

Issues in Improving Web Server performance
Ravindranath Kokku, Lorenzo Alvisi, Harrick Vin
Department of Computer Sciences, University of Texas at Austin

in collaboration with

Ram Rajamony
IBM Austin Research Laboratory

Server clusters are behind most of the dynamic web services—such as e-commerce and web personalizations—deployed on the Internet today. Such systems are attractive because they combine the price-performance characteristics of commodity hardware with the ability to scale server throughput by adding more server nodes to the cluster. To respond to the explosive growth in the number of users on the Internet, the design of server clusters will need to address two fundamental problems.

- *Scalability*: Today, because of the difficulty in balancing the load on the servers within a cluster, the number of client requests that a cluster can service does not grow linearly with number of servers.
- *Service differentiation*: Today, clusters that provide multiple services partition their re-sources statically among them. Thus, service differentiation is accomplished simply by assigning different size partitions to services. This approach, however, does not scale; even if machines are cheap, they occupy physical space and need to be maintained. Further, with this approach, the sizes of the partitions are governed by the peak resource needs of the services, an approach that in general, yields poor utilization of cluster resources.

The long-term goal of this project is to investigate techniques for effectively managing resources—such as CPU, memory, disks, and network bandwidth—within a server cluster. Our objective is to design a cluster operating system that, in addition to efficiently managing cluster resources, provides programming abstractions and interfaces that simplify the development of cluster applications.

2 Plan of Work

As a first step towards accomplishing our long-term objective, we are focusing on the request distribution problem. Recent research has shown that distributing client requests within a cluster based on the requested *content* improves throughput and overall cluster resource utilization. However, existing methods do not cope well with persistent client connections, wherein a client uses the same TCP connection to send multiple, possibly unrelated requests to a server.

We have designed a distributed TCP implementation that efficiently distributes requests arriving on persistent connections. Our approach enables each request to be serviced by the cluster member that can best serve the request, while at the same time minimizing the overhead involved in handling the request. Thus, the “best” server for a request directly sends the response to the client in a manner that is transparent to the client.

We are currently building a simulator to compare and study the relative performance of our proposed strategy against the existing strategies. We will be implementing our design in the TCP stack of the Linux operating system. The distributed TCP

infrastructure we are developing can also be used to solve related problems in clusters that host e-business services and middle-tier applications.

We also propose to design and implement a thin distributed shared memory (DSM) layer for a server cluster. Today, the need for sharing data structures across servers within a cluster is either addressed in an ad hoc fashion (by developing data structure specific algorithms) or is not addressed at all. This limits scalability and presents an awkward programming abstraction for applications.

To illustrate, consider the problem of balancing load across servers within a cluster. If services are stateless and if each request requires roughly the same amount of resources, then a simple round-robin scheme that distributes incoming requests to servers within a cluster is likely to work well. Even when the resource requirements for requests differ from one another, simple load balancing techniques such as shortest-request-queue-first work well. Unfortunately, today's web services are "stateful"; they maintain state about the actions that users may have taken during a session (e.g., the set of links that a user has traversed before reaching this page, the content of the shopping cart, etc.). Because this state is accessed and updated for each request within a session and since there is no mechanism for sharing this information across servers, all requests within a session are directed to the same server within the cluster. This "stickiness" inherently limits scalability and complicates the task of managing cluster resources.

We propose to design and implement a thin DSM layer that addresses these problems. With this, multiple servers will be able to share transparently any state information maintained for a session. This will allow a single session to span multiple servers, thereby simplifying load balancing and improving scalability. Further, a DSM provides a familiar programming model, which is likely to ease the development of web services.

The main challenges here are to define the functionality of the thin DSM layer and to develop mechanisms for their efficient implementation. We expect our DSM layer to combine some of the guarantees provided by conventional DSM systems—such as Treadmarks and Munin—with the high-performance and scalability requirements of web services. Our approach to achieve this goal will be two-fold. First, we will study the nature of interactions between e-business applications and web server infrastructures (e.g., WebSphere) and identify the aspects of conventional software DSM systems that are necessary for this class of services. Second, we will investigate techniques for utilizing programmable network adapters to implement efficiently the functions that we will identify.