijo (1 %) del total del *funding* de la aplicación, haciendo que se programe periódicamente mientras limita su consume de recursos. Para loterías inversas, puede ser apropiado permitir al cliente perdedor ejecutar un fragmento de código administrador pequeño con el fin de ajustar los niveles de *funding*. El sistema debería de proveer administradores para la mayoría de de aplicaciones. Las aplicaciones más sofisticadases pueden definir sus propias estratégias de de adminitración.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

Los *schedulers* convencionales no ofrecen un control de respuesta sobre las tasas de ejecución relativas de los cómputos que se considere eficiente y, esto es deseable en sistema que sirven solicitudes de variada importancia como bases de datos, aplicaciones multimedia y redes.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

La programación de cómputos (*scheduling computations*) en sistemas multi-hilo es un problema complejo y retador. Recursos escasos deben ser multiplexados para servir solicitudes de diversa importancia y, la política elegida para administrar esta multiplexación puede tener un impacto enorme en el *throughput* y los tiempos de respuesta. Un control preciso sobre la calidad del servicio provisto a los usuarios y aplicaciones requiere soporte para especificar tasas de cómputo relativas. Tal control es deseable en un amplio espectro de sistemas. Para cálculos de larga duración tal y como aplicaciones científicas y simulaciones, el consumo de recursos de computadora que es compartido entre los usuarios y aplicaciones debe de ser regulado. Para entornos interactivos como los que se podría encontrar en aplicaciones multimedia y de bases de datos, los programadores y los usuarios necesitan la habilidad de enfocar rápidamente los recursos disponibles en las tareas actuales que son importantes.

[2]En donde el propietario de un mutex con pocos fondos podría ejecutarse muy lentamente debeido a la competencia con otros hilos por el procesador, mientras un hilo con muchos fondos se mantiene bloqueado por el mutex.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

Algunos esquemas de propósito general se acercan en suministrar un control sensible sobre las tasas de los servicios que sea flexible. Aquellas que lo hacen generalmente se apoyan en la noción de prioridad la cual no provee la encapsulación y propiedades de modularización requeridas para sistema s de software grandes. Incluso, esquemas populares de priorización para asignación de CPU tal y como *decay-usage scheduling* son pobremente entendidos, a pesar de que se usan en numerosos sistemas operativos. *Fair share schedulers* existentes y *Microeconomic schedulers* solventan algunos de los problemas con esquemas de prioridad absoluta pero por otro lado el *overhead* asociado con estos sistemas los limita a tener un control relativo sobre cálculos de larga duración.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Un mecanismo de asignación de recursos aleatorios llamado *lottery scheduling*, el cual provee un control mas sensible sobre las tasas de ejecución relativas de los cómputos. *Lottery scheduling* implementa eficientemente una gestión de recursos de participación proporcional(*proportional-share*): la tasa de consumo de recursos de cómputos activos son proporcional a la participaciones relativas a las que están asignado. También soporta administración de recursos modular al habilitar a los módulos concurrentes a aislar sus políticas de asignación de recursos el uno del otro. *Lottery scheduling* también puede ser generalizada para administrar mucho recursos diversos, tal y como entrada/salida, memoria y acceso a *locks*.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

De acuerdo a la sección **5. Experiments:**

1. **Fairness:** medir la precisión en la cual el *lottery scheduler* podría controlar la ejecución relativa de cómputos. En cómputos de larga duración los resultados muestran que el *scheduler* puede controlar exitosamente las tasas de ejecución. En cómputos de corta duración hay alguna variación.
2. **Flexible Control** (Control dinámica de inflación de tiquetes): tres integraciones Monte-Carlo son iniciadas con dos minutos de separación. A cada tarea se le asigna una parte de tiempo que es proporcional al cuadrado de su error relativo. Cuando una nueva tarea inicia, recibe inicialmente un parte grande de procesador. Esta parte dismuye conforme la tarea reduce su error a un valor cercano al que tienen otras tareas en ejecución.
3. **Client-Server Computation:** servidores multi-hilo van a procesar solicitudes de diferentes clientes a tasas definidas por sus respectivas asinaciones de tiquetes. Tres clientes compitieron por servicio. Las proporciones en el *throughput* observado y el tiempo de respuesta coinciden estrechamente con su asignación.
4. **Multimedia Applications:** y otro tipo de aplicaciones se pueden beneficiar del *lottery scheduling* al habilitar más control a nivel del sistema operativo, eliminando la necesidad de aplicaciones mutuamente confiables.
5. **Load Insulation:** la abstracción de moneda(*currency*) puede ser usada para aislar de forma flexible grupos de usuarios, tareas e hilos. Se ejecutaron 5 tareas a las cuales se le realizaron fluctuaciones en las asignaciones de tiquetes y cada cambio fue contenido localmente sin causar efectos secundarios en las otras tareas.
6. **System Overhead:** el mecanismo "core" de *lottery-scheduling* es muy ligero: una lotería basada en un árbol que necesita generar solamente un número aleatorio y realizar $\lg n$ sumas y comparaciones para seleccionar el ganador entre n clientes. En general se encontró que el *overhead* impuesto por el prototipo de *lottery scheduler* es comparable con el que tiene la política de tiempo compartido de un Mach estándar.