LinuxDirector: A Connection Director for Scalable Internet Services

ZHANG Wensong (章文嵩), JIN Shiyao (金士尧) and WU Quanyuan (吴泉源)

National Laboratory for Parallel & Distributed Processing, National University of Defense Technology, Changsha 410073, P.R. China
E-mail: wensong@LinuxVirtualServer.org
Received July 24, 1999; revised June 8, 2000.

Abstract LinuxDirector is a connection director that supports load balancing among multiple Internet servers, which can be used to build scalable Internet services based on clusters of servers. LinuxDirector extends the TCP/IP stack of Linux kernel to support three IP load balancing techniques, VS/NAT, VS/TUN and VS/DR. Four scheduling algorithms have been implemented to assign connections to different servers. Scalability is achieved by transparently adding or removing a node in the cluster. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately. This paper describes the design and implementation of LinuxDirector and presents several of its features including scalability, high availability and connection affinity.

Keywords Internet services, server clustering, load balancing, high availability

1 Introduction

With the explosive growth of the Internet and its increasingly important role in our lives, traffic on the Internet is increasing dramatically, which has been growing at over 100% annual rate. More and more sites are often unable to serve their workload, particularly during peak periods of activity. Some of them have already received tens of millions hits per day. The long delay of services will lower the quality of services. With the increasing number of users and the increasing workload, companies often worry about how systems grow over time. Companies are moving their mission-critical applications on the Internet, and any stop of services causes companies to loose customers and money. Therefore, the demand for hardware and software solution to support highly scalable and highly available services is growing urgently. The requirements can be summarized as follows:

- **Scalability**, when the load offered to the service increases, system can be scaled to meet the requirement.
- **24×7 availability**, the service as a whole must be available 24×7, despite of transient partial hardware and software failures.
- **Cost-effectiveness**, the whole system must be economical to afford and expand.
- **Manageability**, although the whole system may be physically large, it should be easy to manage.

A single server is usually not sufficient to handle this aggressively increasing load. The server upgrading process is complex, and the server is a single point of failure. The higher end the server is upgraded to, the much higher cost we have to pay.

Clusters of servers, connected by a faster network, are emerging as a viable architecture for building a high-performance and highly available server. This type of loose-coupled architecture is more scalable, more cost-effective and more reliable than a single-processor system or a tightly coupled multiprocessor system. However, there are challenges to provide transparency, efficiency, scalability and high availability of parallel services in the cluster.
LinuxDirector\(^{[18]}\) is our solution to the requirements. LinuxDirector is a connection director that supports load balancing among multiple Internet servers, which can be used to build scalable Internet services based on clusters of servers. Prototypes of LinuxDirector have already been used to build many sites of heavy load in the Internet.

LinuxDirector directs network connections to different servers according to scheduling algorithms and makes parallel services of the cluster appear as a virtual service on one IP address. Client applications interact with the cluster as if it were a single high-performance and highly available server. The clients are not affected by interaction with the cluster and do not need modification. Scalability is achieved by transparently adding or removing a node in the cluster. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately.

The remainder of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we describe three request dispatching techniques and their working principles, and also discuss their advantages and disadvantages. In Section 4, we describe four scheduling algorithms that have been developed for LinuxDirector. In Section 5, we describe the high availability issue of LinuxDirector. In Section 6, we describe how connection affinity is handled in LinuxDirector. In Section 7, we present some big LinuxDirector application. Finally, conclusion and future work are given in Section 8.

## 2 Related Work

In the client/server applications, one end is the client, the other end is the server, and there may be a proxy in the middle. Based on this scenario, we can see that there are many ways to dispatch requests to a cluster of servers on different levels. Existing request dispatching techniques can be classified into the following categories:

- **The client-side approach**
  Berkeley's Smart Client\(^{[17]}\) suggests that the service provide an applet running at the client side. The applet makes requests to the cluster of servers to collect load information of all the servers, then chooses a server based on the information and forwards requests to that server. The applet tries other servers when it finds the chosen server is down. However, these client-side approaches are not client-transparent. They require modification of client applications, so they cannot be applied to all TCP/IP services. Moreover, they will potentially increase network traffic by extra querying or probing.

- **The server-side Round-Robin DNS approach**
  The NCSA scalable web server is the first prototype of a scalable web server using the Round-Robin DNS approach\(^{[5,12,13]}\). The RR-DNS server maps a single name to different IP addresses in a round-robin manner so that different clients will access different servers in the cluster for the ideal situation and load are distributed among the servers. However, due to the caching nature of clients and hierarchical DNS system, it easily leads to dynamic load imbalance among the servers, thus it is not easy for a server to handle its peak load. The TTL (Time To Live) value of a name mapping cannot be well chosen at RR-DNS. With small values the RR-DNS will be a bottleneck, and with high values the dynamic load imbalance will get even worse. Even if the TTL value is set with zero, the scheduling granularity is per host. Different client access patterns may lead to dynamic load imbalance, because some clients (such as a proxy server) may pull lots of pages from the site, and others may just surf a few pages and leave. Furthermore, it is not so reliable, when a server node fails, the client who maps a name to the IP address will find the server is down, and the problem still exists even if they press "reload" button in the browsers.

- **The server-side application-level scheduling approach**
  EDDIE\(^{[6]}\), Reverse-proxy\(^{[10]}\) and SWEB\(^{[4]}\) use the application-level scheduling approach to build a scalable web server. They all forward HTTP requests to different web servers.