RACC: An Approach to Cluster-Based Web Servers

Xiaolan Zhang, Ravi Shanmugan, Michael Barrientos, J. Bradley Chen
Harvard University

RACC: An Approach to Cluster-Based Web Servers

Xiaolan Zhang, Ravi Shanmugan, Michael Barrientos, J. Bradley Chen
Division of Engineering and Applied Sciences
Harvard University

Abstract

RACC is a cluster-based design for scalable cost-effective web servers. Organized around the goal of locality enhancement, the RACC approach seeks to distribute requests arriving at the cluster among the nodes so as to enhance the locality of reference that occurs on individual nodes in the cluster. By improving locality on individual cluster nodes, we can reduce their working set sizes, thereby achieving superior performance for less cost than conventional approaches. We describe here the RACC architecture and its prototype implementation on Windows NT.

1. Introduction

Cluster-based web server designs have great potential because of its scalability and cost-effectiveness. A simple approach to building a cluster-based web server is to put an HTTP request router or “IP sprayer” between the Internet and a cluster of web servers. The router distributes HTTP requests among the cluster nodes, typically assigning clients to server nodes in a round-robin fashion. Adding more nodes to the cluster can increase the aggregate performance of the cluster.

This simple approach to clustering is not a panacea. For example, a server node for a large web site might require a large amount of physical memory in order to handle requests efficiently. Each node added to the system will receive requests for the same set of documents, and so will have the same large memory requirements. If the document set grows beyond the memory size of the server nodes, all nodes will begin to thrash. As a result the server nodes tend to be either expensive, slow, or both.

The RACC design seeks to eliminate this inefficient resource usage by enhancing the inherent locality of the request streams delivered to the server nodes. Rather than distributing requests in a round-robin fashion, it distributes requests based on the URL, such that the current working set of the web server is partitioned among the server nodes. We achieve this improvement by replacing the IP sprayer with a Smart Router in RACC. This design has a number of advantages over the standard IP sprayer approach. First, each node in the cluster is responsible for only a fraction of the total working set of the web server. Second, the size of each node’s working set decreases each time a node is added to the cluster. The Smart Router can tune the load presented to each node in the cluster based on that node's capacity. This makes it possible to build the cluster out of relatively inexpensive machines, and to increase the capacity of the cluster by small increments.

2. Implementation

The heart of the RACC cluster is the Smart Router. The Smart Router is partitioned into two layers, the user level High Smart Router (HSR), and the kernel level Low Smart Router (LSR).

The kernel-LSR is a driver sitting above NT’s TCP/IP TDI interface. It listens on the HTTP port for a connect request. When a request is received, TCP passes it to the LSR. The LSR extracts the URL from the request and passes it up to HSR. The LSR then waits for the HSR to indicate which cluster node should handle the request. The LSR maintains TCP connections with each cluster node on which it forwards requests. After delivering a request, the LSR ferries data between the two connections until either end closes the connection. A key implementation challenge in the LSR is efficient handling of many simultaneous TCP/IP connections.

The job of the HSR is to monitor the state of the document store, the nodes in the cluster, and properties of the documents passing through the LSR. It then must use the information to make decisions about how to distribute requests over RACC cluster nodes. Internally, the HSR builds a tree structure that matches the document store name-space. Leaves in the tree represent documents that can be requested from the web site. Nodes in the tree represent internal nodes in a hierarchical name space. As the HSR processes requests, it annotates the tree with information about the document store to be applied in load balancing. This information could include document sizes, number of times each document is requested, and compute cycles required to deliver a given document.

We evaluated our design for both web file service and dynamically generated web pages using Lotus Domino. Our preliminary experiments show an improvement in throughput of up to 8x over IP Sprayer for an artificial static web file service workload, and a 20% improvement for a workload based on Lotus Domino.