Expressing and Enforcing
Distributed Resource Sharing Agreements

Tao Zhao and Vijay Karamcheti
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{*taozhao, vijayk*}*@cs.nyu.edu*

Abstract

Advances in computing and networking technology, and an explosion in information sources has resulted in a growing number of distributed systems getting constructed out of resources contributed by multiple sources. Use of such resources is typically governed by sharing agreements between owning principals, which limit both who can access a resource and in what quantity. Despite their increasing importance, existing resource management infrastructures offer only limited support for the expression and enforcement of sharing agreements, typically restricting themselves to identifying compatible resources. In this paper, we present a novel approach building on the concepts of tickets and currencies to express resource sharing agreements in an abstract, dynamic, and uniform fashion. We also formulate the allocation problem of enforcing these agreements as a linear-programming model, automatically factoring the transitive availability of resources via chained agreements. A case study modeling resource sharing among ISP-level web proxies shows the benefits of enforcing transitive agreements: worst-case waiting times of clients accessing these proxies improves by up to two orders of magnitude.

1 Introduction

With advances in computing and networking technology, more and more distributed computers are being interconnected to provide a large collection of computing and communication resources. There is an increasing desire to make this resource collection sharable by participating entities. Although current uses of such sharing focus on distributed systems built from components belonging to either a single or a small number of administrative domains (e.g., US supercomputer sites), the growth of the internet has resulted in newer models of resource sharing that span multiple domains. For example, an increasing number of companies find themselves having to share information in their databases with others. Companies with limited budgets may want to share hardware (e.g., network bandwidth and disk capacity) with others. The growing popularity of application-specific providers (ASPs) exemplifies a situation where the ASP is sharing its resources with several client organizations.

In each of the above scenarios, resources are owned by different organizations and span multiple administrative domains. Consequently, resource sharing is governed by *sharing agreements* between participating entities. For example, organization *A* and *B* have resource sharing agreements between them so that *A* can use 30% of *B*'s network bandwidth, and in return *B* can use 20% of the CPU power of *A*'s supercomputer. These agreements specify both the obligations and privileges of participants, and executions of applications should follow or be forced to follow those agreements. Moreover, agreements must be enforced in the presence of heterogeneous resource types and dynamically changing user set and resource availability. Also, sharing agreements can be transitive, i. e., one principal is able to share resources from another via a chain of sharing agreements even though the two principals do not have any direct sharing agreements. Unfortunately, most existing resource management infrastructures such as Condor [11], Globus [6], Legion [5], and others [2, 13, 15] provide very limited support for expressing and efficiently enforcing such sharing agreements. This support typically takes the form of "matching" up requests with compatible resource types and does not factor in any capacity restrictions implied by the agreements. The enforcement of capacity constraints is usually the responsibility of end points in those systems.

In this paper, we present a novel approach for expressing resource sharing agreements, which extends the con-

0-7803-9802-5/2000/$10.00 ©2000 IEEE.

cepts of *tickets* and *currencies* originally proposed in the context of uniprocessor operating system scheduling [14]. Our approach abstracts out resource heterogeneity by uniformly representing resource capacities in terms of tickets with differing values. Thus, diverse resources such as an Intel Pentium II CPU and an Intel Pentium III CPU can both be represented as a CPU ticket with different value. Agreements themselves are captured in terms of other kinds of tickets that are issued by currencies. The value of a currency fluctuates dynamically as a function of changing resource availability and affects the actual value of the tickets. This allows capturing of both absolute and relative sharing agreements. The scheme is complemented by a global resource allocation algorithm that uses linear programming techniques to enforce these agreements, automatically factoring the transitive availability of resources via chained agreements. When multiple scheduling choices are available, the algorithm chooses the one that perturbs global resource availability the least. By exposing resource sharing agreements to the resource scheduler, the scheduler is able to make more informed decisions that take both the resource availability and resource sharing agreements into account.

To assess the advantages of sharing agreements and the benefits from global allocation decisions, we describe a case study modeling resource sharing among ISP-level web proxies. Our results, based on trace-driven simulation of client request streams, show that sharing agreements result in higher performance at each proxy without requiring increased capacity investments, particularly when participants exhibit resource utilization profiles that are skewed in time (as might happen with geographically-distant proxies). Moreover, enforcing transitive agreements can result in performance improvements that compensate for the cost of redirecting requests to available resources: worst-case client waiting times improve by up to two orders of magnitude. Simulation results also show that with global information about resource availability and resource sharing agreements, a resource scheduler can make better scheduling decisions.

The techniques described in this paper are part of a flexible execution substrate for distributed applications that we are constructing in the Computing Communities project [3] at New York University. Other techniques (e.g., resource-constrained sandboxing [4]) being developed as part of this project focus on the complementary problem of security concerns, e. g., ensuring that a malicious application does not misuse a resource that it has been allocated.

In the rest of this paper, we describe in turn agreement expression (Section 2), agreement enforcement (Section 3), and the case study (Section 4). Section 5 discusses related work and we conclude in Section 6.

2 Expressing Sharing Agreements

A resource management system requires precise expression of three kinds of information: resource availability, resource requests, and agreements between users. Resources refer to both physical entities such as CPU and disk, as well as logical entities such as access rights. Both resource availability and resource requests are represented as vectors, with entries quantifying the quantity or need for each different kind of resource. Our approach expresses actual resource capacities and agreements between participants using a uniform framework, permitting the convenient computation of dynamic resource availability for each individual participant.

2.1 A Taxonomy of Agreements

Sharing agreements between the owner of a resource and its users define both which users can access the resource and any capacity constraints governing such access. For example, an owner A might specify that its resources are available for use by users B and C but not by user D. Moreover, A might enable B to use a larger fraction of the resource as compared to C.

Agreements can be either *granting* or *sharing* in nature, according to who can use the resources after the agreement is applied. The former refers to the case where the grantor gives up the resource to the grantee, i.e., the grantor cannot use the resource itself after granting unless it revokes the resource from the grantee (agreement ends). With sharing agreements, both the grantor and grantee have the rights to use the resource. In this paper, we restrict our attention to consumable resources such as CPU and disk bandwidth, which can only be used by one principal at a time.

Another dimension along which agreements can be classified is the expression of the quantity constraints on sharing agreements. Agreements can be either *absolute* or *relative*. In absolute agreements, the quantity is a constant (e. g. 10MB disk space), while in relative agreements, the quantity varies as a function of available resources (e. g. 50% of the available CPU). Relative agreements are useful for ensuring that a certain fraction of a principal's resources are always available for its own use, even when the available resources decrease a lot due to intensive usage.

2.2 Tickets and Currencies

The primary challenge with expressing sharing agreements is to capture the heterogeneous nature and dynamically changing availability of resources that impact both which resources are accessible using an agreement and in what quantity. We use the concepts of *tickets* and *currencies,* originally proposed in the context of a randomized uniprocessor scheduling mechanism called Lottery Scheduling [14], to address this challenge.

Tickets are abstract entities that differ in type and value. In our approach, tickets are used to encapsulate both access and capacity constraints governing resource usage. Possessing the right ticket type permits access to the resource and the ticket value determines the resource quantity that can be accessed. Tickets can be either *absolute* or *relative*. An absolute ticket captures the right to use an absolute quantity of resource (e.g., 10MB of disk), while the value of a relative ticket depends on the *issuing* currency. **Currencies** denominate tickets. Each currency is backed (or funded) by tickets and in turn issue its own tickets. Agreements between two entities A and B can be represented by A's currency issuing a ticket to support B's currency, which means B can use part of the resources of A (see Figure 1).

The value of an absolute ticket is its face value. And a relative ticket's real value is computed by multiplying the value of the currency from which it is issued by its share of all the amount issued by that currency. The value of a currency is determined by the summation of all the backing tickets (both absolute ones and relative ones).

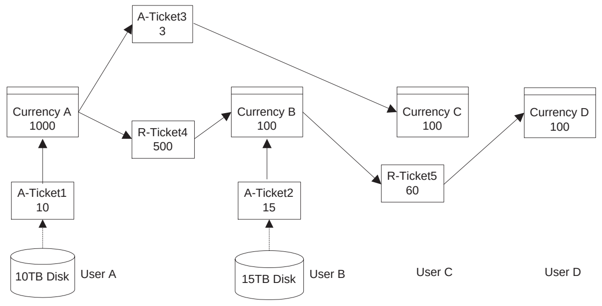

Figure 1. Use of tickets and currencies to express sharing agreements.

Example 1. Figure 1 shows an example of the use of tickets and currencies for expressing sharing agreements. This system has four principals, A, B, C, and D, and two disk resources of 10 TB and 15 TB capacity owned by principals A and B respectively. These resources are represented by two absolute tickets (A-Ticket1 and A-Ticket2), which fund currencies A and B, respectively. Principal A has an absolute agreement with principal C to share 3 TB of its resources, and a relative agreement with principal B to share 50% of its available resources. The former is captured by an absolute ticket (A-Ticket3) with value 3 and the latter by a relative ticket (R-Ticket4) with a face value of 500 issued by currency A, which has a face value of 1000. Thus, the real value of R-Ticket4 is $10 \times 500/1000 = 5$. This relative ticket boosts the value of currency B to $5+15=20$, increasing the amount of disk resources available to jobs submitted by principal B. Principal B in turn has a relative agreement with principal D, captured via relative ticket R-Ticket5 of face value 60 issued by currency B with a face value of 100. Note that the true value of this ticket is $20 \times 60/100 = 12$, which implicitly integrates the resources available to principal D via its direct agreement with B, as well as its transitive agreement with A (via B).

Thus, the actual resource capacities in a system are expressed using absolute tickets funding the owner's currency and agreements between participants take the form of absolute or relative tickets issued by one and backing the other's currency. The quantity constraint of this sharing agreement is the value of the ticket. Notice that the real value of relative tickets can change dynamically as more supporting tickets join the issuing currency or as some supporting tickets leave as a consequence of changing resource sharing agreements. Also we can inflate or deflate the currencies by increasing or decreasing the number of units in the currency, similar to inflation caused by the government printing more paper money. Also, new currencies can be created in addition to the default per-participant currencies: these permit a participant to decouple the quantity of resources transferred along some subset of its agreements from fluctuations in another subset.

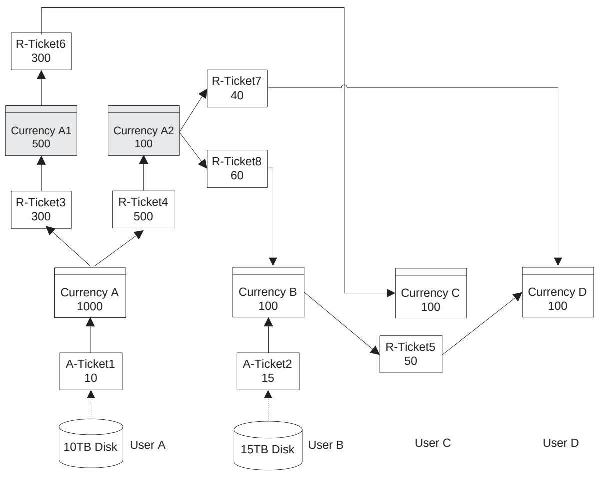

Figure 2. Use of virtual currencies.

Example 2. Figure 2 shows an example of the use of virtual currencies to isolate one subset of agreements from fluctuations of another subset. As in Example 1, the system has four principals, A, B, C, and D. In addition to four default currencies representing their resources, two new currencies are created from tickets issued by currency A: A_1 and A_2, shown shaded in the figure. We call these currencies *virtual currencies*, distinguished from default per-participant currencies. The value of virtual currencies is calculated the same way as normal currencies. So in Figure 2, virtual currency A_1 has the value of R-Ticket3, which is 3, and virtual currency A_2 has the value of R-Ticket4, which is 5. Both virtual currencies issue tickets, R-Ticket6 from A_1, and R-Ticket7 and R-Ticket8 from A_2. These tickets in turn fund currencies *C*, *D*, and *B* respectively, representing three agreements between *A* and each of *B*, *C*, and *D*. The agreements are divided into two subsets: one comprising B and D, and the other C. *A* can make change to one of the agreement subsets (e.g. by inflating the face value of currency A_1, or by issuing another ticket from A_1) without having any influence on the other subset. Therefore, by creating virtual currencies, *A* can effectively decouple the quantity of resources transferred along one subset of its agreements from fluctuations in another one.

As illustrated in the above two examples, using tickets and currencies permits us to express resource capacities and sharing agreements in an abstract, dynamic, and uniform fashion. Abstract, because tickets and currencies quantify resource rights independent from resource details. Dynamic, since the fraction of a resource that tickets represent varies dynamically in proportion to the currency that issues the tickets. And uniform, because tickets homogeneously represent rights for heterogeneous resources.

Although we have restricted our discussion here to the case where one resource is represented by a single ticket type, this mechanism can be extended to handle multiple views of the same resources by enabling resources backing multiple ticket types. This is useful in several situations. For example, the disk bandwidth resource can be viewed as two kinds of resources: read bandwidth and write bandwidth. This more complex case is a subject of future work and is not dealt with further in this paper.

Despite the generality afforded by our expression mechanism, in practice, we expect most sharing agreements to fall into one of the following structures:

Complete Each participant has sharing agreements with every other participant. This situation is most likely to happen when there are a small number of participants.

Sparse Every participant only has sharing agreements with a relatively small number other participants. This structure happens in situations where the number of participants is relative large.

Hierarchical In this structure, users are divided into groups. Inside a group, users have complete resource sharing. Between groups there are higher level sparse sharing agreements.

3 Enforcing Sharing Agreements

Most existing resource management infrastructures provide only limited support for enforcing agreements, typically providing a matching service that permits a request to use compatible and accessible resources. Enforcing constraints on resource capacity has typically been the responsibility of the end points. To enforce sharing agreements, the scheduler must keep track of dynamically changing resource availability via both direct and transitive agreements, and optimize a global criterion when faced with multiple choices for how to allocate resources.

We have constructed a linear equation model for the problem of allocating resources to a request in a distributed system in the presence of sharing agreements. This model works with the abstract notion of resources in the form of tickets and currencies described earlier. A request for specific quantities of one or more resources is viewed as a request for the set of ticket types representing these resources with ticket values capturing the desired quantities. Coupled resources, which must be allocated as a group, can be bound together and viewed as a new type of resource. The linear equation model permits the scheduler to easily determine (a) whether the principal submitting the job has sufficient resources available to it either directly or transitively (i.e., sufficient-valued tickets of the requisite type), and (b) in the event of there being multiple options, which actual resources to allocate the needed capacity from.

3.1 Basic Model

We first describe a simple form of this model, which only considers a single resource type and only *relative sharing* agreements. Suppose there are *n* users, each of whom own a resource with actual capacity $V_i \geq 0$. The matrix *S* captures all of the agreements between principals; S_{ij} indicates the value of the relative ticket issued by currency *i* and backing currency *j*. For example, $S_{ij} = 0.3$ signifies that user *i* agrees to share 30% of its available resources with user *j*. Three constraints on the elements of matrix *S* are: $S_{ii} = 0$, $S_{ij} \geq 0$ and $\sum_{1\leq k\leq n} S_{ik} \leq 1$.

To determine whether or not a user has sufficient resources available to satisfy its job request, we first calculate the resource flow from one currency node to another. Let $I_{ij}^{(m)}$ denote the resource amount that issues from currency node *i* and backs currency node *j* through at most *m* levels of transitive agreements. We have (see Figure 3):

$$I_{ij}^{(m)} = I_{ij}^{(m-1)} + V_i \times \sum_{\substack{1 \leq k_1, \cdots, k_{m-1} \leq n \\ k_p \neq k_q (\forall p \neq q) \\ k_p \neq i, j (\forall p)}} S_{ik_1} S_{k_1 k_2} \cdots S_{k_{m-1} j}$$

The constraints under the summation ensure that there is no cycle along the transitive agreement path.

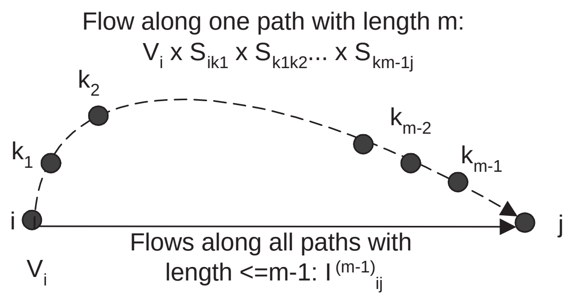
Flow along one path with length m:
$V_i \times S_{ik1} \times S_{k1k2} \ldots \times S_{km-1j}$
k_2
k_1
k_{m-2}
k_{m-1}
i
j
V_i
Flows along all paths with length <=m-1: $I^{(m-1)}_{ij}$

Figure 3. Flow of resources between node *i* and node *j* via a transitive chain of agreements involving intermediate nodes $k_1, k_2, \ldots, k_{m-1}$.

Although the formula appears very involved, it can be rewritten as:

$$I_{ij}^{(m)} = V_i \times T_{ij}^{(m)}$$

where $T_{ij}^{(m)}$ is a constant which involves only terms of elements in agreement matrix S.

The transitive closure of agreements is given by $I_{ij}^{(n-1)}$ for all i, j. Thus, user i has access to a resource amount C_i where

$$C_i = V_i + \sum_{k \neq i} I_{ki}^{(n-1)}$$

Having determined that there are sufficient resources available to the user to satisfy its job request, the second scheduling problem is to determine which actual resources to obtain the required capacity from. In general this decision depends on several factors such as the cost of borrowing resources from a different site and concerns of fairness. Here, we restrict our attention to optimizing a global metric that captures the notion of perturbing future resource availability in the system the least. Intuitively, the goal here is to leave the system in a state where it has sufficient resources to satisfy future requests *independent* of which principal is making the request.

Adding additional constraints to the linear programming model helps solve this problem. Let V_i' be the amount of resources left with principal i after the current allocation has taken place, changing the resource availability for user i from C_i to C_i'. Our objective is to minimize $\theta = \max_{1 \leq i \leq n}(C_i - C_i')$. The additional constraints can be expressed as below, where $I_{ij}^{(n-1)'}$ refers to $I_{ij}^{(n-1)}$ after allocation, A refers to the principal making the current request, and x to the requested amount:

$$I_{ij}^{(n-1)'} = V_i' \times T_{ij}^{(n-1)} \quad (1)$$

$$C_i' = V_i' + \sum_{k \neq i} I_{ki}^{(n-1)'} \quad (2)$$

$$C_A' = C_A - x \quad (3)$$

$$0 \leq (V_i - V_i') \leq I_{iA}^{(n-1)} \quad (4)$$

$$\sum (V_i - V_i') = x \quad (5)$$

$$(C_i - \theta) \leq C_i' \leq C_i \quad (6)$$

The variable θ is the global metric we want to minimize. There are $n \times (n-1)$ variables $I_{ij}^{(n-1)'}$, n variables C_i', n variables V_i' and 1 variable θ, leading to a total of $(n^2 + n + 1)$ variables. Using linear programming [8], we can minimize θ and thereby obtain the allocation schedule. The complexity of the linear programming model can be reduced by exploiting additional structure in commonly encountered agreement graphs as described in the following section.

3.2 Extensions to the Basic Model

In the above model, we only consider a single resource type in a resource request. In practice, resource requests usually include multiple resource types at the same time. In this case, a request for k types of resources is in the form of a vector $< r_1, r_2, ..., r_k >$, in which r_i is the amount requested for resource type i. To schedule this request,we need to solve k linear systems, one for each resource requested, and allocate resources according to the results. One problem with this scheme is that it does not handle resources that must be allocated together. For example, CPU and memory resources need to be on the same machine and cannot be allocated separately. One way to solve the latter is to bind these types of resources into a new type of resource so that they are always allocated together.

In the basic model, we have a restriction on the agreement matrix: $\sum_{1 \leq k \leq n} S_{ik} \leq 1$. This restriction is to prevent a participant from sharing more resources with others than it possesses. For example, suppose A has 10 units of resources. If there were no such restriction, A could share 60% of its resources with B and 60% with C. If B shared 100% of its resource with C, C would have 6 units resources directly from A and another 6 via B, which exceeds what A has in total! But in some situations it is desirable to have this kind of "overdraft" and it does not conflict with the semantics of *sharing*. In order to lift the restriction, we need to modify our calculations slightly, namely, change the equation:

$$I_{ij}^{(m)} = V_i \times T_{ij}^{(m)}$$

to

$$I_{ij}^{(m)} = V_i \times K_{ij}^{(m)}$$

where

$$K_{ij}^{(m)} = \begin{cases} T_{ij}^{(m)} & : \quad T_{ij}^{(m)} < 1 \\ 1 & : \quad T_{ij}^{(m)} \geq 1 \end{cases}$$

Therefore, in the above example the quantity of resources C can obtain is limited to 10 instead of 12.

The basic model considers only relative sharing agreements. To take absolute sharing agreements into account, we need an absolute agreement matrix A in addition to the relative agreement matrix S. A_{ij} indicates the value of the absolute ticket issued by currency i and backing currency j. We extend formulas in the basic model to deal with absolute tickets in the system as follows.

Let

$$U_{ki} = \begin{cases} I_{ki}^{(n-1)} + A_{ki} & : \quad I_{ki}^{(n-1)} + A_{ki} < V_k \\ V_k & : \quad I_{ki}^{(n-1)} + A_{ki} \geq V_k \end{cases}$$

The resource capacity of user i then becomes

$$C_i = V_i + \sum_{k \neq i} U_{ki}$$

In practice, the complexity of the linear programming model can be reduced by exploiting additional structure in the agreement graph. For example, one can use faster algorithms to deal with sparse matrices when the agreement structure is sparse. In the case of a hierarchical agreement structure, we can use techniques motivated by multi-grid refinement: once a request comes to a group, and that group cannot satisfy the request, we use LP to find the distribution of resources among groups; based on the distribution result, we run LP inside each group to further refine the resource allocation, iterating this process as required.

We are in the progress of implementing this model in a cluster-level resource management system built on top of Globus, which would allows us to further validate our model. The resource management system has two components: a centralized global resource manager (GRM) and multiple local resource managers (LRM). The GRM provides services to manage sharing agreements and to schedule resources among local resource managers. LRMs are responsible for providing resource availability information to the GRM dynamically, and fulfilling resource allocation according to the GRM's decisions. The architecture also permits splitting of the GRMs into multiple levels, each responsible for a subset of the LRMs.

4 Case Study: Resource Sharing among ISP-level Web Proxies

To assess the advantages of resource sharing under agreements and study the effects of various parameters such as the density of agreements and the impact of transitive agreements, we report on results obtained using a trace-based web proxy simulator that simulates the workload of a group of ISP-level web proxies that cooperate using resource sharing agreements. This simulator models an infrastructure for shared access to web-based services that is being currently developed by the authors.

4.1 Simulation Model

The simulation model is depicted in Figure 4. Each proxy serves web content to its local customers. A request from a customer consumes resources such as CPU, disk, memory and network bandwidth of the proxy server. To improve their service (response time is the main concern in this case) to customers, ISP-level resource sharing agreements permit proxy servers to share each others resources. Specifically, if the local resources of a proxy are not enough to satisfy all the customers' requests in time, the proxy can redirect some of these requests to other proxy servers.

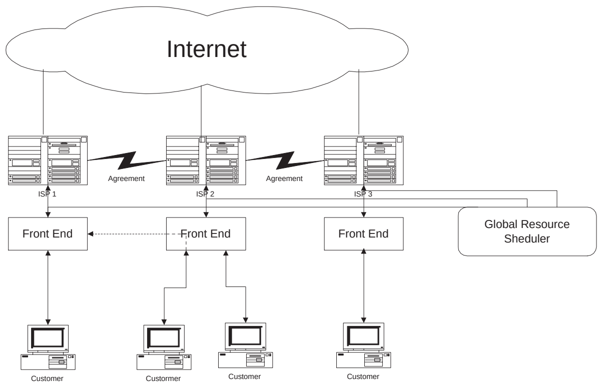

Figure 4. A group of cooperating web proxy servers with resource sharing agreements

.

A global resource scheduler stores information about the sharing agreements and keeps track of available resources at each proxy server using the tickets and currencies approach described in Section 2. For simplicity, all proxy server resources are collapsed together into a single "general" resource, that determines the processing power of the server. A request arrives at the server front-end and asks for a specific amount of this resource. Given our focus on characterizing the qualitative advantages of sharing agreements, we assume a priori knowledge of per-request resource requirements using a simple linear model based on the response length: a request producing a response of length x requires server resources $a + bx$ (in the experiments reported here $a = 0.1$ seconds and $b = 10^{-6}$ seconds; also to avoid extremely long response lengths from causing spikes in the waiting time, we set the maximum resources needed per request to be $c = 30$ seconds). When the resource requirements of requests queued up at a proxy's front-end exceed a threshold, the global scheduler is consulted to decide how to redirect some of these requests to other proxies. The scheduler makes resource allocation decisions enforcing the sharing agreements by solving the linear system described in the previous section.

The input to the simulator is a per-proxy stream of HTTP requests, generated from the UC Berkeley Home IP Web Traces [1]. The trace consists of 18 days' worth of HTTP traces collected in November 1996 and comprises more than 9 million references. In the results reported here, we focus on a 24-hour period obtained by averaging the 18-day trace.

4.2 Simulation Results

The solid line in Figure 5 shows the average number of requests during each 10-minute slot of a 24-hour period. We can see that the proxy load is heaviest around midnight and lightest around the early morning hours. The dotted line in Figure 5 shows the average waiting time of a request for the simulator parameters above; as expected the waiting time peaks at midnight with some requests seeing a wait time of as much as 250 seconds.

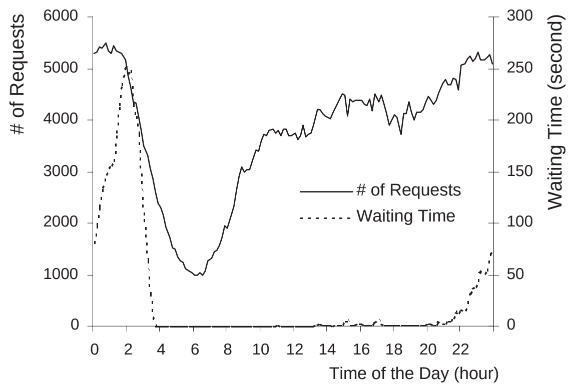

Figure 5. The number of requests and average waiting time per request without resource sharing.

Overall Benefits of Resource Sharing Because the request traffic fluctuates with time, if different ISPs are in different geographical regions so that they exhibit "rush hour" behavior at different times, busy servers can benefit from resource sharing by borrowing resources from idle servers. Figure 6 shows the impact of resource sharing agreements between a group of 10 ISPs on the average waiting time of a client request at a particular ISP, parameterized for different amounts of time skew between the client request streams. The agreement structure is a complete graph where each server shares 10% of its resources with every other server. Figure 6 shows that with a gap of 3600 seconds, the average waiting time drops dramatically from 250 seconds to below 2 seconds, demonstrating the ability of sharing agreements to spread out peak loads.

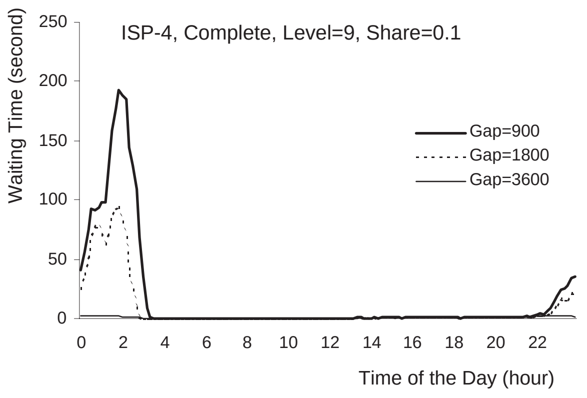

Figure 6. Average waiting time per request with resource sharing for different "gap" times (skew) between the proxy servers. The agreement structure is a complete graph between 10 servers: each server shares 10% of its resources with every other server.

Figure 7 compares this performance with the average waiting time obtained when sharing is disabled, but the proxy server has more processing power (corresponding to an increased capacity investment). We can see that 25%-35% more resources are required to match the performance obtained by resource sharing.

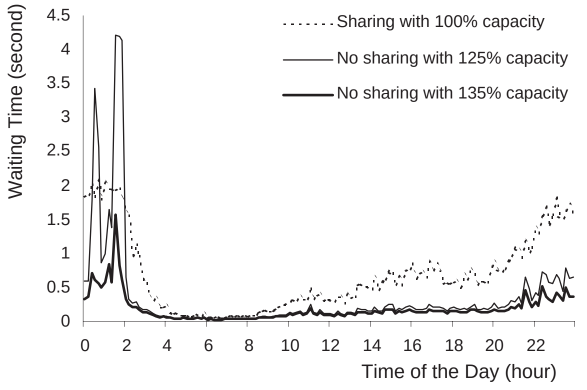

Figure 7. Average waiting times with and without resource sharing for different processing capabilities of the proxy server.

Benefits from Transitive Agreements The formulation in Section 3 automatically factors in transitive agreements. To assess the additional performance benefits that can result from doing so, we compare the average waiting times obtained for different levels of transitivity in the context of two agreement structures, a complete graph and a cyclic loop. In the latter, each server only has a sharing agreement with one other server. Considering only a single level of transitivity is equivalent to enforcing only direct agreements.

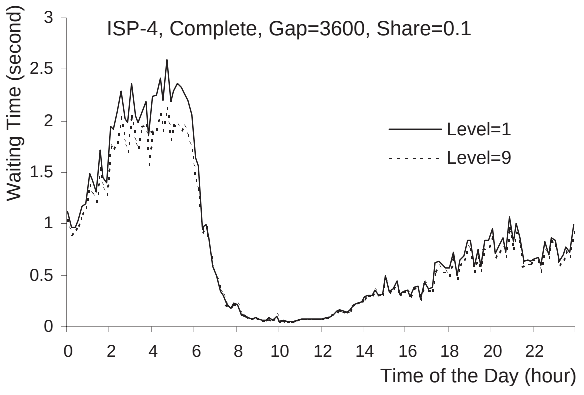

Figure 8. Average waiting time with resource sharing for different levels of transitivity of indirect agreements for a complete graph agreement structure between 10 ISPs where each ISP shares 10% of its resource with every other ISP.

Figure 8 shows that in the complete graph case, resource sharing helps but the incremental improvement by considering indirect transitive agreements is small. This is explained by the fact that all of the servers are already reachable from the requesting server using direct agreements. To further investigate the benefits of transitive agreements, we ran the simulator on three different loop based agreement structures. Figures 9, 10 and 11 show the results of these simulations.

Although all three agreement structures are loops in which each ISP has sharing agreement with one neighbor ISP, the time zone gap (skip) between two neighbors are different in these three configurations. As we can see from the figures, this difference is reflected in the simulation results. The worst-case waiting time when considering only direct agreements (level=1) is 35 seconds in Figure 9 (skip=1), and dropped to 7 seconds in Figure 10 (skip=3) and further to about 3 seconds in Figure 11 (skip=7). This is because of the skip value of the loop structure. By enforcing only direct agreements, each ISP can only use resources from its neighbor. In Figure 9, the neighbor is only one-hour time zone away for each ISP, it is quite likely that once one of them is busy another is too. Thus one ISP may not get much help from its neighbor. By pushing direct neighbor further as in Figure 10 and Figure 11, we can get more resource sharing benefits.

When three or more levels of transitive agreements are considered, the worst-case waiting time drops to about 2 seconds in all three configurations. This shows that factoring in transitive agreements yields robust performance. Note that considering longer chains of agreements yields small incremental benefit because of an exponential decrease in the amount of resources accessible along the chain.

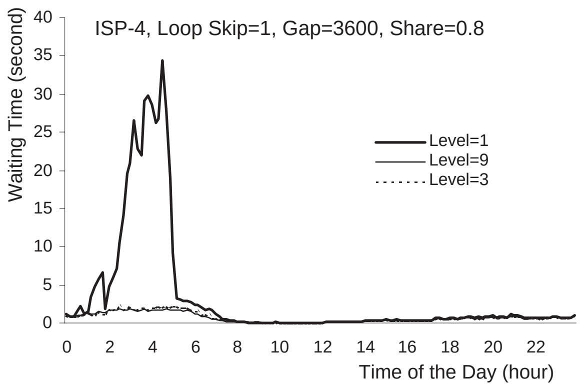

Figure 9. Average waiting time with resource sharing for different levels of transitivity of indirect agreements where the agreement structure is a loop with each ISP sharing 80% of its resources with the next one. The sharing neighbors are one-hour time zone away.

Effect of Redirection Cost The previous results assumed that the cost of redirection was negligible. Here we consider the impact on waiting time when each redirected request must incur a fixed overhead that is either 0.1 seconds or 0.2 seconds. These costs are approximately the same as or double the average processing time. Figure 12 shows that in the complete agreement graph case, the added cost has negligible impact on the average waiting time. This is because only a small number of requests (less than 1.5%) are redirected. Even at peak time, this amount is less than 6%. Although the waiting time of these requests has significant penalty, this penalty has negligible impact on overall performance. Even for the particular request, the redirection pays off because without redirection this request would suffer much longer delay. Thus, the benefits of resource sharing can compensate for fairly large overheads in using remote resources.

Centralized v.s. End-point Enforcement We use a centralized resource manager and linear programming model to schedule resources in the belief that with global resource

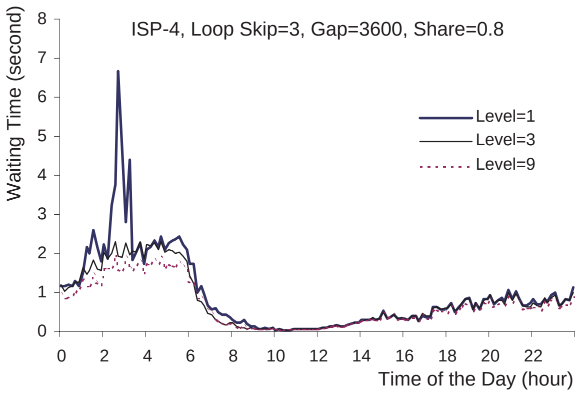

Figure 10. Average waiting times with resource sharing for different levels of transitivity of indirect agreements where the agreement structure is a loop and the sharing neighbors are three-hour time zone away.

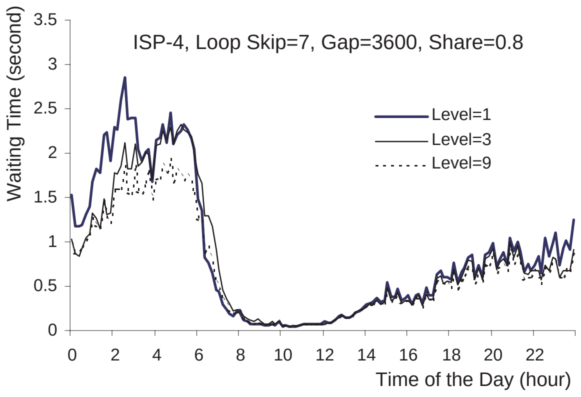

Figure 11. Average waiting times with resource sharing for different levels of transitivity of indirect agreements where the agreement structure is a loop and the sharing neighbors are seven-hour time zone away.

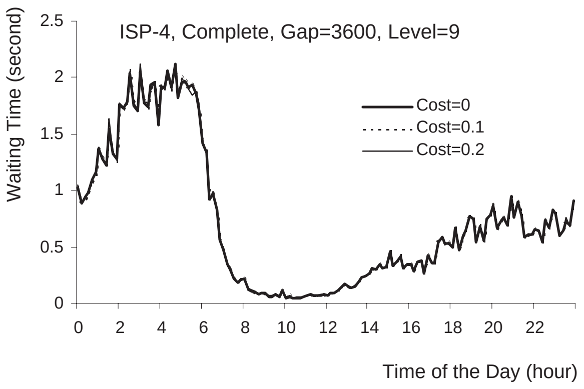

Figure 12. Average waiting time of a request as a function of redirection cost.

availability information and all transitive agreements information in hand, the centralized resource scheduler can make better decisions and improve the resource utilization. In this experiment, we compare the performance using our scheme with a scheme where agreement enforcement is performed at the end points.

The agreement structure is a complete graph where each ISP shares 20% of its resources with neighbors one-hour time zone away, 10% with neighbors two-hour time zone away, 5% with those three hours away and 3% with further neighbors. The basic scheme we used redistributes requests queued up at a proxy's front-end to all other ISPs. The number of requests redistributed is proportional to the quantity of sharing agreements with other ISPs. Therefore, when an ISP is busy, it tends to redirect more requests to nearby ISPs than faraway ISPs. Figure 13 shows a comparison of this scheme with the linear programming scheme discussed in Section 3.

As we can see from the figure, the linear programming scheme reduces the average waiting time by more than 50% at traffic peak time. This is because the non-linear scheme tends to redistribute requests to nearby ISPs no matter whether they are busy or not, while linear programming scheme takes both the resource availability status and sharing agreements into account when making decisions.

5 Related Work

Most large-scale resource management infrastructures in current use provide only limited support for expressing and enforcing resource sharing agreements.

Condor [11], a distributed batch processing system running on a cluster of of workstations, permits agreements between entities to be specified as part of the requirement

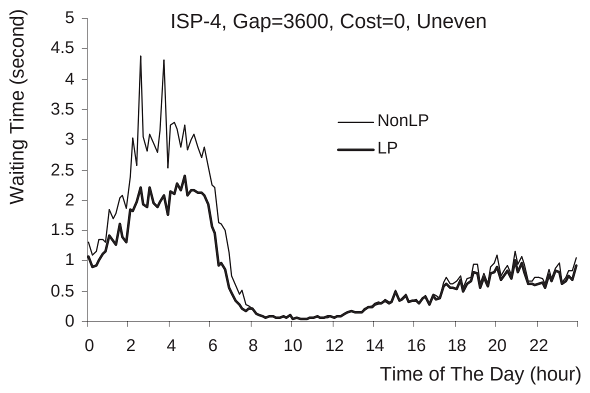

Figure 13. Average waiting time of a request using linear programming scheme and non-linear programming scheme, respectively.

expressions of *classads* from both the resource supplier and the resource consumer end. The Condor MatchMaker [12] module checks for compatibility but does not provide any mechanisms to specify the amount of resources a user is willing to share with others or to even change a posted classad. **Globus** [7] provides a toolkit for building execution environments where application execution can span multiple geographically distributed resources. The natural place for expressing agreements in Globus is in the context of the extensible *resource specification language* (RSL); however, in its current form, it is only possible to specify access constraint agreements, namely whether or not a particular resource is accessible to a job. **Legion** [9, 10], which provides an object-based model for orchestrating interactions between components in a distributed system, defines a collection query language similar to Condor and Globus that can be used to represent simple agreements, primarily expressing access constraints.

In each of the above systems, it is conceivable that agreement enforcement can be done at the end-points (based upon returned information); however, doing so restricts the scalability of the scheme and cannot take advantage of transitive agreements. Our approach differs in providing the resource management system with an abstract, dynamic, and uniform view of both resources and diverse sharing agreements enabling the design of policies that can enforce these agreements while optimizing global criteria.

There are also several systems that use models derived from economic theory to manage sharing among distributed resources. Spawn [13] uses auctions to allocate resources among clients that bid monetary funds. Funds encapsulate resource rights and serve as a form of priority. In MarketNet [15], access to system resources is governed by a market economy. Prices reflect a balance between supply and demand for a resource while the budgets assigned to client processes determine their priority in gaining QoS access to resources. A notable difference of our approach is that unlike the competitive environment assumed by these systems, we assume a cooperative environment where global benefit is the common goal. In addition, a shortcoming of economic-model based resource allocation is the difficulty of achieving competitive equilibrium relative to the dynamic perturbations of supply and demand.

6 Conclusions and Future Work

We have described a novel approach, based on the notions of *ticket* and *currency*, to express and enforce resource sharing agreements in a distributed system. It provides an abstract, dynamic and uniform way to represent heterogeneous resources. We formulated the resource allocation problem with agreements constraints using linear programming model to optimize global metrics. Results based on simulation show that resource sharing under agreements has advantages and the structure of agreements has significant effects on the benefits. Future work involves implementing and evaluating this scheme in the context of resource management system for the Computing Community (CC) Project [3], which provides a distributed, collaborative environment in which a dynamic changing set of partners work together to share resources.

Acknowledgments

We would like to thank Zvi Kedem, who frequently participated in and provided many helpful insights during discussions. This research was sponsored by the Defense Advanced Research Projects Agency under agreement numbers F30602-96-1-0320, F30602-99-1-0157, and N66001-00-1-8920; by the National Science Foundation under CAREER award number CCR-9876128; and Microsoft. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, SPAWAR SYSCEN, or the U.S. Government.

References

[1] http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html.

[2] A. Baratloo, Z. M. Kedem A. Itzkovitz, and Y. Zhao. Mechanism for just-in-time allocation of resources to

adaptive parallel programs. In *Proceedings of International Parallel Processing Symposium (IPPS/SPDP)*, 1999.

[3] A. Baratloo, P. Dasgupta, V. Karamcheti, and Z. M. Kedem. Metacomputing with milan. In *Proceedings of International Parallel Processing Symposium (IPPS/SPDP)*, 1999.

[4] F. Chang, A. Itzkovitz, and V. Karamcheti. User-level resource-constrained sandboxing. In *Proceedings of 4th USENIX Windows Systems Symposium*, 2000.

[5] Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew S. Grimshaw. Resource management in legion. Technical Report CS-98-09, Department of Computer Science, University of Virginia, February 11 1998.

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[7] I. Foster and C. Kesselman. The Globus project: A status report. In *Proceedings of the Heterogeneous Computing Workshop*, pages 4–18. IEEE Computer Society Press, 1998.

[8] S. I. Gass. *Linear Programming: Methods and Applications*. International Thomson Publishing, 5th edition, 1985.

[9] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Legion: An operating system for wide-area computing. Technical Report CS-99-12, Department of Computer Science, University of Virginia, March 23 1999.

[10] Michael J. Lewis and Andrew Grimshaw. The core legion object model. Technical Report CS-95-35, Department of Computer Science, University of Virginia, August 1995.

[11] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor : A hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111, Washington, D.C., June 1988. IEEE Computer Society Press.

[12] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.

[13] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. In *IEEE Transactions on Software Engineering*, 1992.

[14] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating System Design and Implementation*, 1994.

[15] Y. Yemini, A. Dailianas, and D. Florissi. Marketnet: A market-based architecture for survivable large-scale information systems. In *Proceedings of Fourth ISSAT International Conference on Reliability and Quality in Design*, 1998.