

Tickets and Currencies Revisited: Extensions to Multi-Resource Lottery Scheduling

David G. Sullivan, Robert Haas, Margo I. Seltzer
Harvard University

Abstract

Lottery scheduling's ticket and currency abstractions provide a resource management framework that allows for both flexible allocation and insulation between groups of processes. We propose extensions to this framework that enable greater flexibility while preserving the ability to isolate groups of processes. In particular, we present a mechanism that allows processes to modify their own resource rights by exchanging resource-specific tickets with other processes. Ticket exchanges limit the effects of the changed allocations to the participants in the exchange, and they allow applications to coordinate with each other in ways that are mutually beneficial. Application-specific "negotiators" can be used to initiate exchanges based on an application's quality-of-service requirements and the current state of the system. We also propose flexible access controls for currencies through extensible "brokers" that solve such problems as the inability of users isolated by currencies to renice background jobs. Finally, we suggest using extensibility to allow users to install specialized allocation mechanisms for their processes. Together, these extensions enable an application-centered approach to resource management that is both secure and effective.

1 Introduction

Different applications have different resource requirements. For example, scientific calculations or simulations tend to be predominately processor-bound, while database management systems are typically I/O- or memory-intensive. Furthermore, an application's needs can change dynamically as it executes. Traditional resource allocation mechanisms are often too general to adequately meet these differing and dynamically changing needs. The resource management framework developed for lottery scheduling [10, 11, 12] provides a means for resource allocation to be more flexible and responsive. Clients receive a share of a given resource that is proportional to the number of "tickets" that they hold for that resource; changing the number of tickets that a client holds automatically leads to a change in its resource rights.

In addition, lottery scheduling provides a *currency* abstraction that allows processes to be grouped together and insulated from other processes. For example, the system might choose to fund each user's applications with tickets issued by a currency specific to that user. Each currency effectively maintains its own exchange rate with a central base currency: the more tickets a currency issues, the less they are worth with respect to the base currency, and their total base value can never exceed the value of the tickets used to back the currency itself. By employing equally-funded "user currencies," the system can ensure that each user receives the same overall rights to the system's resources for his or her processes, and that no single user can monopolize the system. Other, more sophisticated policies are also possible; some of them are discussed below.

We have developed several extensions to lottery scheduling's resource management framework¹ that increase its flexibility and enable an *application-centered* approach to operating system resource management. One of these extensions, *ticket exchanges*, allows processes to adjust their own resource rights by bartering with each other over tickets that are specific to particular resources. For example, a CPU-intensive application could exchange some of its disk tickets for some of the CPU tickets of an I/O-intensive application. These exchanges allow applications to safely circumvent the upper resource limits imposed by currencies, without sacrificing the protection that currencies provide. Application-specific *resource negotiators* can be used to initiate exchanges in response to an application's quality-of-service (QoS) requirements and the current levels of contention in the system, allowing applications to adapt their resource usage over time. Ticket exchanges and resource negotiators are discussed in greater detail in Section 3.

In order for the ticket/currency framework to be secure in a multi-user setting, some sort of access controls

1. Note that lottery scheduling's ticket/currency framework can accommodate scheduling mechanisms other than the probabilistic "lottery" algorithm that Waldspurger and Wehl proposed for scheduling the CPU. In particular, more reliable *deterministic* mechanisms can be used, as Waldspurger and Wehl's work itself attests [11, 12].

are needed. We thus propose associating an *extensible broker* object with each currency. Brokers allow a wide variety of policies to be implemented concerning how currencies can be manipulated by users, and they help to eliminate undesirable limitations that currencies would otherwise impose. Extensibility can also be used to further the goal of application-centered resource management by allowing users to install specialized allocation mechanisms on a per-currency basis. Brokers and other uses of extensibility are discussed in Section 4.

In the next section, we briefly motivate the use of the lottery scheduling framework and, in particular, the insulation between groups of processes that it enables. We then provide greater detail about our proposed extensions to the framework. Finally, we discuss related work and conclude with a summary of the status of our implementation.

2 Why lottery scheduling?

We have already suggested that the framework provided by lottery scheduling is valuable because it provides a means for flexible responsiveness to the needs of applications, as well as the ability to easily group and isolate users, processes, and threads. While the insulation provided by currencies may not be essential on a desktop PC or workstation, it could be extremely valuable in systems where multiple users compete for the resources provided by a central server or group of servers², as is often the case in academic computing environments. A system based on the thin-client computing model would also greatly benefit from the insulation of currencies. In general, central service providers can use currencies to guarantee that users who pay the same amount receive the same rights to a system's resources. For example, a currency-based system could be designed for use on Web servers that provide virtual hosting. Each site would have its own currency, and requests for files belonging to that site would be serviced by threads funded by its currency. Such a system would ensure that requests for any site hosted on the server are handled at a reasonable rate, even in the face of an overwhelming volume of requests for a particular site.

Lottery scheduling's ticket and currency abstractions also lend themselves to economic interpretations. For example, the tickets that fund a particular process can be viewed as assets belonging to that process. This interpretation leads naturally to the idea of ticket exchanges, which we now present in detail.

2. Indeed, SUN Microsystems has developed a system to provide insulation in such environments [9].

3 Ticket exchanges

While the lottery scheduling framework provides a means for flexible responsiveness to the needs of applications and for insulation between groups of processes, it is important to realize that these two goals are often at odds with each other. The protection provided by currencies necessarily restricts the adjustments that can be made to an application's resource allocations. For example, in a system that employs the user-currency scheme described above, users are free to modify the allocations of tickets issued by their own currencies, but that only affects the *relative* resource shares received by their applications. The largest absolute resource share that a user's application can receive in such a system is limited by the number of tickets that the operating system uses to fund the user's currency, something unprivileged users would ordinarily not be allowed to increase.

While the upper limits that currencies impose on resource rights are essential to maintaining insulation, there may be instances in which they are unnecessarily restrictive. Since certain resources may be more crucial than others to the performance of an application, the application may benefit from giving up a fraction of its rights to one resource in order to receive a larger share of another resource. Ticket exchanges provide a means for applications to initiate deals in which tickets for different resources are traded. They allow applications to take advantage of their differing resource needs while preserving the insulation of processes that do not participate in the exchange.

3.1 Exchanges preserve insulation

To see that the resource rights of non-participants are unaffected by ticket exchanges, consider the following scenario: processes A, B and C each start out with default allocations of 200 CPU tickets and 200 memory tickets, where the values are given with respect to the base currency. Process A is willing to give up CPU tickets worth 50 base value units in return for memory tickets with the same base value, and process B agrees to this exchange, as shown in the before and after picture in Figure 1. Since the total values of the tickets for each resource do not change as a result of the exchange, Process C, which does not take part, still has one-third of the tickets for each resource. Therefore, its resource rights are unchanged. The only applications whose resource rights are affected are those who voluntarily take part in the exchange.

Ticket exchanges and currencies complement each other. Ticket exchanges allow for flexibility in the face of the restrictions imposed by currencies, while currencies insulate processes from the malicious use of exchanges.

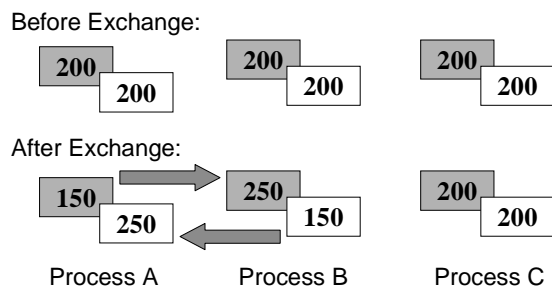


Figure 1: Ticket Exchanges Insulate Non-Participants. Processes A and B exchange tickets. Process C remains unaffected, since it still has one-third of the total of each ticket type.

For example, a process could fork off children that use exchanges to give the process all of their tickets. With currencies, however, this tactic would only affect tasks funded by the same currency as the malicious process.

3.2 Exchanges Enable Coordination

Ticket exchanges also enable applications to coordinate with each other in ways that are mutually beneficial and that may increase overall system efficiency. Various levels of sophistication could be employed by applications to determine what types of exchanges they are willing to make, and at what rates of exchange.

Certain types of clients may be sufficiently bounded by a given resource that they will always attempt to make an exchange that obtains more tickets for that resource. For example, consider two Web sites that are virtually hosted on the same server. Site A has a small number of frequently accessed files that it could keep in memory if it had additional memory tickets for its currency. Site B has a uniformly-accessed working set that is too large to fit in memory, and it would thus benefit from giving up some of its currency's memory tickets for some of A's disk tickets.

Processes could also apply economic and decision-theoretic models to determine, based on information about their performance (such as how often they are scheduled per second, how many page faults they incur per second, etc.) and the current state of the system (such as how many tickets of each type are active in the system), when to initiate an exchange and at what rate.

3.3 Resource Negotiators

An actual implementation of ticket exchanges could take a variety of forms. We are currently developing a prototype ticket exchange system in the VINO operating system that uses VINO's extensibility mechanism (grafting) [6, 7] to allow applications to safely download code for a

resource-negotiator graft that determines when to initiate an exchange. A kernel *dealer* process will periodically call this grafted function in each client that registers with the dealer (provided there is more than one such client), and the graft can at any time return a proposed exchange. A graft that has no need to make additional exchanges can inform the dealer of this fact by returning a special value.

When the dealer receives complementary exchange proposals from two or more clients, it will modify the ticket allocations of the processes accordingly. The dealer will also inform negotiator grafts of the results of their proposed exchanges the next time they are invoked. If the exchange could not be satisfied, these results will include any corresponding proposals with conflicting exchange rates (e.g., process A requests 20 CPU tickets in exchange for 10 memory tickets, while process B requests 10 memory tickets but only offers 10 CPU tickets in return). In this way, a negotiator graft can decide whether to modify its exchange rate and try again for a compromise deal. We still need to work out some of the details of this approach, such as the policy used by the dealer to handle rounds with multiple possible pairings of proposed exchanges. The complication here is that a given exchange may benefit one application more than another.

Using a graft that is periodically polled by the system allows applications to base requests for exchanges on the current state of the system. Application developers and users can thereby encode QoS requirements, and the graft will initiate exchanges whenever the requirements are not being met. Also, no alteration of program code is required except for the lines needed to install the negotiator, and these instructions can be executed by a separate driver program if the application's source code is not available.

4 Currency Brokers

Extensibility can be used to add additional flexibility to currencies and to enable a wider variety of resource allocation and scheduling policies. In particular, we plan to associate *extensible brokers* with each currency. A broker will control access to a currency, determining who can issue or revoke its tickets, fund or unfund it, or remove it entirely. Each currency will have an owner, group and UNIX-style mode, and the default broker methods will implement permission checks based solely on this information. However, users with the necessary privileges will be able to download grafted versions of broker methods to specify different access control policies.

4.1 Enabling the Semantics of Nice

To illustrate the value of extensible brokers, we first note that currencies not only enforce an upper limit on the

resources that a group of runnable processes can receive, but they also enforce a lower limit. To see this, consider a situation in which a user wants to run a non-essential job in the background. On a standard UNIX system, she could *renice* it, giving it a lower priority as a favor to other users of the system. In a system with user currencies, however, the only way to accomplish this would be to reduce the number of tickets used to fund the user's currency as a whole. Otherwise, no matter how few user-currency tickets a task holds, if it is the only runnable process in that currency, it will receive *all* of the user's share of the CPU. A user would presumably be allowed to reduce the number of tickets backing her currency, but if she later wanted to use a different application, it too would suffer.

To solve this problem, a grafted version of the base-currency broker's `may_issue()` method could allow users to issue base currency tickets, provided that the sum of the base currency tickets backing their currency and the base currency tickets they have already issued is less than some upper bound. In this way, users could reduce the number of tickets backing their currency by some small amount, and then issue that same amount from the base currency to fund a background job. The user's total resource rights would be unchanged (preserving the insulation of other users), and the background job would run at a reduced priority without crippling the user's other jobs.

4.2 Why Extensibility?

An extensible broker is not the only way to solve this particular problem; a special system call could be designed to shift funding from one currency to another. However, an extensible broker allows for additional, unforeseen problems to be addressed without modifying and recompiling the kernel. It also enables a wide variety of resource management policies to be implemented. For example, the base-currency broker could enforce strict limits during periods of normal usage, but permit greater flexibility during off-peak hours.

Extensibility also allows different policies to be used when assigning users to currencies. A graftable method in the system's currency table is called whenever a process changes its real user id; this method determines which currency should be used to fund the process, and creates and funds that currency if it doesn't already exist. This method is used to correctly fund the login shell of a user; thereafter, processes forked by that shell are funded by the same currency. Depending on the version of this method that is installed³, the system could employ one currency per user, one per group, or some combination of the two; in envi-

3. Note that this method would be *root-graftable*, which means that only superusers could replace it.

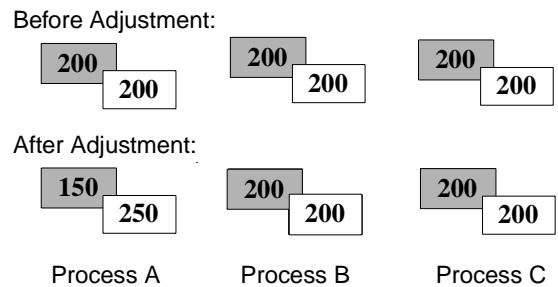


Figure 2: An Approach That Fails to Insulate. Process A shifts 50 of its tickets from one resource to another. Other processes now have a smaller percentage of the tickets devoted to the second (“white ticket”) resource, and thus a smaller share of that resource.

ronments where insulation is not critical, all processes could be funded by the base currency directly.

Finally, we plan to allow users with the necessary privileges to download specialized scheduling methods for a given currency. For example, our implementation of lottery CPU scheduling starts at the base currency and works its way down the currency tree, holding a new lottery at each level. If a user's currency wins the lottery at the top level, then a second lottery is held to pick one of the processes (or subcurrencies) funded by that currency. By grafting a currency's `get_winner()` method, a user can gain greater control over how the processes funded by that currency are scheduled.

5 Related Work

Waldspurger [11] recommended a different approach to resource tradeoffs, in which processes are given tickets that are not resource-specific and allowed to devote them to resources as they see fit. While such an approach gives processes greater flexibility, it violates the insulation properties of currencies by causing changes in the total number of tickets applied toward a given resource, as the example in Figure 2 demonstrates. Ticket exchanges, on the other hand, provide flexibility while preserving insulation. If insulation is less important in a given environment, then the system's dealer thread could be modified through grafting to carry out the exchanges that processes propose on the processes themselves (e.g., to take away 20 CPU tickets from a process and give it 20 extra memory tickets in return), thus giving an approach equivalent to the one suggested by Waldspurger.

Other systems have allowed applications to coordinate their use of system resources. The resource management system proposed for the Rialto OS [2] allows applications to negotiate for needed resources with a local “resource planner.” Applications renegotiate with the resource planner as their needs change, and make tradeoffs

among resources if their requested allocations cannot be met. Negotiator grafts in our proposed system will be able to perform the types of self-monitoring, reasoning about resources, and performance-tuning that Rialto requires of applications. Currency brokers and the system's dealer thread will serve some of the same functions as the Rialto resource planner.

In the Odyssey system for mobile computing [5], the system monitors changes in resource availability, notifies applications of relevant changes, and allows them to decide how best to adapt. Our proposed system is similar in spirit to their "application-aware" approach, leaving adaptation to applications while using the system to control and enforce resource allocations.

The system we have outlined will differ from existing ones in the ability it gives applications to coordinate their resource usage with *each other*, as well as with the system as a whole.

Much of the recent work on resource allocation and scheduling has been motivated by the need to accommodate soft real-time (i.e., multimedia) applications [1, 3, 4, 8]. Waldspurger and Weihl [12] have suggested an allocation policy that allows the lottery scheduling framework to support the resource reservations that such applications require. Extensible brokers could be used to implement this policy, and we hope to investigate whether the framework can adequately support such applications.

6 Conclusions

The lottery scheduling resource management framework [10, 11, 12] provides a means for both flexible responsiveness to the differing needs of applications and insulation between groups of processes. We believe that its ability to easily group and isolate users, processes, and threads makes it an excellent choice for systems in which multiple users are using the resources of central server, as in thin-client networks or Web servers used for virtual hosting. We have proposed extensions to the basic framework that increase its flexibility and enable an application-centered approach to resource management that is both secure and effective. Ticket exchanges allow processes to adjust their ticket allocations while insulating clients that do not take part in the exchange, and they provide a means for applications to coordinate their resource usage with each other. These exchanges can be initiated by application-specific negotiators that allow an application to monitor its resource usage and adapt as needed. Brokers associated with each currency provide extensible access controls, enabling currency-related policies to be both flexible and secure. Finally, extensibility can also be used to allow a variety of policies regarding how users are

assigned to currencies, and to install specialized resource allocation mechanisms on a per-currency basis.

We have implemented and tested a complete lottery scheduling framework in VINO, and the randomized lottery scheduling mechanism is used to schedule the CPU in the 0.50 release. We are in the process of implementing ticket-based allocation of memory and disk bandwidth, and of adding currency brokers and ticket exchanges to the system.

References

- [1] Bruno, J., Gabber, E., Ozden, B., Silberschatz, A., "The Eclipse Operating System: Providing Quality of Service via Reservation Domains," *Proceedings of the USENIX 1998 Annual Technical Conference*, pp. 235-246, June 1998.
- [2] Jones, M.B., Leach, P.J., Draves, R.P., Barrera, J.S. III, "Modular Real-Time Resource Management in the Rialto Operating System," *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, pp. 12-17, May 1995.
- [3] Jones, M.B., Rosu, D., Rosu, M-C., "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities," *Proceedings of the 16th ACM Symposium on Operating System Principles*, pp. 198-211, October 1997.
- [4] Nieh, J., Lam, M., "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," *Proceedings of the 16th ACM Symposium on Operating System Principles*, pp. 184-197, October 1997.
- [5] Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.R., "Agile Application-Aware Adaptation for Mobility," *Proceedings of the 16th ACM Symposium on Operating System Principles*, pp. 276-287, October 1997.
- [6] Seltzer, M., Endo, Y., Small, C., Smith, K., "Dealing with Disaster: Surviving Misbehaved Kernel Extensions," *Proceedings of the Second Symposium on Operating System Design and Implementation*, pp. 213-228, October 1996.
- [7] Small, C., "Building an Extensible Operating System," Ph.D. thesis, Harvard University, Division of Engineering and Applied Sciences, October 1998.
- [8] Steere, D.C., Goel, A., Gruenberg, J., McNamee, D., Pu, C., Walpole, J., "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling," *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pp. 145-158, February 1999.
- [9] "Solaris Resource Manager 1.0: Controlling System Resources Effectively: A White Paper," <http://www.sun.com/software/white-papers/wp-srm/>.
- [10] Waldspurger, C.A., Weihl, W., "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proceedings of the First Symposium on Operating System Design and Implementation*, pp. 1-11, November 1994.
- [11] Waldspurger, C.A., "Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management," Ph.D. thesis, MIT/LCS/TR-667, MIT Laboratory for Computer Science, September 1995.
- [12] Waldspurger, C.A., Weihl, W., "An Object-Oriented Framework for Modular Resource Management," *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pp. 138-143, October 1996.