Postprint of a Novel Network Server Architecture Based on Virtualization and Pipeline Technology

Authors: Sun Yuzhong, Zhang Yufang, Wang Ruoni, Yaqiong Li

Date: 2016-11-02T00:00:00+00:00

Abstract

The rapid expansion of Internet scale poses tremendous challenges to network servers: the software architecture of traditional network servers exhibits increasingly prominent shortcomings such as low throughput and poor reliability, owing to tightly coupled processing logic, lack of effective isolation, and monolithic resource management models. Consequently, there is an urgent imperative to develop high-throughput, highly available network server architectures. On the other hand, the benefits afforded by server virtualization—including scalable resource management and isolation between execution environments—present significant opportunities for improving network server architecture design and enhancing both throughput and availability. To address the inherent problems in traditional network server software architectures and the associated practical challenges, this paper proposes and implements a novel virtualization-based pipelined network server architecture, MVMDP, which provides effective isolation guarantees and flexible on-demand resource allocation for each pipeline stage. This architecture realizes mechanisms such as HTTP processing logic pipelining, dynamic pipeline structures, and request-oriented reliability guarantees, thereby ensuring stable and effective service delivery under high load. Experimental results demonstrate that, compared to traditional network server architectures such as Apache, MVMDP achieves a 47% improvement in throughput under high-load conditions. Regarding availability, MVMDP not only leverages virtualization technology to provide security isolation between execution environments but also implements request-oriented reliability guarantee mechanisms, attaining 99% availability—a substantial improvement over traditional network servers.

Full Text

A Novel Network Server Architecture Based on Virtualization and Pipeline Technology

Yuzhong Sun, Yufang Zhang, Ruoni Wang, Yaqiong Li

Abstract

The dramatic expansion of the Internet has posed significant challenges to network servers. Traditional network server software architectures suffer from low throughput and poor reliability due to tightly coupled processing logic, lack of effective isolation, and monolithic resource management models. Consequently, there is an urgent need to develop high-throughput, highly available network server architectures. On the other hand, server virtualization offers advantages such as scalable resource management and isolation between execution environments, creating substantial opportunities for improving network server architecture design and enhancing throughput and availability. To address the problems in traditional network server software architectures and meet real-world challenges, this paper proposes and implements a novel pipeline-based network server architecture called MVMDP (Multi-Virtual-Machine Dynamic-Pipeline). MVMDP provides effective isolation guarantees and flexible on-demand resource flow for each pipeline stage, and implements mechanisms for HTTP processing logic pipelining, dynamic pipeline structures, and request-oriented reliability assurance to ensure stable and effective services under high load. Experiments demonstrate that MVMDP achieves a 47% throughput improvement over traditional web server architectures such as Apache under high-load conditions. In terms of availability, MVMDP not only leverages virtualization technology to provide secure isolation between execution environments but also implements a request-oriented reliability assurance mechanism, achieving 99% availability—a substantial improvement over traditional network servers.

Keywords: network server architecture; virtualization; dynamic pipeline; availability

1 Introduction

Since the 1990s, the Internet has achieved massive-scale popularization. With the rapid growth of Internet scale, network servers face challenges of massive concurrency and high availability. Due to structural limitations, traditional network servers suffer from low throughput and poor reliability. Therefore, there is an urgent need to develop high-throughput, highly available network server architectures. On the other hand, the advantages of server virtualization technology—such as scalable resource management and isolation between execution environments—open promising prospects for improving network server architecture design and enhancing service quality.

This paper proposes and implements a novel network server architecture based on virtualization technology and pipeline concepts: Multi-Virtual-Machine Dynamic-Pipeline (MVMDP). Based on this architecture, we have implemented HTTP request processing pipelining, dynamic pipeline structures, and request-oriented reliability assurance mechanisms (HTTP request-oriented admission control).

1.1 Challenges Facing Network Servers

Entering the 21st century, the Internet has continued its rapid growth trajectory from the 1990s. As of February 2009, the number of websites worldwide exceeded 200 million [1], and since 1997, Internet data volume has been doubling annually—a phenomenon known as "Moore' s Law for data transmission" [2]. This dramatic expansion and growing demand have brought unprecedented challenges to network servers.

The first challenge is massive concurrency. Along with the expansion of Internet scale comes rapid growth in user numbers. As of February 2009, the number of active Internet users worldwide exceeded 1.5 billion [3]. Such a massive user base presents a challenge of large-scale concurrency. As of February 2009, Google received 1.5 billion daily visits [4], and after the opening ceremony of the Beijing Olympics, the official Olympic website received 183 million daily visits. Requiring network servers to continuously respond to such massive user requests with high throughput is extremely difficult.

The second challenge is high availability. After nearly fifteen years of large-scale popularization, the Internet has become critical infrastructure for socio-economic development. From portal websites to enterprise sites, e-commerce, and even encyclopedias [5], the Internet has penetrated every aspect of social life. This heavy dependence on the Internet creates demands for 7×24 hour high availability, which is difficult to satisfy.

First, security attack threats are becoming increasingly prominent. Security attacks have accompanied the Internet since its early days, evolving from early Denial of Service attacks to today' s application-oriented attacks [6], constantly threatening stable and secure Internet services. In the third quarter of 2008 alone, Akamai detected security attacks from 179 countries—an increase of over 30% from the previous quarter [7]—with a substantial portion targeting Internet services. It is estimated that 30%-40% of global e-commerce websites face security attack threats [8], and since some attacks "do not require sophisticated techniques" [9], this problem will inevitably become more severe.

Second, Internet requests exhibit Poisson distribution characteristics [10][11]. Research indicates that peak HTTP request rates exceed average rates by 8-10 times [12]. When request arrival rates exceed network server processing capacity, resource contention and related overhead cause throughput to decline [13]. An extreme example occurred on October 30, 2007, when the Olympic ticket website received 8 million visits per hour, causing the site to "crash."

This requires network servers to support not only normal loads but also peak loads.

1.2 Existing Network Server Architectures and Their Problems

Several architectures have been proposed for network servers, including Multi-Process (MP), Multi-Thread (MT), Single-Process Event-Driven (SPED), Asynchronous Multi-Process Event-Driven (AMPED), and Pipeline.

The multi-process architecture consists of a process pool, where each process executes basic operations related to serving a request. Because it uses multiple processes, it can concurrently handle many HTTP requests. The multi-thread architecture consists of a thread pool belonging to the same kernel thread within a shared address space. Each thread handles a client request and independently executes request processing steps. The single-process event-driven architecture uses non-blocking I/O operations, with one process interleaving the handling of all requests, avoiding context switching and process/thread synchronization overhead. The asynchronous multi-process event-driven architecture consists of a main server process and many helper threads primarily used for handling read/write operations. With multiple helpers serving disk-oriented requests, the main network server process can handle only cache-hit requests, reducing blocking caused by disk I/O and improving performance. Pipeline architectures typically divide HTTP processing logic into multiple stages, with each stage having separate processing logic, and multiple stages forming a pipeline.

However, in existing architectures, HTTP processing logic exists in a tightly coupled relationship (within the same process or thread), which affects system reliability and performance: a failure in one processing logic often affects the entire HTTP processing flow, and interference between processing logics impacts overall system throughput and reliability. Additionally, this tight coupling makes server software structure optimization and upgrading difficult, as changes to any mechanism may affect other mechanisms in the program. Furthermore, existing architectures contain elements that grow linearly with the number of requests. For example, in multi-process (multi-thread) architectures, the number of processes (threads) grows linearly with requests; in event-driven architectures, the file descriptor set size grows linearly with requests. When the number of requests is large, these elements become system bottlenecks, affecting server scalability. Moreover, existing architectures rely on operating system resource management mechanisms with monolithic resource allocation patterns, making it difficult to meet resource demands when load requirements fluctuate. Consequently, existing architectures cannot match network server performance with hardware resources.

On the other hand, server virtualization technology has flourished in recent years, with its potential for scalable resource management, isolation between execution environments, and cost reduction opening attractive prospects for using virtualization to improve network server architecture design and enhance

service quality.

1.3.1 Xen Virtualization Technology Xen [14] is an open-source system-level virtual machine monitor (VMM) developed by the University of Cambridge that can run up to 128 fully functional operating systems on a single computer. [Figure 1: see original paper] shows the Xen architecture. As illustrated, Xen uses paravirtualization technology, requiring explicit modifications to the operating system (forming a Guest OS) to run on it, but Xen provides compatibility for user applications. The Guest OS and Xen coordinate their work, enabling Xen to achieve high-performance virtualization without special hardware support. Typically, Xen incurs only about 2% performance overhead, with worst-case overhead of 8% [14], which contrasts sharply with other full virtualization solutions that can cause up to 20% overhead. Even on architectures traditionally hostile to virtualization (x86), Xen demonstrates excellent performance. To date, Xen supports multiple mainstream operating systems (including Linux and Unix). With hardware extensions (Intel' s VT-X Vanderpool or AMD' s Pacifica), Xen even allows unmodified Guest OSes, including Windows, to run in Xen virtual machines. In addition to providing effective isolation, Xen enables live migration of virtual machines between physical hosts without stopping execution. During migration, memory is repeatedly copied to the target machine while the VM continues running. Before the VM begins execution at its final destination, there is a very brief 60-300ms pause for final synchronization, but to users this appears as seamless migration.

1.3.2 Distributed System-Level Virtual Machine Monitor and Rainbow-A System The Distributed Virtual Machine Monitor (DVMM) is a novel virtualization technology developed by the Operating System Group at the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, based on Xen [14]. It manages distributed physical server resources by integrating them to help upper layers establish a series of virtual servers, thereby providing flexible and efficient resource management models. Resource management in DVMM sits between application-layer resource management and various resource virtualization modules, with its main function being to implement global mapping and dynamic variable mapping between virtual components in the virtual platform and virtual resources on the physical platform. Through global mapping, it supports access to and sharing of globally virtualized physical resources by the virtual platform; through dynamic variable mapping, it supports on-demand, dynamic, and transparent "flow" of virtualized physical resources between different virtual platforms and between different virtual nodes within the same virtual platform, ensuring the entire system maintains high resource utilization, application execution efficiency, and load balancing.

Rainbow-A is a virtual service computing platform built by the Operating System Group at the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, based on the

capability server concept [15][16]. Built on DVMM, it implements the "Capacity Service Computing Framework (CSCF)" [46] and provides multi-VM capability flow mechanisms [18] and multi-level resource scheduling mechanisms [19], effectively meeting enterprise computing environment needs for flexible resource management and optimization of large-scale coexisting services. The DVMM and Rainbow-A architecture is shown in [Figure 2: see original paper].

1.4.1 Contributions of This Paper

To address problems in existing network server architectures, this paper proposes and implements a pipeline-based network server architecture using virtualization technology—Multi-Virtual-Machine Dynamic-Pipeline. Based on this architecture, we have implemented HTTP request processing pipelining, dynamic pipeline structures, and request-oriented reliability assurance mechanisms.

The MVMDP architecture adopts a decomposition approach, dividing the entire HTTP processing logic into phases, with each phase processed separately to form a business pipeline for HTTP processing logic. Simultaneously, each phase is placed in a separate execution entity—a virtual machine. The processing phase and virtual machine are bound together to form a pipeline stage. Through this mechanism, HTTP processing logic is decoupled from the tight coupling in traditional architectures, becoming loosely coupled. This architecture has the following characteristics:

1. More efficient processing mode. Dividing HTTP processing logic into phases makes the business logic in each phase simpler and faster to respond. Additionally, the pipeline structure avoids linear growth of process/thread count with request count, reducing the per-request cost from a process/thread to a request descriptor structure.

2. Higher availability. Leveraging the asynchrony between pipeline stages and cached request information within stages, failed request processing logic can be recovered on the pipeline, improving system availability.

3. More flexible resource management mode. In addition to providing an execution environment for pipeline stages, virtual machines can be viewed as resource "containers." Using virtualization' s resource flow characteristics between VMs can better meet resource demands of each pipeline stage during load fluctuations and requirement changes, freeing the architecture from the limitation of resource allocation only through the operating system.

4. More flexible software structure. Each pipeline stage in the pipeline architecture only handles one logic of HTTP processing. Compared with traditional architectures that must handle all HTTP request logic, implementation is simpler and facilitates individual optimization of each stage' s implementation. With well-defined interfaces between stages, a pipeline stage' s implementation can even be completely replaced without affecting

other stages, greatly improving software structure flexibility and extensibility.

These characteristics enable the new architecture to provide higher throughput and reliability for network servers.

1.4.2 Research Background and Motivation

This research is supported by the "863" project "Research on Novel Network Servers Based on Virtualization Technology" undertaken by the Institute of Computing Technology, Chinese Academy of Sciences, and represents further development of the "Hundred Talents Program" project "Capability Server Technology Research." The goal of the "863" novel network server project is to build a high-performance, highly reliable network server architecture based on the DVMM developed by the Operating System Group at the Key Laboratory of Computer System and Architecture. During our research, we found that combining pipeline architecture with the "Capacity Service Computing Framework" developed in the "Capability Server" project, along with its multi-VM capability flow mechanism [18] and multi-level resource scheduling mechanism [19], effectively meets the resource capability flow requirements between pipeline stages in pipeline server architectures, thereby adapting to load-induced resource demand changes.

2.1 Related Research on Network Server Software Architecture

Existing research [20] has shown that network server software architecture is a critical factor affecting server performance. Several architectures have been proposed for network servers, including multi-process, multi-thread, single-process event-driven, asynchronous multi-process event-driven, and pipeline. This section introduces these approaches.

2.1.1 Multi-Process

As shown in [Figure 3: see original paper], the multi-process architecture consists of a process pool, where each process executes basic operations related to serving a request. Because it uses multiple processes, it can concurrently handle many HTTP requests. The Apache server [21] originally adopted this multi-process model. The advantage of this model is its simple structure and easy implementation. However, processes cannot easily share any global information (such as shared cache). Compared with other models, a multi-process web server requires more memory to maintain equivalent cache sizes for each process. Additionally, the system call fork used to create processes has high latency and cannot promptly create enough service processes [22], and process context switching overhead is relatively large. Therefore, this model' s overall performance is lower than other models discussed below.

2.1.2 Multi-Thread

As shown in [Figure 4: see original paper], the multi-thread architecture consists of a thread pool belonging to the same kernel thread within a shared address space. Each thread handles a client request and independently executes request processing steps. The advantage of this model is that all processes can share any global information, particularly shared data caches. The widely used Apache server has been upgraded to a multi-thread model. However, not all operating systems support kernel threads, and sharing data caches between processes can lead to high synchronization overhead.

2.1.3 Single-Process Event-Driven

The single-process event-driven architecture uses non-blocking I/O operations, with one process interleaving the handling of all requests. This avoids context switching and process/thread synchronization overhead. This model has been implemented by Zeus Technology [23]. However, non-blocking I/O operations in this model may actually block, particularly when the operating system has certain restrictions on disk-related operations. Therefore, for disk-intensive workloads, the single-process event-driven model may not perform better than multi-thread models.

2.1.4 Asynchronous Multi-Process Event-Driven

The asynchronous multi-process event-driven architecture consists of a main network server process and many helper threads primarily used for handling read/write operations (as shown in [Figure 6: see original paper]). With multiple helpers serving disk-oriented requests, the main network server process can handle only cache-hit requests, reducing blocking caused by disk I/O and improving performance. However, in this event-driven programming model, system calls like select() are often used for event dispatching, which does not scale well for large numbers of connections, affecting server scalability under high connection counts. Additionally, because this model uses a single-process structure, it cannot effectively utilize multi-core technology.

2.1.5 Pipeline

Research on pipeline architectures has been reported previously. Reference [24] designed a test structure to evaluate the shortest-connection-first algorithm, dividing post-connection processing into three steps: protocol processing, disk service, and network service, with multiple threads handling each step. This structure can be viewed as a simple pipeline where requests flow through three thread pools. The University of California, Berkeley proposed the Staged Event-Driven Architecture (SEDA) [25], which is similar. As shown in [Figure 7: see original paper], SEDA uses event queues to connect multiple worker threads, with each worker thread retrieving event descriptors from the queue for processing. HTTP processing tasks are divided into multiple stages, with each stage handling part

of the processing logic. Stages are isolated by task queues, enabling independent load scheduling between stages. Jilin University implemented a pipeline-based web server structure called PRWS (Pipeline and Resource Manager Web Server) [26], which emphasizes resource management through structured request descriptions, reuse of structured descriptions, and comprehensive resource collection mechanisms to reduce redundant resource requests and improve resource utilization and response throughput. Unlike previous work, the National University of Defense Technology implemented a kernel-level pipeline web server called KETA [27], which divides HTTP request processing into several pipeline stacks, with each thread responsible for a specific stack without interference. Because the server is implemented in the kernel, it can reduce overhead from switching between kernel and user space.

Additionally, as server hardware platforms have evolved toward multi-core, a new Multi-Thread Pipelined web server architecture [28] has been proposed (as shown in [Figure 8: see original paper]). In this model, HTTP processing logic is divided into multiple stages, with each stage completed by a separate thread, and multiple stage threads form a business pipeline. Multiple pipelines can simultaneously provide services within a process according to load changes. This structure fully utilizes multi-core through multi-threading while the pipeline architecture avoids linear growth of thread count with request count.

2.1.6 Summary

From a "cost per request" perspective, multi-process (multi-thread) architectures have a per-request "cost" of one process (thread), which is too expensive. Each process (thread) occupies a certain amount of memory space, and when processing complex logic (such as generating dynamic web pages), the occupied space becomes even larger. For example, the resident size of a default-configured Apache process obtained through the top command is about 7MB. This makes it impossible for multi-process (thread) architectures to support large-scale concurrent connections. When load increases and causes more processes (threads), thrashing occurs due to large numbers of processes (threads) occupying memory space, reducing performance. Additionally, frequent context switching overhead reduces cache hit rates, and increased process count raises scheduling overhead, further reducing overall system performance. Event-driven and pipeline structures do not have these problems. In these structures, only an HTTP request descriptor structure needs to be allocated for each new connection request. Compared with multi-process (multi-thread) structures, the per-connection "cost" is only a descriptor structure occupying less than 1KB, enabling support for more concurrent connections. However, for event-driven models, designing reasonable event dispatch strategies is a difficult task. For example, poorly designed event priorities can cause the system to waste time processing useless events, causing overall throughput to drop sharply.

Additionally, in the above architectures, HTTP processing logic exists in a tightly coupled relationship (within the same process or thread), affecting sys-

tem reliability: a failure in one processing logic often affects the entire HTTP processing flow, and interference between processing logics impacts overall system performance and reliability. Although pipeline structures provide some isolation, the thread pools forming each stage remain within the same process address space, so when one stage fails, it can still affect overall system reliability and performance.

Furthermore, existing architectures contain elements that grow linearly with request count: for multi-thread and multi-process structures, process/thread count grows linearly with connections; for single-process event-driven and asynchronous multi-process event-driven structures, file descriptor set size grows linearly with connections; for pipeline structures, thread pool sizes in each stage grow linearly with request connections. Thus, when network servers handle large numbers of requests, these request-proportional factors become performance bottlenecks, affecting server scalability.

Moreover, existing architectures have monolithic resource request and management patterns that cannot well satisfy resource demands when load requirements fluctuate.

2.2 Prospects of Virtualization Technology for Network Servers

In recent years, server virtualization technology has made great progress and been widely applied. Its functions of service consolidation, security isolation, dynamic resource scaling, and simplified management have profoundly impacted network server development. This section discusses these functions.

2.2.1 Server Consolidation To meet demands for continuous service under massive concurrent requests and high load, service providers typically need to deploy large numbers of network servers. Taking Wikipedia [5] as an example, the entire site is served by 400 servers, of which 250 are web servers. Large-scale services like Google require even more servers. Meanwhile, network service processing logic is becoming increasingly complex, with more application types, and current operating system security mechanisms cannot well satisfy security requirements for various applications coexisting. Therefore, to ensure security isolation, network servers are typically deployed exclusively on isolated physical servers. However, network service requests are volatile, resulting in low resource utilization for exclusively deployed servers most of the time.

[Figure 9: see original paper] shows CPU resource utilization of the Wikimedia server cluster over a 24-hour period. The figure shows that CPU utilization remains below 35% most of the time. In the cloud computing context, large numbers of services need to run on cloud computing service platforms, many of which provide services via the web. Providing a separate physical platform for each different service would be prohibitively expensive. Virtualization technology can create multiple independent operating systems (Guest OS) on a single physical server and provide reliable security isolation between them. Consoli-

dating traditionally exclusive services onto a shared virtualization platform can effectively improve resource utilization and reduce costs. To meet these needs, the industry has launched a series of related products. VMware' s latest infrastructure virtualization software, VMware Infrastructure [29] (as shown in [Figure 10: see original paper]), provides a complete server consolidation solution that can integrate multiple services onto a unified physical platform, providing reliable security isolation between virtual machines to improve resource utilization and reduce costs. Additionally, IBM' s z/VM [30] also provides a virtualization-based infrastructure for server consolidation.

2.2.2 Dynamic Resource Scaling When large numbers of HTTP requests arrive at a network server system, the time and computational resources required to serve these requests depend on many factors, such as request service type, current network bandwidth conditions, and current server resource utilization. Some heavy-load requests require compute-intensive queries, database access, and long response data streams, while light-load requests often only need to read an HTML page or perform simple calculations. Meanwhile, network traffic often exhibits significant fluctuations, with literature [31] revealing self-similar characteristics in both wide-area and local-area network traffic, as well as in web access streams. The combined result of these phenomena is server utilization skew—i.e., load imbalance between servers. Traditional solutions use a series of scheduling algorithms for request distribution [32][33] to evenly distribute load across multiple servers. However, distributed requests have non-uniform demands for various resource types, which cannot guarantee full utilization of all resources on each server.

Virtualization technology provides a new solution to this problem. Using virtualization technology, dynamic resource allocation can be achieved between virtual machines providing services according to load changes, achieving resource-load matching. IBM's PLM [34] and VMware's DRS [35] dynamically adjust resource allocation between partitions (virtual machines) based on resource utilization. Reference [36] proposes a two-layer resource scheduling model that performs scheduling at both the virtual machine layer and server layer. Reference [37] implements a mechanism for adjusting CPU resource allocation among multiple virtual machines, suitable for resource adjustment among multiple VMs providing the same service on the same server. This mechanism uses CPU capped mode to achieve performance isolation between virtual machines, but this mode can lead to reduced resource utilization and service quality. The Operating System Group at the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, proposed a multi-level resource flow mechanism [17]. As shown in [Figure 11: see original paper], this model adds a layer of VM inter-resource and capability scheduling beyond traditional request scheduling, using multi-VM capability flow to achieve better matching between computing capacity/resources and service requests, improving resource utilization. Unlike reference [37], reference [17] provides a resource flow mechanism among VMs deployed across multiple distributed servers provid-

ing different services; moreover, compared with reference [37], this mechanism uses non-CPU-capped mode to achieve CPU and memory flow, obtaining better service quality.

2.2.3 Simplified Management As network service scale and complexity increase, management costs rise accordingly. To improve software delivery efficiency and simplify management, VMware introduced Virtual Appliance [38]. As shown in [Figure 12: see original paper], a virtual appliance packages preinstalled and configured applications (such as Apache) with a streamlined operating system as a single image for delivery. This approach can accelerate software delivery, reduce software management costs, and improve security. Virtual appliances based on Apache are currently available for network applications. However, this approach merely packages existing applications (such as Apache) with operating systems and virtual machines without optimizing the network server architecture for virtualization, leaving inherent defects of traditional network server architectures unaddressed.

Additionally, it should be noted that current network services often adopt multi-tier structures. When multi-tier services are consolidated on a single platform through virtualization, inter-VM communication suffers performance degradation due to intermediate processing. To address this issue, several mechanisms for improving inter-VM communication have been proposed, including Xensocket [39], XenLoop [40], and XWay [41]. Xensocket uses new application programming interfaces, requiring application rewriting to utilize it; XenLoop and XWay are application-transparent and do not require application modifications. Through these mechanisms, inter-VM communication on the same physical machine can achieve performance close to inter-process communication [40].

3 Design Considerations for Multi-Virtual-Machine Dynamic-Pipeline Architecture

Based on the analysis of existing service architecture problems and new possibilities enabled by virtualization technology, we propose the following guiding principles for developing the MVMDP architecture:

(1) Use pipeline concepts to divide HTTP processing logic into stages, simplifying implementation logic for each stage. Simultaneously, avoid linear growth of process/thread count with request count through pipeline structure, reducing per-request cost from a process/thread to a request descriptor structure.

(2) Provide isolated execution environments for pipeline stages using virtualization technology to achieve performance and reliability isolation, improving server availability. The combined effect of these two points is to transform HTTP processing logic from traditional tight coupling to loose coupling.

(3) Leverage asynchrony between pipeline stages and implement request recoverability on the pipeline based on user-level "acknowledgment" (ACK) responses and timeout mechanisms to improve availability.

(4) Treat the virtual machine execution environment as a resource "container." Use the multi-VM resource flow mechanism provided by DVMM and Rainbow-A to achieve flexible resource management that adapts to pipeline stage resource demand changes with load fluctuations. Bind pipeline stages to virtual machine execution environments, with inter-stage communication via TCP/IP and standardized interfaces between stages to facilitate pipeline stage migration as VM images for pipeline reconstruction. This also facilitates individual optimization and upgrading of a pipeline stage' s implementation without affecting other system components.

4 System Evaluation

This chapter presents our evaluation of the MVMDP architecture, first using standard benchmarks for performance evaluation, then conducting availability evaluation.

4.1.1 Experimental Design

To gain intuitive understanding of MVMDP performance, we selected two easily accessible and representative web servers for comparison: Apache, the most widely used industry server, and Flash, a well-known research server. These servers employ different software architectures. In our experiments, we compared the performance of the three web servers on the same physical platform using standard benchmarks, focusing primarily on server throughput.

Typically, after each experiment group completes, the requested content is temporarily stored in the page cache by the server operating system. When the next experiment group runs, since the standard benchmark requests the same content, the server reads data directly from the page cache, essentially performing only network I/O without disk I/O. This scenario does not reflect real-world network server performance. Therefore, we used functionality provided in the Linux /proc filesystem since kernel 2.6.16, setting /proc/sys/vm/drop_caches to 3 to clear the page cache before each experiment group.

For Apache and Flash, in addition to default configurations, we evaluated different configuration parameters to ensure fair comparison. For Apache, we tested performance with maximum client connections of 256 (default), 1024, and 2048. For Flash, we tested with FD_SETSIZE values of 1024 (default) and 65536.

We used four physical servers to build the test platform. Two servers deployed web servers as the server side, one deployed the standard benchmark as the client, and another deployed Linux Virtual Server Clusters (LVS) as the service

access point and load balancing scheduler. The four servers were connected via Gigabit Ethernet.

The experimental configuration is shown in [Figure 13: see original paper]. The MVMDP implementation deployed three VMs as pipeline stages on each of two physical server nodes, forming two pipelines, with LVS distributing requests at the access point. For Apache and Flash, server programs were deployed on two physical servers each, also using LVS for request distribution at the access point. Additionally, LVS periodically sent requests to backend servers to detect service status; when backend servers became unavailable, LVS stopped scheduling new requests to them, improving overall system reliability. The client ran the httperf standard benchmark [43] to send requests to servers through LVS. Httperf can continuously send requests at a given load to web servers, testing performance by measuring server responses. To test server performance, a given file set must be accessed; we used the e-commerce portion of the Specweb2005 [44] file set, which is 5GB in size.

4.1.3 Experimental Results

Performance evaluation results for the three web server architectures are shown in [Figure 14: see original paper]. The horizontal axis represents load in connections per second, and the vertical axis represents server throughput in responses per second. We tested Apache with maximum concurrent connections of 256 (default), 1024, and 2048; Flash with file descriptor set sizes (FD_SETSIZE) of 1024 (default) and 65536; and the MVMDP architecture. The results show that under light load, the three servers have similar performance. However, as load increases, Apache and Flash gradually reach performance bottlenecks and throughput begins to decline, while MVMDP throughput maintains linear growth with increasing load, showing significant performance improvement over the other two architectures. Compared with Apache (256 maximum connections), MVMDP achieves up to 47% performance improvement under high load, with even greater improvement over Flash.

[Figure 14: see original paper] also shows that adjusting maximum concurrent connections does not improve Apache performance; instead, performance decreases. Apache with 2048 and 1024 maximum connections shows significantly lower performance than the default configuration. Similarly, adjusting the select() file descriptor set size does not improve Flash performance; on the contrary, it causes greater performance degradation under high load.

4.2 Multi-Virtual-Machine Dynamic-Pipeline Availability Evaluation

As mentioned earlier, in addition to the security isolation provided by virtualization technology, MVMDP implements a request-oriented reliability mechanism that uses user-level acknowledgment responses and timeout detection to enhance system reliability. This section evaluates this mechanism.

4.2.1 Experimental Design In large-scale network server systems, service status is typically detected through heartbeat mechanisms. When a service becomes unavailable, requests are no longer scheduled to the corresponding node, and the service is restarted to resolve the issue. In this experiment, we selected a general network server system scenario—Linux Virtual Server Cluster + ldirectord + Apache—as a comparison object for reliability testing. Ldirectord [45] is a daemon running on the load scheduling node that monitors backend web server status by periodically sending requests. When it detects a backend web server failure, it can stop scheduling new requests to it by clearing its entry in the LVS allocation table, reducing request errors caused by scheduling requests to a failed node.

In network server systems, software failures constitute the vast majority. Software failures can be divided into operating system crashes, service crashes, service hangs, and data inconsistency. Here we consider only operating system crashes. The MVMDP architecture uses virtualization technology to create multiple operating systems on a single physical server; a crash in one OS does not cause others to crash. Since MVMDP also uses LVS + ldirectord at the access point, and the first pipeline stage directly accepts client requests without a preceding stage, the reliability when the first pipeline stage crashes is essentially the same as the LVS + Apache + ldirectord mechanism—both rely on ldirectord to notify new requests not to be distributed to failed nodes, while requests already distributed to failed nodes are discarded. Therefore, we only simulated second and third pipeline stage crashes in MVMDP, as well as simultaneous crashes of both stages, and compared their availability with the scenario of a single OS crash in the LVS + ldirectord + Apache scheme. In the experiments, crash conditions were triggered by manually restarting the operating system while the server was providing services. System availability is defined as:

Availability = (Total Requests - Failed Requests) / Total Requests

4.2.2 Experimental Results shows timeout request counts and availability for MVMDP when the second pipeline stage crashes, third stage crashes, and both stages crash simultaneously. The results show that when the second or third pipeline stage crashes, only a small number of requests timeout, with system availability reaching nearly 100%. This is because when the second or third stage crashes, the preceding stage detects timeouts for requests sent to that stage (no acknowledgment received within the given time). The system then sends cached request copies to a newly selected next pipeline stage through a scheduler, ensuring that requests already distributed to the failed stage can still be correctly processed. For Apache OS crashes, ldirectord notifies LVS to stop scheduling new requests to the corresponding node, but requests already assigned to these nodes are discarded, causing many requests to enter the system but not be properly processed or recovered, resulting in many timeouts. Compared with traditional reliability mechanisms, MVMDP availability is significantly improved.

5 Conclusion

Network server software architecture has a tremendous impact on server performance and availability. Traditional network servers suffer from low throughput and poor reliability due to inherent architectural defects. Our experiments confirm that even modifying related configurations cannot significantly improve server performance. On the other hand, server virtualization technology' s characteristics of scalable resource management and isolation between execution environments create opportunities for improving network server software architecture design.

Based on real-world challenges and problems in traditional network server architectures, this paper proposes and implements a novel network server architecture based on pipeline and virtualization technology—Multi-Virtual-Machine Dynamic-Pipeline. Based on this architecture, we implemented the following mechanisms:

(1) HTTP Processing Logic Pipelining: MVMDP divides HTTP processing logic into stages, with each stage executed by a separate pipeline stage. This makes the business logic in each stage simpler and more efficient, simplifies event dispatching mechanisms within stages, and reduces interference between events. Simultaneously, the pipeline structure avoids linear growth of process/thread count with request count, reducing per-request cost from a process/thread to a request descriptor structure. Finally, virtualization technology provides reliable security isolation between pipeline stages, improving system reliability.

(2) Dynamic Pipeline Structure: In the dynamic pipeline structure, the relationship between upstream and downstream pipeline stages is dynamically determined. After completing its HTTP processing logic, each pipeline stage selects the next stage to send requests to based on scheduling strategy, rather than fixedly sending requests to the same stage. Various scheduling strategies can be applied for request distribution between stages: simple round-robin can evenly distribute requests among all candidate next stages to ensure load balancing, or requests can be distributed based on feedback about load and resource status to ensure maximum overall resource utilization. Additionally, the request-oriented reliability mechanism can be used on the dynamic pipeline structure to improve reliability.

(3) Request-Oriented Reliability Assurance Mechanism: In MVMDP, request processing and response processes are pipelined, constructing numerous business pipelines that are precisely mapped to VM pipeline structures, thereby establishing an internal service request flow for each business pipeline. This internal service request flow is divided into different phases by the VM pipeline. Phase i is used to detect the working status of its neighbor phase i+1. If phase i+1 fails to respond to requests from phase i within the timeout period, it indicates phase i+1 has failed. The system

then selects a new pipeline stage and sends cached request copies to it for processing. This mechanism can promptly detect pipeline stage failures and ensure requests are correctly processed even when stages fail.

These mechanisms enable MVMDP to provide stable and effective services under high load. Experiments show that compared with traditional web server architectures such as Apache, MVMDP achieves 47% performance improvement under high load. In terms of availability, MVMDP not only leverages virtualization technology to provide secure isolation between execution environments but also implements a request-oriented reliability assurance mechanism. This mechanism achieves 99% availability, representing significant improvement over the heartbeat mechanisms commonly used in traditional network server systems.

Furthermore, this work lays a solid foundation for future MVMDP applications and research on improving overall MVMDP performance using inter-VM capability flow and cooperative caching based on DVMM.

References

[1] Netcraft. February 2009 Web Server Survey
[2] Odlyzko, A. Internet Traffic Growth: Sources and Implications. Proceedings of SPIE Optical Transmission Systems and Equipment for WDM NetworkingII, Vol. 5247: 1-15
[3] Miniwatts Marketing Group. World Internet Users and Population Stats. http://www.internetworldstats.com/stats.htm. 2009
[4] Alexa Internet. Global Top Sites. http://www.alexa.com/
[5] Wikipedia. Wikipedia' s main page. http://en.wikipedia.org/
[6] David Scott, Richard Sharp. Abstracting application-level web security. Proceedings of the 11th international conference on World Wide Web, 2002: 396-407
[7] Akamai Technologies. The state of Internet. 3rd Quarter, 2008. http://www.akamai.com/
[8] L. Lorek. New e-rip-off maneuver: Swapping price tags. ZD-Net. 5th March, 2001. http://www.zdnet.com/intweek/stories/news/0,4164,2692337,00.html
[9] David Scott, Richard Sharp. Abstracting application-level web security. Proceedings of the 11th International Conference on World Wide Web, 2002: 396-407
[10] Jijun Lu, Swapna S. Gokhale. "Web server performance analysis" . Proceedings of the 6th International Conference on Web engineering. July 2006
[11] Raphael Guerra, Raphael Guerra, Gerhard Fohler. "Attaining soft real-time constraint and Energy-efficiency in Web Servers" . Proceedings of the 2008 ACM Symposium on Applied Computing. Fortaleza, Ceara, Brazil, 2008
[12] Mogul JC. Network behavior of a Busy Web Server and Its Clients. Technical Report, WRL 95/5, Palo Alto: DEC Western Research Laboratory, 1995
[13] Vipul Mathur, Varsha Apte. "An Overhead and Resource Contention Aware Analytical Model for Overloaded Web Servers." Proceedings of the

6th International Workshop on Software and Performance. Buenes Aires, Argentina, 2007
[14] P.Barham, B.Dragovic, K.Fraser, S.Hand, T.Harris, A.Ho, R.Neugebauery, I.Pratt,and A.Wareld. Xen and the Art of Virtualization. SOSP' 03. ACM, 2003
[15] Guojie Li, Strategic Thinking on Supercomputing and Capability Services. Information Technology Letters, January 2005, 3(1): 1-8.
[16] Yuzhong Sun, Liang Chang, The Foundation of Future Enterprise Informatization—Capability Service System, Information Technology Letters, March 2006, 4(2): 21-33
[17] Ying Song, Yaqiong Li, Hui Wang, Yufang Zhang, Binquan Feng, Hongyong Zang, Yuzhong Sun, "A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing" , International Conference on High Performance Computing (HiPC), 2008, LNCS 5374: 220-231
[18] Ying Song, Yuzhong Sun, Hui Wang, Xining Song, "An Adaptive Resource Flowing Scheme amongst VMs in a VM-Based Utility Computing" , IEEE 7th International Conference on Computer and Information Technology (IEEE CIT 2007), Oct 2007: 1053-1058
[19] Ying Song, Hui Wang, Yaqiong Li, Binquan Feng, Yuzhong Sun, "Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center", 9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid09), Shanghai, China, May 2009: 18-21
[20] Pariag, D., Brecht, T., Harji, A., Buhr, P., Shukla, A., and Cheriton, D. R. 2007. "Comparing the Performance of Web Server Architectures." In Proceedings of the 2nd ACM Sigops/Eurosys European Conference on Computer Systems 2007. EuroSys ' 07. ACM, New York, NY: 231-243
[21] The Apache Software Foundation. The Apache HTTP Server Project, 2003. http://httpd.apache.org
[22] Felix von Leitner. Scalable Network Programming. 2003. http://bulk.fefe.de/scalable-networking.pdf
[23] Zeus Technology Limited. Zeus Web Server, 2003. http://www.zeus.com/
[24] Crovella ME, Frangioso R, Harchol-Balter M. Connection scheduling in Web Servers. Technical Report, BUCS-TR-99-003, Boulder, 1999
[25] Welsh M, Culler D, Brewer E. SEDA: An Architecture for Well-conditioned, Scalable Internet Services. In Proceedings of the 18th Symposium on Operating Systems Principles (SOSP-18). Banff, 2001
[26] Nianmin Yao, Mingyang Zheng, Jiubin Ju. High-Performance Web Server Based on Pipeline. Journal of Software, 2003
[27] Yusong Tan, Huadong Dai, et al. Kernel-level Web Server Based on Software Pipeline Architecture—KETA. Computer Engineering and Science. Oct 2006: 138-142
[28] Gyu Sang Choi, Jin-Ha Kim, Deniz Ersoz and Chita R. Das. A Multi-Threaded PIPELINED Web Server Architecture for SMP/SoC Machines. The International World Wide Web Conference Committee. Chiba, Japan, 2005
[29] VMware Inc. "VMware Infrastructure Architecture"http://www.vmware.com/files/cn/pdf/architecture_wp
[30] IBM Corporation. IBM z/VM Homepage, 2009. http://www.vm.ibm.com/

[31] Kihong Park, Gitae Kim, Mark Crovella, "On the Effect of Traffic Self-similarity on Network Performance" , In Proceedings of the 1997 SPIE International Conference on Performance and Control of Network Systems, 1997
[32] Jin-Ha Kim, Gyu Sang Choi, Chita R. Das. "Coscheduled Distributed-Web Servers on System Area Network" . Journal of Parallel and Distributed Computing. Volume 68, Issue 8, August 2008
[33] Michele Colajanni, Philip S. Yu, Valeria Cardllini, Emiliano Casalicchio. The State of the Art in Locally Distributed Web-Server Systems. ACM Computing Surveys, Vol. 34, No. 2, June 2002
[34] IBM Redbook: "Advanced POWER Virtualization on IBM System p5: Introduction and Configuration" , Jan 2007
[35] VMware Infrastructure: "Resource Management with VMware DRS"
[36] J.Xu, M.Zhao, etc., "On the Use of Fuzzy Modeling in Virtualized Data Center Management" , ICAC07: 25
[37] P.Padala, X.Zhu, M.Uysal, etc., "Adaptive Control of Virtualized Resources in Utility Computing Environments" , EuroSys' 07: 289-302
[38] VMware Inc. Virtual Appliances: A New Paradigm for Software Delivery. http://www.vmware.com/, 2008
[39] X. Zhang, S. McIntosh, P. Rohatgi, and J.L. Griffin. Xensocket: A High-throughput Interdomain Transport for Virtual machines. In Proc. of Middleware, 2007
[40] J. Wang, K. L. Wright, and K. Gopalan. XenLoop: A Transparent High Performance Inter-VM Network Loopback. In Proceedings of the 17th International Symposium on High Performance Distributed Computing, June 2008
[41] K.Kim, C.Kim, S.-I.Jung, H.Shin and J.-S. Kim. Inter-domain Socket Communications Supporting High Performance and Full Binary Compatibility on Xen. In Proc. of Virtual Execution Environments,
[42] LINUX VIRTUAL SERVER. 2002. Linux Virtual Server project. http://www.linuxvirtualserver.org/.
[43] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In The First Workshop on Internet Server Performance, Madison, WI, June 1998: 59–67
[44] Standard Performance Evaluation Corporation. The SPECweb2005 benchmark. http://www.spec.org/osg/web2005/
[45] Jacob Rief, Horms. Ldirectord's Main Page. http://www.vergenet.net/linux/ldirectord/.2008
[46] Ying Song, Hui Wang, Yaqiong Li, Yuzhong Sun, Yu Zeng, "Can VoD Streaming Service Co-Exist with Other Services on a VM-Based Virtualized Computing Platform?" , HPC workshop on SC2007, Nov, 2007

Author Biographies

Yuzhong Sun: Researcher at Institute of Computing Technology, Chinese Academy of Sciences. Email: yuzhongsun@ict.ac.cn
Yufang Zhang: Master at Institute of Computing Technology, Chinese

Academy of Sciences
Ruoni Wang: Master at Institute of Computing Technology, Chinese Academy of Sciences
Yaqiong Li: Ph.D. at Institute of Computing Technology, Chinese Academy of Sciences

Note: Figure translations are in progress. See original paper for figures.