Resource Management for Extensible Internet Servers

Grzegorz Czajkowski, Chi-Chao Chang, Chris Hawblitzel, Deyu Hu, Thorsten von Eicken
Department of Computer Science, Cornell University
Ithaca, NY 14850, USA
{grzes,chichao,hawblitz,hu,tve}@cs.cornell.edu

1 Introduction

With the continued spread of the Internet the typical computing model for servers is undergoing a drastic change. In the past, server systems have moved from providing interactive time-sharing service to providing fileserver and now more general back-office (mail, database, web, etc.) services. While the characteristics of the new Internet server systems are not yet clear, we expect that Internet servers will have at least three characteristics that distinguish them drastically from today's servers: (i) high code mobility, (ii) large numbers of anonymous users, and (iii) significant concern for the efficient use of resources.

Due to the large number of users in popular Internet sites, the sophistication of the services that can be offered is primarily limited by the available computing resources and by security concerns. In this paper, we focus on the resource management issues and describe our approach to providing extensibility in server systems that must support mobile, untrusted code and large numbers of users.

The primary goal of the resource management we propose is to take advantage of resource consumption tradeoffs. In traditional server systems, the administrator decides which services run on which machines. Databases, for example, use three-tier architectures to allow request validation, database processing, and data formatting to occur on different server machines. The administrator also decides how to split computation between the servers and the client machines. The lure of the Internet combined with portable safe language technology (such as Java) is that these tradeoffs can be made much more dynamically and can be customized more easily. Basically, this is possible because it is feasible to move code around on demand.

In order to take advantage of these tradeoffs, however, the programs need information on their current resource consumption, indications on the available resources in the systems, as well as incentives to use the resources leading to the highest overall throughput. We claim that in many situations the response of a service to resource shortage does not have to be a degradation of quality. Instead, it is often possible to trade one resource for another one and continue to provide the same quality of service.

In this paper, we focus on a Java-based system, the J-Kernel [2], which supports the execution of untrusted code, called *servlets*[1], in servers. The servlets can be stored on the server itself or they can be uploaded dynamically by users in order to customize the server functionality. We discuss how we extend the J-Kernel to provide resource usage feedback as well as incentives for servlets to adapt their behavior in response to load fluctuations.

We also claim that short-term guarantees are essential in enabling resource usage tradeoffs: if a servlet decides to change its behavior in order to take advantage of an under-utilized resource, the system should be able to guarantee the actual availability of the resource. This is in contrast to long-term guarantees which seem to lead to high costs because they inevitably require overly conservative reservations. Finally, we argue that economics-based models are not really applicable in dynamically extensible Web servers: while these models are currently fashionable, the actual implementations remain unconvincing and do not appear to map onto real user behaviors.

[1] The first servlet architecture and implementation was proposed in the Java Web Server []. The J-Kernel differs significantly from the Java Web Server in its emphasis on protection and efficiency.

2 Advantages of being resource-aware

Resource-aware servlets can utilize information about resource availability to decide what resources and in what amount it is best to consume. This provides incentives for servlet writers to look for appropriate runtime adjustments or applicable resource tradeoffs. Different tradeoffs are possible. A simple example is a servlet sending documents to its client. Depending on CPU and network bandwidth utilization, the servlet may use a compressed data format, trading off CPU time for network bandwidth. Another example is the representation of certain data structures, like sparse matrices and hashtables, where well-known tradeoffs exist between storage space and element access time.

More complex examples exist in various domains of computing. Database query processing systems track memory availability and, based on it, invoke different implementations of query operations. For instance, some commercially available databases choose between three implementations of table joins, each of them providing different memory vs. CPU usage tradeoffs. An important part of the system in this case is the feedback mechanism that drives the tradeoff. These databases provide the feedback necessary for making a near-optimal run-time choice of the appropriate algorithm.

Caching is another instance of a possible resource tradeoff, where various resources are traded for memory. For instance, network bandwidth is saved at the expense of increased memory requirements by HTML documents caching in browsers; time is traded for memory in information retrieval systems, where answers to the most recent queries are cached.

3 The target environment

The environment we propose is based on the architecture of the J-Kernel [3]. The J-Kernel is a μ-kernel that executes on a single Java Virtual Machine (JVM) and provides multiple protection domains in that JVM. The J-Kernel is written entirely in Java and provides a new class loader which allows servlets to be uploaded and executed each in its own protection domain. Protection in the J-Kernel is based on software capabilities. There is no hardware protection available inside the J-Kernel to protect two potentially malicious domains from each other or to protect the J-Kernel from the domains. Each protection domain (i.e. a servlet) has a namespace that it controls as well as a set of threads. Cross-domain calls are performed by invoking methods of capabilities obtained from other domains. The J-Kernel's class loader interposes a special calling convention for these calls as follows. Arguments and return values are passed by reference if they are also capabilities, but they are passed by copy if they are primitive types or non-capability objects (deep copy in this case). The effect is that only capabilities can be shared between protection domains and references to regular objects are confined to single domains.

This system was carefully designed to facilitate resource management. An important decision made is to disallow object sharing. The rationale is that sharing objects arbitrarily makes decision resource accounting problematic (who should be accountable for an object shared by many servlets?) and resource reclamation and domain termination become difficult at best.

The server should support large numbers of simultaneously executing servlets. In such an environment, the task behavior and resource consumption assumptions valid for traditional workloads do not hold any more. First, even though the execution of servlets will be triggered by independent users, it is likely that very large numbers of servlets will be identical, be it a data mining agent, a set of image manipulating methods, or a custom program that handles electronic submission of documents. This is caused mostly by the fact that very few users in the real world actually write programs themselves - from operating systems to text editors to databases, users rely on code delivered by a relatively small number of software vendors for any given applications niche.

Another difference is the relative need for different resources, when servlets are compared to most applications executed in traditional environments. We expect that servlets will require relatively large quantities of network bandwidth. Higher network requirements come from the nature of the proposed system - servlets communicate with remote clients. Memory needs will most likely be no different than these for “traditional” applications, while CPU time consumption of servlets may be larger since the quality of (just-in-time-)compiled Java code is still rather unsatisfactory.

4 Resource management mechanisms

The implementation of resource management in the J-Kernel, running on Microsoft's JVM, is just being completed. Virtual memory, CPU and network usage are accounted for. Accounting for memory allocation and object lifetime (in terms of garbage collection periods survived) is accomplished via automatic bytecode rewriting: non-array objects have their finalizers modified, while special counting objects are associated with arrays. Per-thread CPU consumption is monitored by an NT kernel-level driver which can be queried from the J-Kernel. Network usage is monitored by an instrumented WinSock2 SPI DLL.

To ensure that fair, high throughput can be achieved and sustained in the environment described above, a comprehensive approach to resource management is needed. Expected large numbers of anonymous users, typically executing under flat rate billing (if there is any billing at all) make us focus on fair share scheduling. Each servlet has a guaranteed equal share of all available resources over the next scheduling period (i.e. over a period during which every servlet gets scheduled once), which prevents denial-of-resource attacks. As with fair queuing, servlets can use resources not used up by the others.

Our approach addresses the issue of how to utilize of these unused resources with the most benefit for the large population of servlets. In particular, we advocate providing servlets with resource availability feedback, based on resource usage accounting and providing means by which servlets can require and obtain various short-term resource- and performance-related guarantees. The first postulate means that every servlet should be able to query the environment for the following information: how much resources it is using and how much guaranteed resources, beyond its fair share, it could obtain in the next scheduling period. The second item allows servlets to plan ahead and decide, which of possible resource tradeoffs (if any) should be taken advantage of. Combined, resource information feedback and guarantees provide both an enabling mechanism and an incentive for utilizing these tradeoffs.

An important role in enabling resource-aware applications is played by providing resource-sensitive standard libraries. In such libraries different implementations of the same functionality exist, each of them with different resource consumption and execution times guarantees. For instance, popular core Java classes implementing hashtables and vectors of objects can be implemented either with fast access time but low memory requirements or with reverse properties. Similarly, two different versions of sockets may exist: a standard one and one which transmits and receives compressed data.

5 Short-time resource and performance guarantees

Utilizing resource tradeoffs goes hand in hand with guarantees that the server can give to servlets. Changing behavior from CPU-bound to network-bound, without a guarantee that at least for a short time this will actually pay off is risky at best. Due to the very dynamic nature of the whole environment, the only long-term guarantee that a servlet can obtain is a guarantee of fair share. However, since the number of servlets changes, even this weak guarantee cannot be translated into absolute quantities. The only absolute guarantee a servlet may count on is the short-term behavior of the system, during the next scheduling period.

Long term guarantees are both impractical in such a system due to expected high servlet traffic and expensive, since they typically tie up more resources than actually needed for long periods of time. Resource-aware applications can avoid over-reserving resources for long periods of time. A way to do this is to request small amount of resources, just necessary for providing basic quality of service. At runtime, exploiting short-term variations in supply and demand for various resources can lead to much better performance than the baseline. The system should provide the necessary information so that servlets can adjust their behavior to take advantage of possible resource tradeoffs, as described below.

6 A critique of economics-based approach to resource management

A number of research projects advocate the use of economical models to force applications to compete for resources [4]. This approach may seem ideal for Internet systems. In fact, there is an unquestionable merit to bidding for resources – prioritizing applications according to the amount of funding they have, which should in turn reflect the relative importance of different tasks. However, there are several problems with this approach. First, the actual pricing mechanism implemented in existing and proposed systems seems to invariably be either cheap to implement but rather arbitrary [3] or more realistic auction-based but with

high-overheads. For example, Edell et al. [1] propose a system that enables billing users for their TCP traffic but it increases the average connection setup time by 66%. A second issue is who actually "funds" applications in economy-based resource management environments. If it is the system, then the accumulation of "money" over periods of inactivity must be dealt with. Some systems propose a form of taxation [3], but the formulas used are again rather arbitrary. If the user funds the applications based on real money the pricing and its evolution must be easy to understand. Most current services appear to quickly converge on flat rates. However, introducing flat rates actually defeats the purpose of economics-based models. If the user funds the applications through "funny money" it may just not be taken seriously, since the user will simply not feel the penalty for misspending or misbehaving.

The problems outlined above made us resign from adopting a system-wide economic approach. Resources used by applications are accounted for, but we leave billing up to the service providers themselves. Applications are given information about their own resource consumption and can change their behavior accordingly. The incentive for doing so is an opportunity to improve performance.

7 Summary

The purpose of this work is to understand resource management for a quickly emerging model of Internet computing. The main characteristics of this model are high code mobility and large numbers of anonymous users, which imply different resource demands of applications when compared to traditional operating systems. Efficient use of distributed resources becomes a must if the model is to be useful. We advocate focusing on resource-aware applications. Exploiting changes in resource availability may prevent degradation of quality of service to end users in case of shortages of a particular resource. The proposed system support consists of providing short-term guarantees of resource availability and of resource-sensitive versions of standard libraries. We criticize economics-based approaches to resource management as inadequate for the envisioned system. A prototype environment, based on the J-Kernel, has just become operational. Our most immediate plan is to collect experimental data.

References

1. Edell, R, McKeown, N and Varaiya, P. *Billing Users and Pricing for TCP.* IEEE Journal on Selected Areas in Communications, Vol. 13, No. 7, September 1995.
2. Hawblitzel, C, Chang, C, Czajkowski, G, Hu, D. and von Eicken, T. *Implementing Multiple Protection Domains in Java.* Proc. Annual USENIX Conference, New Orleans, LA, June 1998.
3. Heiser, G, Lam, F, and Russel S. *Resource Management in the Mungi Single-Address-Space Operating System.* Proc. 21st Australasian Computer Science Conference, Perth, Australia, February 1998.
4. Waldspruger, C. A *Spawn: A Distributed Computational Economy.* IEEE Transactions on Software Engineering, 18(2):103-117, February 1992.