Scheduling Technology for Application Load in Capability Service Computing (Postprint)

Authors: Feng Binquan, Song Ying, Sun Yuzhong

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

Abstract

Currently, the utilization of virtualization technology for service integration in enterprise-level data centers has become increasingly prevalent. This paper proposes a dynamically scalable virtual capability service computing framework by combining the concepts of virtualization technology and utility computing, and based on this framework, designs a dynamic feedback load balancing scheduling algorithm named DFBS. The algorithm achieves the scheduling of service requests within the capability service framework through the layer-4 network request forwarding mechanism of the Open Systems Interconnection model, attempting to accomplish optimized matching and scheduling of network requests as resources flow within this framework. Furthermore, we employ curve fitting techniques on several selected types of Web applications to abstract a functional relationship between Service Level Agreement (SLA) and both the number of concurrent service requests and required resources. Experimental results fully demonstrate that when resource contention occurs, DFBS can cooperate with resource flow to jointly achieve the objective of improving resource utilization while guaranteeing service quality in large-scale heterogeneous service concurrent environments.

Full Text

Preamble

Vol. 7 No. 6 Information Technology Letters

Capability Service Computing Application Load Scheduling Technology
Feng Binquan, Song Ying, Sun Yuzhong

Abstract

Currently, enterprise data centers increasingly employ virtualization technology for service consolidation. This paper proposes a dynamically scalable virtual ca-

pability service computing framework that integrates virtualization technology with utility computing concepts, and designs a dynamic feedback load balancing scheduling algorithm called DFBS. The algorithm implements service request scheduling within the capability service framework through Layer 4 network request forwarding mechanisms in the Open Systems Interconnection model, aiming to optimize the matching of network requests with resource flows under this framework. We also utilize curve fitting techniques to abstract functional relationships between Service Level Agreements (SLAs) and both concurrent service request volumes and required resources for several selected Web applications. Experimental results demonstrate that when resource contention occurs, DFBS can cooperate with resource flows to improve resource utilization while guaranteeing service quality in large-scale heterogeneous concurrent service environments.

Keywords: data center; virtualization; utility computing; resource flowing; Web load balancing scheduling

1. Introduction

Informatization serves as a crucial cornerstone for the rejuvenation and prosperous development of the Chinese nation. The core challenge of informatization lies in reducing the societal cost of information access and comprehensive information processing costs, while enhancing the efficiency, availability, security, and maintainability of societal information processing to promote sustainable development. Addressing these issues represents a significant challenge we face.

Using or providing a computing service necessitates acquiring computing resources to support that service. Under existing computing paradigms, this typically means purchasing and owning these resources, along with possessing the specialized knowledge and talent required for deployment, management, and maintenance, which results in high total cost of ownership for users. Utility computing emerges as a commercial model proposed by the IT industry to solve these problems, also known as on-demand computing. In this model, resources (such as computing and storage) are encapsulated and provided as a metered service, similar to public utilities like electricity, water, and natural gas. Obtaining computing services no longer requires owning computing resources; instead, users rent or purchase usage rights as needed. Rental costs are metered based on resource consumption, akin to familiar public utility services. In recent years, with the development of virtualization technology, increasing research has focused on using virtualization to consolidate application services into shared resource pools, creating virtual utility computing environments to improve resource utilization and reduce maintenance costs.

Meanwhile, contemporary computer technology has entered a network-centric era. Due to its simplicity, manageability, and maintainability, the client/server model has been widely adopted on networks. In the mid-1990s, the emergence of the World Wide Web brought graphical, information-rich online content to the

general public through simple operations. The Web is evolving from a content delivery mechanism into a service platform, with numerous services and applications (such as news services, online banking, and e-commerce) implemented through Web technologies. This has driven explosive growth in both Internet users and traffic. Consequently, how to utilize virtual utility computing platforms to provide Internet services has become a primary research focus and challenge for enterprise data centers.

To support large-scale heterogeneous concurrent network services in utility computing environments, a key technology that urgently needs resolution is resource management—specifically, how to dynamically provision resources for multiple concurrent network services. This technology encompasses three aspects: system-allocated resources must adapt to changes in application quantity and type; the system needs to adjust resource allocation to meet evolving user demands during runtime; and system resource provisioning must satisfy Service Level Agreements (SLAs). Current resource management solutions primarily utilize virtualization technology to dynamically adjust resource quotas among different virtual machines within a physical node, achieving on-demand resource flowing. However, existing virtualization technology can only realize resource flowing within physical nodes, while the total resources of a single service node fall far short of meeting large-scale network service demands. Common network services are deployed in distributed clusters. With virtualization technology, this deployment architecture can be extended into distributed virtual machine clusters, where each virtual machine may run on different physical nodes. Through a request scheduler, Web requests sent by clients can be scheduled across the Web virtual cluster. In this new application environment, virtual machine resources change dynamically on demand, while service request scheduling must accurately assess these servers' dynamic processing capabilities for request allocation. This creates a matching problem between resource flowing and application load scheduling.

This paper addresses the resource flowing and application load matching problem in the new virtual utility architecture by proposing a predictive model for service resource requirements and dynamic virtual server capabilities, and designing a scheduling system based on dynamic feedback service requests. In distributed resource-elastic virtual machine clusters, we first propose an integrated deployment solution for configuring and maintaining numerous network services within this system. Then, by dynamically adjusting the distribution of service requests among these virtual servers on demand, we achieve cross-physical-node management and maintenance of the entire system resource pool. Ultimately, this aims to improve system resource utilization, reduce maintenance costs, enhance system throughput, and guarantee service quality, thereby exploring a new path for resource management methods in utility computing environments.

1.1 Utility Computing Based on Virtualization Technology

Utility computing is not a new concept. In 1961, Turing Award winner John McCarthy envisioned in his speech at MIT' s centennial celebration: "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just like the telephone system···utility computing will become the basis of a new and important industry" [1]. IBM and other mainframe vendors dominated this business model for the subsequent two decades, providing computing and storage resources to banks and large organizations through data centers worldwide, representing the early form of utility computing development. In the mid-to-late 1970s, microprocessor technology development and the emergence of minicomputers enabled people to acquire computer hardware at much lower prices than before. Computer users tended to purchase their own computing centers rather than buying usage rights for computing resources from large vendors. It was not until the late 1990s that the concept of utility computing regained attention and achieved new developments. IBM's Global Services department defined utility computing as: "Utility computing is the use of infrastructure, applications, and business processes over the Internet on a pay-per-use basis in a secure, shared, scalable, and standards-based computing environment. Users can access IT resources as easily as they currently use electricity and water, and pay for them based on usage [2]."

The service-oriented utility computing (SOUC) architecture is illustrated in Figure 1 [Figure 1: see original paper]. The architecture comprises three main components: a service gateway, resource management, and a shared resource pool. The service gateway functions similarly to a service distribution pipeline. Resource management serves as the core module of the entire architecture, capable of managing resources while monitoring and discovering them through push-pull mechanisms provided by virtualization technology. It records current and historical data on all resource allocations and can employ heuristic and predictive resource analysis methods to help the resource management module dynamically adjust resources according to appropriate decisions and timing. These analyses can also be used for performance tuning and capacity planning predictions to ensure optimized service distribution. The shared resource layer integrates computing, storage, network, and data resources into a unified resource pool.

Utility computing represents a resource-sharing, on-demand service application model and represents the future direction of data centers. Many current technologies aim toward this goal. For example, Web Services are application-layer on-demand service technologies for Internet applications, grid computing is an application-layer on-demand service technology for high-performance computing applications, virtual machines are resource-sharing technologies for computer systems, and partitioning is resource-sharing technology for large servers. However, virtualization technology holds particular significance for effectively supporting utility computing at the architectural level.

Through virtualization technology, storage or computing resources from multiple backend computers can be integrated into a unified, manageable resource pool. A virtual infrastructure represents all resources in the entire IT environment, aggregating all X86 computers and associated network and storage resources into a unified resource pool, which can significantly improve resource utilization.

Compared with direct use of physical servers, server virtualization technology offers numerous benefits:

Global Unified Physical Computing Resource Management: Virtualized servers help solve the problem of substantial server resource waste through unified management of global physical resources, enabling us to dynamically construct virtual machines with capabilities that meet application demands without purchasing new servers to satisfy application requirements.

Additionally, virtualization technology effectively addresses the utilization of old servers. Previously, servers required frequent updates and replacements, with old servers being discarded. After introducing virtualization, we can integrate the computing resources of old servers, including heterogeneous devices, with those of new servers, greatly improving resource utilization and saving costs.

Real-Time Computing Resource Flowing: When the most powerful physical servers reach their performance limits and can no longer handle additional parallel requests, we previously had no choice but to purchase more powerful servers to take over their business. With resource flowing technology from the Virtual Machine Manager (VMM), when a virtual machine' s load reaches its limit, we can dynamically invoke more computing resources from the resource pool to enhance its processing capability.

Since virtualization technology integrates all physical server computing resources into a unified whole, eliminating the natural boundaries that originally existed between servers, we can construct extremely powerful virtual servers to meet application demands until all system resources are exhausted—something impossible to achieve with physical servers.

Standardized Virtual Server Images: In practical application scenarios, each server is often designated for a specialized application, forming application servers. Virtualization technology enables standardized operations for each type of application server, establishing independent images for each application server type that are decoupled from the underlying infrastructure. When needing to add such application servers, we simply construct identical virtual machines using the saved images, avoiding the need to install and configure applications individually on physical machines as before.

Convenient Application and Data Backup: In previous PC+server information architectures, comprehensive data backup was a difficult problem to solve, with the sheer number of PCs alone causing headaches for administrators. Using virtualization technology, we store each user' s data on server-side virtual machines. By saving virtual machine images, we can preserve all current user

application data. Even if a user loses their terminal, it will not cause a data disaster.

1.2 Research Background and Significance

In the network-centric computing era, the client/server model has been widely adopted, with numerous services and applications implemented through Web services. Currently, Web service traffic accounts for 60% of Internet traffic. Figure 1.3 shows the growth in the number of Internet-connected sites from 1995 to 2009 [5], demonstrating an even more rapid growth trend than before.

The rapid development of the Internet poses enormous challenges to network bandwidth and servers. Historical development shows that network bandwidth growth far exceeds processor speed and memory access speed growth, with technologies such as 100M Ethernet, Asynchronous Transfer Mode (ATM), Gigabit Ethernet, and soon 10 Gigabit Ethernet emerging successively. Dense Wavelength Division Multiplexing (DWDM) will become the mainstream technology for broadband IP on backbone networks [6], with Lucent having introduced the WaveStar OLS 800G product that transmits 800Gbit/s on a single fiber [7]. Therefore, in the coming period, bottlenecks will increasingly appear on the server side rather than in network bandwidth.

Furthermore, network service access demands are constantly changing dynamically. Popular sites attract unprecedented access traffic. For example, during the Olympics, the official website' s daily visits exceeded 183 million. Some network services receive massive traffic volumes, such as American Online' s (AOL) Web caching system, which processes 5.02 billion user Web requests daily, with an average response length of 5.5K bytes per request. Meanwhile, many network services become overwhelmed by explosive growth in access volume, unable to process user requests promptly, resulting in long user wait times and significantly degraded service quality.

Most importantly, the majority of commercial websites require 24-hour, 7-day service, particularly for e-commerce sites, where any service interruption or critical data loss causes direct business losses. This imposes increasingly high reliability requirements on network services.

Network service application requests exhibit temporal and spatial imbalance, with peak loads reaching 8 to 10 times the average load, while the emergence of hot events or news is largely unpredictable. How to establish and deploy scalable, highly reliable network services in server clusters with limited processing capacity to meet dynamic and growing load demands has become an urgent problem.

To support these network application requirements, we need to establish a scalable, highly available, and reliable Web service environment. The typical solution for Internet applications is to run them in a server cluster, commonly known as a data center. In recent years, with the emergence of virtualization

technology and changing service demands, enterprise data center infrastructure is undergoing significant transformation to meet evolving requirements.

Traditional enterprise data center design has primarily followed an exclusive architecture approach. Each application possesses its own dedicated servers, storage devices, and network infrastructure, with the application's software stack globally controlling all these resources. However, application-exclusive data centers often remain idle most of the time due to over-provisioning, resulting in severe resource waste.

Figures 3(a) and (b) [8] show processor usage in two nodes of an enterprise data center over one week. Each node has six processors, and we can observe that processor usage remains below 10% for most of the time, while maximum processor usage far exceeds average usage. Similar issues can be observed in the utilization of other resources (disk, network, and memory). Therefore, if we provision resources based on maximum or average values, data centers will suffer from either low global utilization or degraded service performance under peak loads due to insufficient resources.

Modern enterprise data center design is shifting toward the utility computing model. Under this model, all hardware resources are integrated into a shared architecture for use by various applications, which acquire resources on demand over time. In this shared, consolidated environment, meeting SLA targets for diverse application resource requirements becomes the most critical challenge. In the two-node data center scenario shown in Figure 3(c), at any given moment, only the resources of one server node are needed to meet the total demand of both node applications. If server resources can be allocated dynamically according to the changing demands of the two applications, the two nodes can easily be reduced to one.

Figure 4 [Figure 4: see original paper] illustrates how virtualization technology introduces new dimensions to resource management. Figure 4(a) shows traditional single-dimensional resource management, where load balancing and request distribution optimization are achieved by coordinating available resources across numerous physical hosts. Figure 4(b) depicts a virtual machine-based resource management scenario. In this new context, we can not only employ traditional approaches for request distribution management but also utilize virtualization technology interfaces to allocate resources from the physical machine cluster resource pool among various virtual machines, completing a second dimension of resource management.

Currently, making resources flow on demand according to dynamic application requirements in virtual environments has become a hot research topic. Related work includes HP' s SoftUDC [40], IBM' s PLM [41], and VMware' s DRS [42] resource dynamic management systems. Our capability service research group has also conducted extensive research in this area: reference [10] proposed a resource flowing framework, [11] studied local resource flowing algorithms, and [12] designed a multi-level resource scheduling mechanism. However, current

virtual machine resource scheduling technologies can only solve resource allocation within local nodes and cannot address the matching problem between resource flowing and load. In the work of [12], we simply modified the Round Robin Scheduling algorithm by removing virtual machines with processor usage exceeding 95% from the scheduling list, while continuing round-robin request distribution among the remaining virtual machines. Using TPC-W as a benchmark, performance improvements shown in Table 1 were achieved. The simple round-robin scheduling algorithm does not consider predictions of load resource requirements or awareness of virtual machine resource flowing changes. Since database requests consist of long and short transactions, simple round-robin distribution inevitably causes scheduling imbalance, leading to mismatches between resource flowing and application load. When virtual machine capabilities expand, loads are not correspondingly allocated.

Because resource flowing is adaptive, dynamic, and transparent, request loads cannot be predicted. To change this situation, the scheduler's request scheduling layer must predict load distribution, capture the dynamic scaling of virtual Web service clusters, and then make correct scheduling decisions to achieve matching between resource flowing and application load.

Therefore, it is necessary to study resource flowing prediction mechanisms in utility computing environments. By predicting the performance requirements of arriving requests and the capabilities of dynamically scaling virtual servers, we can achieve dynamic load-balanced request distribution based on feedback system state information to guide and control the scheduling management work of service request schedulers under virtual resource dynamic scaling scenarios. This paper focuses on how to independently leverage the management advantages of the request scheduling dimension and solve its mismatch with resource flowing, thereby comprehensively and uniformly managing and fully utilizing all underlying resource pools. This enables utility computing platforms to improve system throughput and guarantee service quality while maximizing resource utilization.

2. Related Work

Traditional load balancing strategies for Web physical cluster systems represent a classic research topic. Although balancing scheduling strategies face new challenges in resource-elastic virtual Web cluster environments, the core ideas remain consistent. Two main issues exist: predicting arriving request loads and their resource requirements, and obtaining system state information to evaluate servers' dynamic comprehensive load capabilities. Only by accurately acquiring these two types of information can requests be effectively scheduled in the system.

Over the past decade, academia and industry have shown increasing interest in cluster-based Web services and conducted substantial research [47]. Previous work has demonstrated that managing these systems requires numerous

algorithms, including load balancing and load sharing [73,60,53,52,67], architecture design and performance optimization [70,64,60,37], overload prevention and access control [71,44,53,45,51,59,69,61,62], and request distribution and connection redirection [70,64,60,37,58]. However, there is almost no published literature on dynamic load balancing scheduling and access control for network services in resource-elastic virtual cluster environments. This paper considers locally distributed virtual cluster network server scenarios without cross-geographical cluster access latency issues. A comprehensive survey of distributed network servers has been well documented in [70]. The following sections briefly introduce relevant literature.

2.1 Web Cluster Architecture

A Web cluster refers to a network service site where two or more servers are networked together to jointly process user requests. Although large-scale clusters possess numerous server nodes, a single host provides a unique access interface for all users. To control all requests arriving at the site and mask the backend distributed server structure, Web clusters provide a single virtual IP address associated with backend actual servers—this is the network scheduler. Figure 5 [Figure 5: see original paper] illustrates the typical three-tier structure of a cluster-based network site, representing the physical deployment of the network application software architecture in Figure 1. All servers in the cluster are placed in the same physical space. Network request flows are distributed to various network servers through network switches. Web caches directly serve requests for small static files. In the first tier, some network servers provide a network interface for clients. Typical backend services such as application processing and database servers assist network servers in providing dynamic content. In this design, service nodes connect to network switches and can handle both static and dynamic requests. The network switch receives client service requests and selects one from the network server pool for subsequent processing operations. Deploying an appropriate request distribution algorithm to balance these servers' loads becomes crucial.

Many mature products for distributed network request distribution have emerged in academia and industry, such as: IBM' s TCP router [32], which forwards all HTTP requests arriving at the switch by changing the destination IP address of incoming packets (i.e., converting the IP packet' s destination address to the selected web server' s private address); the LVS project [20], a completely open-source, kernel-level IP packet transformation Linux cluster implementation; Magicrouter [28], a fast packet parsing processor that handles network packets at the user level and changes their IP addresses and checksum fields; Cisco' s LocalDirector, which rewrites IP header information for each arriving packet while maintaining a table to map each session to a server [31]; and IBM' s Network Dispatcher, which differs from TCP router by not requiring packet address changes since packet forwarding is handled at the Media Access Controller (MAC) address level [33].

2.2 Load Balancing Algorithms

Research on dynamic request distribution methods in Web cluster environments primarily targets several objectives: Quality of Service (QoS) guarantees, performance isolation, and maximizing resource utilization. This can be abstracted as an online optimization problem that periodically monitors resource usage and allocates loads according to request resource requirements [3]. In large cluster environments supporting numerous services, to save energy consumption overhead, resource provisioning can also be modeled as a bound processing procedure [13]. Consequently, a service' s active working set size dynamically adapts to given processing capabilities. In [14], the authors propose an integrated framework that combines cluster-level load balancers with node-level schedulers to achieve overall system efficiency and guarantee service response time targets. The work in [15] abstracts resource allocation as a two-dimensional packing problem, dynamically changing application instance deployment locations as resource requirements change.

Many scheduling algorithms [69,64,52,67] have been proposed to solve balancing problems in network cluster servers. According to the generalized Open System Interconnect (OSI) reference model network protocol stack layers, these algorithms can be simply categorized into Layer 4 (network layer) and Layer 7 (presentation layer) [70] scheduling.

A Layer 4 (content-agnostic) algorithm selects a target server when a client establishes a TCP connection, after which all HTTP requests on this connection are sent to that server for execution. Global scheduling algorithms operating at Layer 4 can be further divided into static algorithms (e.g., random, round-robin) and dynamic algorithms that consider network client information (e.g., client IP address, TCP port) or server state information (e.g., number of active connections, least-loaded server), or both.

On the other hand, a Layer 7 (content-aware) algorithm can establish a complete TCP connection with a client. It can predict HTTP request content before making scheduling decisions and benefit from session-layer information such as session identification, URL size, URL type, and Cookies, in addition to Layer 4 information. In this manner, scheduling algorithms can deploy content-aware request distribution. Layer 7 scheduling algorithms can also be broadly categorized into static and dynamic algorithms [52]. Static algorithms do not consider any load state information, while dynamic algorithms employ mechanisms to monitor and evaluate server load states and request types, making them more effective but also more complex than their static counterparts.

Ongoing research continues to advance load balancing algorithms. A simple and commonly used Layer 4 load balancing algorithm is Weighted Round Robin (WRR). This algorithm' s core involves dynamically adjusting weights for each server in the cluster, making these weights proportional to server load states [62,52]. Many commercial products, such as Foundry Network' s Server Iron XL [55], Cisco switches with Local Director functionality [48], IBM' s WebSphere

Edge Server [57], and BEA' s WebLogic Cluster [46], use weighted round-robin algorithms. Additionally, the work in [54] proposes a weighted round-robin algorithm implemented using optimized distribution mechanisms that can adaptively evaluate each server' s load state.

The LARD [38] algorithm is a well-known Layer 7 distribution algorithm whose goal is improving web service node cache hit rates. LARD' s main principle is scheduling all access requests to the same network target to the same web server for execution. The LARD algorithm works well on traditional cluster sites serving static pages. Another Layer 7 load balancing scheduling algorithm [71,70,72] statically divides requests into different categories and sends requests of the same category to specific servers for execution. Although effective from system management and high cache hit rate perspectives, this algorithm ultimately results in very low server utilization because some server resources remain unused and servers cannot be shared by all clients.

The ADAPTLOAD algorithm proposed in [67] classifies long and short transactions based on request size and schedules them to specific servers for separate execution to achieve shortest job execution times. This strategy is based on empirical load resource requirement distribution information, such as request size and frequency, and dynamically adjusts these parameters based on real-time load monitoring. However, this method exhibits relatively poor load balancing for dynamic requests [67]. Multi-class Round Robin (MC-RR) or Client-Aware Policy (CAP) [63,52] are also commonly used Layer 7 algorithms. CAP' s main idea is that although network switches cannot accurately evaluate request service times, they can classify requests from URLs and assess their server resource requirements. CAP' s biggest drawback is its lack of feedback on server load state information.

The key point of the above methods is request classification and configuring different categories according to various classifications. The advantage of Layer 7 network scheduling lies in its ability to classify network content and place it on specific service nodes in heterogeneous cluster servers to achieve higher cache hit rates, as in the LARD [38] strategy. On the other hand, Layer 7 scheduling introduces additional overhead on switches, creating system bottlenecks. To overcome this disadvantage, some propose distributing requests based on content awareness to design scalable network service systems [29,34]. These structured solutions focus on optimizing scheduling algorithm architecture deployment, which exceeds the scope of this paper' s research.

2.3 Network Service Load Description

Almost all load balancing scheduling strategies evaluate server node load states based on periodic sampling. Therefore, the accuracy of server load state evaluation is an important issue. Many load balancing algorithms collect instantaneous server load information, such as processor load, disk usage, and active network connection counts, as load descriptors [71,70,68,72,69,49,65,73]. State

information collected periodically by load monitors experiences intense fluctuations during some peak load variation periods but remains relatively stable most of the time, and historical load distribution data can help predict future states.

Previous work [50,66] used linear models to evaluate actual loads through measured resource information. Such models work well when measurement values are highly correlated. However, when network servers are under high traffic loads, the model' s evaluation effectiveness becomes poor. Many more mature methods exist but each has limitations. For example, ARIMA [66] requires extensive training periods to calculate parameters, making it unsuitable for real-time load balancing scheduling.

2.4 Performance Prediction

Scheduling decisions cannot rely solely on server resource state information; they also require predictions of request arrival distributions and their resource demands. For a long time, the primary methods for load performance model research have been curve fitting and queuing models [36] to predict shared resource requirements for mainframes or large-scale servers. Curve fitting and enterprise demand forecasting methods are used to quantitatively infer application resource demands for various resources. Queuing models may be used to seek the inherent quantitative relationship between expected average response time under general working sets and the goal of maximizing resource utilization [35]. A major limitation of using queuing theory models is their assumption of a stable arrival rate for request distributions and their prediction of only overall average performance. Additionally, queuing theory models cannot resolve load competition conflicts under shared resource conditions. Implementation of these methods requires significant human involvement, thereby increasing processing costs. Moreover, because modeling quantification makes assumptions that deviate substantially from the dynamic changes in actual load distributions, their extensive computations and cumbersome operations are unsuitable for online automated systems. Most organizations must carefully consider their ability to bear the increased overhead when adopting these methods. Furthermore, modern enterprise data centers contain hundreds or thousands of large shared servers. To reduce management overhead, enterprises are competing to upgrade their data center architectures to explore management applications for resource requirements. In addition to these two methods, there are complex offline machine learning methods [43], genetic algorithms [75], content-based classification [76], data mining [74], and time series analysis [16] prediction models. Their common major disadvantage is the need to introduce substantial computational and control overhead. Meanwhile, complex mathematical computation models require idealized assumptions about loads, preventing them from truly reflecting load information, and calculation results will inevitably deviate from actual conditions.

2.5 Access Control

An important goal is enabling network servers to maintain request execution even at maximum load or overload. However, increased load leads to increased response times. If an impatient user aborts request execution, the server processing this request enters a request timeout state. Each request task requires resource allocation for TCP connection operations, packet parsing, and request processing. Even when timeouts occur, these resources are still consumed. This inefficient operation causes web server system throughput to drop dramatically under high load, even maintaining resource utilization at 100%. To prevent this behavior, we need a high-load control mechanism in load balancing scheduling. Methods such as access control [71], special scheduling strategies [62], or comprehensive approaches combining both have been proposed to solve this problem. Simple access control methods discard requests directly without considering their resource impact. However, different request types have different resource demands, and load balancing scheduling algorithm access control mechanisms must consider these differences.

Previous work has addressed web server access control research. In [69], the authors used classical feedback control theory to propose a performance control mechanism that guarantees Apache web server response times within a desired range. In [45], an adaptive control method was used to bind response times in three-tier architecture network sites. In [39], a QoS-aware load balancing mechanism used stochastic high-level Petri nets to model and simulate cluster network servers. In [56], a control theory-based load balancing algorithm was proposed for deploying user session migration on cluster network servers. In [44], the authors proposed a kernel-level solution that periodically monitors socket receive queues and processor run queues to maintain QoS requirements. This method can adapt to load changes and adjust resource allocation on loads.

3. Network Service Performance Prediction Model

In the capability service computing architecture, we primarily study concurrent execution control of web services, database services, and typical office applications within this shared management framework. Therefore, our service performance model analyzes the application characteristics and request distributions of these three service types through extensive experiments, attempting to abstract the quantitative relationship between performance requirements of these three typical enterprise applications and request quantities and required resources (this paper only concerns processors and memory, though similar work can be extended to other resources). This model is ultimately applied to service request distribution scheduling decisions and integrated as a functional module into the capability service computing framework to coordinate the Distributed Virtual Machine Monitor (DVMM) for dynamic adaptive management of large-scale heterogeneous services running on the framework.

In reference [12], we first proposed this performance model and applied it to

resource demand prediction models. The model' s key input is the dynamic number of active connections, monitored at the request scheduling layer. To maintain independence between traditional management and virtual resource management dimensions, this model is used by the request scheduling module. Experimental results demonstrate the model' s effectiveness.

3.2 Database Service Performance Model

To test database service performance, we used TPC-W [19] as the workload, where Emulated Browsers (EBs) simulate users sending and receiving HTML content using HTTP and TCP/IP protocols on the network. Each EB submits a series of requests within a user session. The emulated user load is "decision-generated" by a customer behavior model specifying browsing patterns, the number of emulated users, and think time (the interval between receiving the last byte of a result page and submitting the first byte of the next request).

The resulting relationship between R and D is shown in equations (3) and (4).

3.3 Office Application Service Performance Model

We used the desktop office benchmark Xnee, a Linux tool for recording and playing back screen operation events on X windows. We created 12 typical office application operation user behaviors based on different application behaviors, using a session to represent a user' s operation trace over a period. According to actual office application pattern patterns collected in reference [36], we adjusted these 12 user behaviors to conform to corresponding application scales for work.

By measuring average response times for several sessions under different processor and memory resource configurations, we can similarly obtain the office application performance relationship, as shown in formulas (5) and (6).

4. Network Service Request Distribution

This paper' s main work focuses on the service scheduling management layer of the capability service computing framework. Based on network service characteristics, we designed a network service request distribution control system as shown in Figure 4.1. The service scheduling manager receives load requests, then uses the performance model to calculate and predict the most reasonable scheduling decision based on performance objectives and system resource information fed back by the online monitor, and finally performs dynamic request distribution. The global control correction module' s function is to analyze whether the performance model' s predictive decisions are correct based on performance data fed back by the monitor combined with historical data. If scheduling imbalance occurs due to prediction errors, the global controller feeds back a deviation value, and the service scheduling manager dynamically modifies each application server' s weight based on this correction value, enabling

the scheduling layer to distribute requests evenly according to server working capacity and load.

4.1 Service Request Scheduling

The service request scheduling management layer handles scheduling relationships among multiple services and numerous virtual machines. User service requests are first obtained by the management node VMScheduler, which then schedules service requests to a specific virtual server for execution according to certain scheduling algorithms based on service type and each application server' s load situation—this is the dynamic distribution of service requests.

For request scheduling, we adopted the LVS (Linux Virtual Server) [20] architecture for network application load distribution and load balancing management. By modifying the kernel' s weighted least-connection scheduling algorithm, we implemented a Dynamic Feedback Balancing Scheduler (DFBS) that can cooperate with resource flowing to enable dynamic adaptive adjustment of large-scale concurrent heterogeneous services across the entire system.

Figure 10 shows the LVS architecture, which mainly includes three functional parts: load balancers for job scheduling, service clusters equivalent to virtual machine-based server images, and backend shared resources that can be managed and integrated using system-level virtual machines. The clear functional division and hierarchical structure make deploying LVS into the utility computing framework very natural, with relatively easy system construction, maintenance, and performance scaling.

4.2 Algorithm Design Principles

Since our request distribution scheduling algorithm is applied in a dynamic adaptive real-time decision module, its design must comply with certain real-time system constraints, mainly in three aspects:

1. Fairness: Due to dynamic flowing of virtual resources, the available resources of virtual hosts managed by the scheduler are dynamically elastic. During scheduling decisions, these available resource changes must be monitored, and a quantitative metric must calculate each virtual server' s comprehensive load. Only by dynamically adjusting request distribution based on comprehensive load can scheduling achieve fairness.

2. Timeliness: General network applications submit thousands of session requests to servers per second on average, requiring decision time to be as short as possible during request distribution. Otherwise, the entire request service response time will be lengthened due to decision delays, degrading overall service performance and user experience. Therefore, we adopt offline performance prediction model learning methods to shorten decision time.

3. Effectiveness: The ultimate scheduling goal is to enable large-scale heterogeneous services running on the entire utility platform to share resources in

isolation, provide normal service functions for users, and improve system resource utilization. Request scheduling must effectively achieve this scheduling objective.

4.3 Scheduling Algorithm Design

When clients access the network through TCP connections, the time required and computing resources consumed vary greatly depending 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, heterogeneous server processing capabilities are in a non-uniform state, and their processing times for the same TCP connection access differ. Particularly in virtual machine cluster server environments, virtual host resources are dynamically elastic and constantly changing.

The vast differences in request processing times and non-uniformity of heterogeneous server processing capabilities may lead to skewed server utilization, where some servers become overloaded with long request queues, continuously receiving new requests, while other servers remain essentially idle. This results in long customer wait times, poor experienced system service quality, and underutilized system capacity.

Actual TCP/IP traffic distribution is wave-shaped, with small and large flows occurring periodically. Web access traffic also exhibits self-similarity, requiring a dynamic feedback mechanism that utilizes server group states to address the self-similarity of access flows.

Based on the design principles introduced in the previous section, we designed the adaptive Dynamic Feedback Balancing Scheduler (DFBS). The algorithm considers servers' real-time loads and actual system response conditions, continuously adjusting the proportion of requests processed among servers to prevent some servers from receiving large numbers of requests when overloaded, thereby improving overall system throughput to ensure fairness and effectiveness.

Figure 11 [77] shows the algorithm' s working environment: a monitor daemon runs on the load scheduler. The system initially uses the weighted least-connection scheduling algorithm in the kernel and assigns initial weights to the virtual machine cluster. Simultaneously, the monitor daemon periodically collects load information from each server to learn about dynamic scaling of virtual resources. This resource load information can calculate a server' s load metric to evaluate current server load levels, i.e., available resource quantities. The IP Virtual Server (IPVS) also periodically returns connection count values to the monitor daemon. Based on this count, we can calculate request distribution predictions across virtual machine nodes as input metrics for request load and use performance models to evaluate whether current connection counts are overloaded. If overload is estimated, the virtual machine' s weight is set to zero;

otherwise, the comprehensive load value is calculated based on previously calculated server load and request load. The monitor daemon calculates a new set of weights from each server' s comprehensive load value and current weight. If the difference between new weights and current weights exceeds a set threshold, the monitor daemon sets the server' s weight in the kernel' s IPVS weighted least-connection scheduling. The algorithm' s main execution flow is shown in Figure 12.

4.4 Summary

This chapter utilizes the Linux Virtual Server, a kernel-level load balancing tool for Linux, adopts its kernel-level implemented weighted least-connection scheduling algorithm, and references the dynamic feedback ideas of Professor Zhang Wensong from National University of Defense Technology [77] to design and implement a dynamic feedback load balancing scheduling algorithm for network service requests under virtual resource dynamic scaling systems. The algorithm collects comprehensive load state parameters such as virtual resource and service request load distribution, calculates server weights, and applies them to kernel algorithm modifications to dynamically and adaptively solve the matching problem between resource flowing and application load.

We chose the Linux Virtual Server tool not only for its network service request load balancing distribution function but also for recognizing the many advantages of this dynamic scaling approach in system reliability, scalability, and availability. Integrating it into our capability service computing framework not only compensates for resource flowing deficiencies but also provides significant flexibility in service deployment and system control, greatly saving overhead for virtual environment management and maintenance. Our research shows that adding a service request scheduling module to the capability service computing framework can combine well with the resource flowing module, achieving comprehensive management of the entire virtual system from local to global dimensions. This also represents a cloud computing implementation method proposed by our group, providing a solution for enterprise data centers facing problems such as low resource utilization, difficult deployment, low availability, and high management costs.

5. Application of Capability Service Computing Framework

Our research group has implemented a resource flowing management system named RAINBOW—a trustworthy, reliable, and efficient virtual computing environment for the Internet. RAINBOW has already implemented on-demand resource flowing algorithms for virtual computing environments. By adding scheduling, monitoring, and control modules to the RAINBOW system, we implemented the DFBS scheduling algorithm, forming a complete adaptive dynamic scheduling system as a practical application implementation of this paper'

s research work.

5.1 Basic Virtualization Architecture for Capability Services

In our system, to achieve virtualization, computing resources are divided into two layers: the application scheduling management layer and the computing resource management layer, as shown in Figure 13.

The computing resource management layer' s function is to use the Distributed Virtual Machine Monitor (DVMM) to integrate physical server resources, unify management of all physical computing resources, and build a virtual computing platform on this foundation to run a series of flexibly constructed virtual machines. The application scheduling management layer' s function is to monitor current loads of virtual machines in the system and perform real-time scheduling and actual execution of applications based on virtual server loads when users issue service requests, achieving system-wide load balancing and improving utilization.

Above the computing resource management layer is the application management layer. Since computing resources have been virtualized, users need not concern themselves with underlying details such as application configuration and execution location during application use. In application virtualization, after each client submits a request, the request' s logical execution is handed over to the lower-layer computing resource virtualization. In computing resource virtualization, applications are flexibly scheduled to execute on a virtual server in the computing capability pool, which is built upon the foundation of DVMM' s management of physical hardware resources.

5.2 Service Scheduling Management Layer

Using LVS' s virtual IP address translation, we separated the actual execution servers of network applications. As shown in Figure 14, the scheduling module that directly receives user-sent requests is only responsible for processing service request scheduling, while specific execution runs within a virtual machine in the real-time allocated computing capability pool.

The functions of each module in the application scheduling management layer structure are as follows:

Global Controller: The global controller manages all physical information in the system, including all managed physical nodes, virtual machines created on each physical node, and resource allocation and usage for each machine. Through this management information, the system uses the performance model introduced in this paper to calculate the maximum load capacity of large numbers of heterogeneous services on the system and returns results to scheduling and local management modules for request and resource allocation.

The global controller also receives service requests proposed by Web services. After receiving application scheduling requests, it sends scheduling inquiry requests

to the corresponding scheduling modules and passes corresponding virtual server information to them. The scheduling module deploys virtual server configurations using this received information, adding corresponding virtual servers to the scheduling list. The global controller periodically requests management information from local management modules of corresponding virtual servers, requiring each node to return server management information including virtual host IP, virtual host name, and physical node location. Simultaneously, it passes global resource allocation calculation results to local management modules' activity parameters to guide intra-node resource flowing decisions. The global controller also frequently interacts with scheduling modules, dynamically updating virtual server information for specific services on one hand, and feeding back maximum supported connection number predictions to scheduling modules to modify scheduling weights on the other.

The global controller also uses performance model predictions to optimize inter-service request scheduling from a macro perspective and optimize application-layer scheduling to achieve admission control for ensuring normal system operation under high-load states, thereby further optimizing the matching between system request scheduling and resource flowing and improving system stability and availability.

Scheduler: The scheduler module implements the DFBS scheduling algorithm and is responsible for processing Web service request scheduling. The scheduler itself cannot directly obtain virtual servers' current load information, which is acquired through Gmond. After obtaining current load conditions of all virtual servers, the scheduler determines an appropriate server as the scheduling result based on the DFBS balancing algorithm and service request types proposed by the global controller. As shown in Figure 14, the scheduler connects to Gmetad, which collects data and sends it to the scheduler for calculation and weight adjustment based on servers' current loads. This weight value is proportional to the number of allocated connections—the larger the weight, the more connections allocated. Weight calculation considers both virtual server processor capabilities and current available resource sizes.

Ganglia: Ganglia is a distributed system performance monitoring system, a Linux-based graphical software for monitoring system performance, including processor, memory, disk utilization, read/write loads, and network traffic. Ganglia has two daemons: Gmond (client) and Gmetad (server). Gmetad collects processor and memory load changes on distributed virtual servers and returns real-time system-wide data on processor and memory loads when the scheduler requests virtual server cluster load information. Gmond and Gmetad work together to monitor current loads on virtual servers at the client side.

Local Manager: The local manager module manages individual physical nodes, primarily maintaining capability allocation information management among virtual machines running on physical nodes. The local manager collects virtual resource utilization data from local sources and obtains current resource quotas for each virtual server, i.e., dynamic decision results of resource flowing. Most

importantly, it can collect dynamic parameters of resource flowing algorithms, including flowing thresholds and activity weights. With this information, the scheduler can timely terminate request allocation to service nodes approaching flowing thresholds, avoiding system overhead from unnecessary resource flowing.

6.1 Experimental Design

Experimental Environment:
Computing platform: 5 physical servers
- 2.0GHz, 2×4-core AMD Opteron (4 cores per CPU)
- 2GB cache
- 8GB RAM
- 1000Mb Ethernet

Client server array: 9 servers and 1 laptop
Network environment: Gigabit Ethernet LAN with server platform and client array in the same LAN
Operating system: CentOS 4.4
Xen version: 3.0.4
Web server: Apache
Database: MySQL
Desktop office software: OpenOffice, Firefox, Adobe Reader, etc.

Experimental Test Benchmarks:
1. httperf [18] for web service performance testing
2. TPC-W [19] for database service performance testing
3. Xnee for office application performance testing

Experimental Deployment:
Seven virtual machines were created on each physical server, each running one type of test benchmark: three as Apache servers, three as database servers, and one as a desktop office application server. Each virtual machine was initially allocated 700MB memory resources (4.9GB total), with remaining memory allocated to domain0. Each virtual machine was configured with one virtual CPU (VCPU), with 7 VCPUs from the 7 virtual machines bound to 4 physical cores. To minimize virtualization' s impact on system performance, the other 4 cores were bound to domain0. Thus, the two 4-core Opteron processors were configured with one shared by domainU and the other exclusively used by domain0. We adopted Xen' s default Credit-based CPU scheduling algorithm. Initial priority settings for the three services were 1:1:1:1:1:1:1. At the service request scheduling layer, the resource monitor was set to collect virtual machine processor resource usage information every second and memory free information every five seconds. The global weight adjustment decision monitor' s historical information interval was 15 minutes. The resource layer started the dynamic resource flowing algorithm [11] to enable virtual machine resource dynamic scaling. For each virtual machine, memory overload threshold was set at 50MB remaining free memory, processor overload threshold at 95% usage, and desired

processor usage at 70%. Local flowing algorithm collection cycles were 1 second for processor and 3 seconds for memory, while the global flowing algorithm collection cycle was 30 seconds. Specific resource flowing algorithm details can be found in [36]. The request scheduling algorithm treats the resource layer as a resource-elastic virtual machine cluster system, viewing the resource flowing layer as a black box.

Performance Parameter Definitions:
1. Web service performance metric uses average response time
2. TPC-W service performance metric uses average WIPS (Web Interactions Per Second)
3. Office application performance metric uses application response time, defined as the sum of application startup time and operation execution time

6.2 Experimental Results and Analysis

Table 2 compares three scenarios: DFBS scheduling without resource flowing, round-robin scheduling with resource flowing, and DFBS scheduling with resource flowing, showing performance and resource utilization improvements relative to round-robin scheduling without resource flowing.

The first row shows DFBS scheduling without resource flowing, which mainly achieves resource allocation through load balancing scheduling. Due to long and short jobs in databases, load scheduling yields significant performance improvements of 4.2%. Desktop office services show substantial response time improvements during application startup and shutdown when resources are abundant. DFBS can fully perceive actual virtual machine available resources, scheduling requests to nodes with minimum comprehensive load and maximum available resources, achieving 23.8% performance improvement over round-robin scheduling. Web services show only 1% improvement because each request is relatively uniform and servers are homogeneous, making round-robin scheduling work well.

The second row shows performance improvements from resource flowing when using round-robin scheduling. In experiments, system parameters were adjusted near service saturation points to create resource competition. Resource flowing allocates resources on demand according to service priorities. Web services have low priority, so while improving the other two services, they can only minimize Web performance degradation. The results show that resource flowing provides substantial performance improvement potential.

The third row shows performance improvements from resource flowing combined with dynamic feedback balancing scheduling. Compared with the second row, comprehensive management of the entire service system resources across two dimensions enables more reasonable request scheduling and higher utilization efficiency. Like the second row, due to resource competition, while improving high-priority TPC-W and desktop office performance, low-priority Web service improvements are much smaller.

TPC-W has the highest priority and is allocated the most resources under the same conditions. Since its request load long/short job distribution is more typical, we primarily evaluate our DFBS scheduling algorithm effectiveness through its performance results. For comparison, we conducted a separate test providing TPC-W with sufficient resources under the same load, measuring a maximum possible improvement of 12%. As shown in Table 2, DFBS scheduling combined with resource flowing can achieve 86.7% of this performance improvement upper bound (10.4/12×100%).

The table shows relatively small resource utilization improvements, primarily because experimental parameters were set near load critical points, with total resource utilization already exceeding 80% before scheduling and flowing. Therefore, the listed resource utilization improvements do not represent improvements from service consolidation but merely additional resource 挖掘 under already high utilization through request scheduling and resource flowing. This paper does not address the impact of service consolidation on resource utilization improvements and will not elaborate further.

Experimental results fully demonstrate that under limited total available resources, to support large-scale concurrent services, the service request scheduling module proposed in this paper can achieve matching with the resource flowing management module. It can ensure performance improvements for high-priority services while keeping performance degradation of low-priority services within acceptable levels, guaranteeing overall service availability. Simultaneously, the service scheduling module effectively compensates for resource flowing deficiencies, expanding performance improvement space beyond what resource flowing alone provides, achieving comprehensive system management from intra-node to inter-node dimensions with significant results.

7. Conclusion

Current information system architectures composed of large numbers of PCs and a few servers still face many problems, such as substantial computing resource waste, high total management costs, incompatibility among different system platforms and applications, and high probability and losses from data disasters. To solve these problems, we propose replacing traditional PC-server models with capability service computing models. The capability service model uses virtualization technology to integrate numerous network services into a shared framework to improve resource utilization. In this shared data center, we implement a multi-level resource flowing architecture that automatically allocates resources on demand among virtual machines providing different services. This architecture models resources through optimization theory and proposes local and global resource flowing algorithms based on the model. These algorithms can ensure higher-priority service performance while minimizing performance degradation of lower-priority services when resource competition occurs. However, in practical applications, we found that current virtual machine resource scheduling technologies can only solve resource allocation within local nodes, and

resource flowing causes dynamic scaling of virtual machine resources. If service request scheduling cannot perceive this dynamic change, resource scheduling management cannot match resource flowing with application load.

As Academician Li Guojie stated [78], integrating the entire Internet into a giant supercomputer to achieve comprehensive sharing of computing resources, storage resources, data resources, and information resources will be the application model of the next-generation network. This model coincides with the cloud computing model. The capability service computing virtualization architecture we propose represents a concrete implementation of Academician Li Guojie' s academic thinking. Currently, we cannot yet expand the computing domain to the entire Internet but have conducted simple preliminary small-scale research within local area networks. Through resource flowing provided by virtualization technology and service request distribution scheduling, we have achieved comprehensive sharing of computing resources, storage resources, and data resources, establishing an implementation example of cloud computing. If computing resource and storage resource virtualization technologies are our power plants, then services deployed on them are the generated electricity. Users using this system are as convenient as using electricity—a mouse click sends a request for execution and returns results. Users need not know where requests are executed, need not perform tedious configuration and installation, nor need to endlessly upgrade their own equipment.

The capability service computing framework is one implementation of cloud computing. This paper' s work explores service request scheduling prediction and algorithms under the capability service computing framework, attempting to solve the matching problem between load distribution space and resource flowing space, hoping to obtain optimal management solutions from the global resource management domain.

References

[1] Michael Wehner, Leonid Oliker, John Shalf. "Towards Ultra-High Resolution Models of Climate and Weather" , International Journal of High Performance Computing Applications, 2008, Vol. 22(2)

[2] M. A. Rappa. IBM Systems Journal, Volume 43, Issue 1, 2004: 32-42

[3] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In Proceedings of the international conference on Measurement and modeling of computer systems (ACM SIGMETRICS), 2000: 90-101

[4] Oracle9i. http://www.oracle.com/technology/products/oracle9i

[5] http://news.netcraft.com/

[6] Lucent Technologies. Web ProForum tutorial: DWDM. October 1999, http://www.webproforum.com/acrobat/dwdm.pdf

[7] Lucent Technologies. Lucent Technologies announces record-breaking 320-channel optical networking system. April 2000

[8] P. Padala, X. Zhu, etc., "Adaptive Control of Virtualized Resources in Utility Computing Environments" , EuroSys' 07: 289-302

[9] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak. A capacity management service for resource pools. In Proceedings of the 5th International Workshop on Software and Performance (WOSP 2005) (Palma, Spain, July 2005): 229–237

[10] 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

[11] 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), Dec. 2008

[12] Ying Song, Hui Wang, Yaqiong Li, Binquan Feng, Yuzhong Sun, "Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center" , CCGrid 2009

[13] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), October 2001

[14] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based internet services. ACM SIGOPS Operating Systems review, 2002, 36(SI):225–238

[15] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In Proceedings of the 15th International conference on World Wide Web, May 2006: 595-604

[16] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In Proceedings of the Eleventh IEEE/ACM International Workshop on Quality of Service (IWQoS)

[17] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, L.J. Dennis Gannon, K. Kennedy, C. Kesselman, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development," International Journal of High-performance Computing Applications, 15(4), Winter 2001

[18] HP, http://www.hpl.hp.com/research/linux/httperf/

[19] H.W. Cain, R. Rajwar, M. Marden, etc., "An architectural evaluation of java TPC-W" , HPCA, 2001

[20] Wensong Zhang, Wenzhuo Zhang. Linux Virtual Server Clusters: Build highly-scalable and highly-available network services at low cost. Linux Magazine, November 2003

[21] Saeed Sharifian, Seyed A. Motamedi, Mohammad K. Akbari, A content-based load balancing algorithm with admission control for cluster web servers. Future Generation Computer Systems, vol.24, Issue 18, October 2008

[22] R.T. Fielding, G. Kaiser, "The Apache HTTP Server Project" , IEEE Internet Computing, vol.1, no.4, July 1997: 88-90

[23] Specweb2005, http://www.spec/org/web2005/

[24] P. Dubois, MySQL, NewRiders, ISBN 0735709211, Dec 1999

[25] http://tomcat.apache.org/

[26] J. Wang, Y. Sun, J. Fan, "Analysis on Resource Utilization Patterns of Office Computer" , The IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), 2005: 626-631

[27] H. Sandklef, "Testing applications with Xnee" , Linux Journal, vol.2004, no.117, Jan 2004: 5

[28] E. Anderson, D. Patterson, E. Brewer, The Magicrouter, an application of fast packet interposing" , unpublished Tech. Rep., Computer Science Department, University of Berkeley, May 1996

[29] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, Scalable content-aware request distribution in cluster-based network servers" , Proc. USENIX 2000, San Diego, CA, June 2000

[30] L. Aversa, A. Bestavros, Load balancing a cluster of Web servers using Distributed Packet rewriting" , Proc. Of IEEE IPCCC' 2000, Phoenix, AZ, February 2000

[31] Cisco' s LocalDirector, in www.cisco.com

[32] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, A scalable and highly available Web server" , Proc. of 41st IEEE Comp. Society Int. Conf., Feb. 1996

[33] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, R. Mukherjee, Network Web switch: A connection router for scalable Internet services" , Proc. of 7th Int. World Wide Web Conf., Brisbane, Australia, April 1998

[34] J. Song, E. Levy-Abegnoli, A. Iyengar, D. Dias, Design alternatives for scalable Web server accelerators" , Proc. 2000 IEEE Int. Symp. On Perf. Analysis of Systems and Software, Austin, TX, Apr. 2000

[35] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, Asser Tantawi, An analytical model for multi-tier internet services and its applications, ACM SIGMETRICS Performance Evaluation Review, v.33 n.1, June 2005

[36] C.S. Yang, M.Y. Luo, A content placement and management system for distributed Web-server systems" , Proc. of IEEE 20th Int. Conf. on Distributed Computing Systems, Taipei, Taiwan, Apr. 2000

[37] V. Cardellini, M. Colajanni, P. Yu, Request redirection algorithms for distributed web systems, IEEE Transactions on Parallel and Distributed Systems 14 (5)(2003)

[38] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, Locality-aware request distribution in cluster-based network servers, in: Proc. of 8th ACM Conf. on Architecture Support for Programming Languages, San Jose, CA, Oct. 1998

[39] Z. Shan, Ch. Lin, D.C. Marinescu, Y. Yang, Modeling and performance analysis of qos-aware load balancing of web-server clusters, Computer Networks 2002, 40 (2): 235–256

[40] M. Kallahalla, M. Uysal, R Swarminatthan etc., "SoftUDC: A Software-Based Data Center for Utility Computing" , IEEE Computer Society, Nov 2004: 38-46

[41] IBM Redbook: "Advanced POWER Virtualization on IBM System p5: Introduction and Configuration" , Jan 2007

[42] VMware Infrastructure: "Resource Management with VMware DRS"

[43] Ahmed A Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, Sunil Kamath, Automatic Virtual Machine Configuration for Database Workloads, International Conference on Management of Data Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008: 953-966

[44] A. Chandra, P. Pradhan, R. Tewari, S. Sahu, P. Shenoy, An observation-based approach towards self-managing web servers, Computer Communications 2006,29 (8, 15): 1174–1188

[45] A. Kamra, V. Misra, E.M. Nahum, Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites, in: 12th IEEE International Workshop on Quality of Service, IWQOS 2004, 2004: 47–56

[46] BEA Systems, BEA Weblogic. http://www.bea.com/weblogic

[47] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, A. Tantawi, An analytical model for multi-tier internet services and its applications, in: Proc. of ACM SIG. International Conf. on Measurement and Modeling of Computer Systems, Performance Evaluation Review 33 (1) (2005)

[48] Cisco Local Director. http://www.cisco.com/

[49] Ch. Li, G. Peng, K. Gopalan, T. Chiueh, Performance guarantees for cluster-based internet services, in: 23rd International Conf. on Distributed Computing Systems, May 2003: 378–385

[50] D. Dinda, D. O' Hallaron, Host load prediction using linear models, Cluster Computing 3 (4) (2000)

[51] D. Villela, P. Pradhan, D. Rubenstein, Provisioning servers in the application tier for E-commerce systems quality of service, in: 12th IEEE International Workshop on QOS, IWQOS 2004, June 2004

[52] E. Casalicchio, S. Tucci, Static and dynamic scheduling algorithms for scalable web server farm, Ninth Euromicro Workshop on Parallel and Distributed Processing, 7–9 Feb. 2001: 369–376

[53] E. Casalicchio, M. Colajanni, A client-aware dispatching algorithm for web clusters providing multiple services, in: Proc. of the 10th International Conf. on WWW, Hong Kong, 2001: 535–544

[54] E. Choi, Performance test and analysis for an adaptive load balancing mechanism on distributed server cluster systems, Future Generation Computer Systems 20, 2004: 237–247

[55] Foundry network Application Switching, ServerIronXL. http://www.foundrynet.com/

[56] G. Teodoro, T. Tavares, B. Coutinho, W. Meira Jr., Guedes, load balancing on stateful clustered web servers, in: Proc. of 15th Symposium on Computer Architecture and High Performance Computing, 10–12 Nov. 2003: 207–215

[57] IBM WebSphere Software. http://www.ibm.com/

[58] J. Cao, X. Wang, S.K. Das, A framework of using cooperating mobile agents to achieve load sharing in distributed web server groups, Future Generation Computer Systems 20, 2004: 591–603

[59] J.L. Hellerstein, K. Katircioglu, M. Surendra, An on-line, business-oriented optimization of performance and availability for utility computing, IEEE Journal on Selected Areas in Communications 23 (10), 2005: 2013–2021

[60] L. Aversa, A. Bestavros, Load balancing a cluster of web servers using distributed packet rewriting, in: Proc. of IEEE IPCCC' 2000, Phoenix, AZ, Feb 2000

[61] L. Cherkasova, P. Phaal, Session-based admission control: A mechanism for peak load management of commercial web sites, IEEE Transactions on Computers 51 (6) (2002)

[62] M. Andreolini, E. Casalicchio, A cluster-based web system providing differentiated and guaranteed services, Cluster Computing 7 (1) (2004): 7–19

[63] M. Andreolini, M. Colajanni, M. Nuccio, Scalability of content-aware server switches for cluster-based web information systems, in: Proc. of 12th International World Wide Web Conf., WWW 2003, Budapest, Hungary, May 2003

[64] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, Scalable content-aware request distribution in cluster-based network servers, in: Proc. USENIX 2000, San Diego, CA, June 2000

[65] M. Dahlin, Interpreting state load information, IEEE Transactions on Parallel and Distributed Systems 11 (10) (2000): 1033–1047

[66] N. Tran, D. Reed, Automatic ARIMA time series modeling for adaptive i/o prefetching, IEEE Transactions on Parallel and Distributed Systems 15 (4), 2004: 362–377

[67] Q. Zhang, A. Riska, W. Sun, E. Smirni, G. Ciardo, Workload-aware load balancing for clustered web servers, IEEE Transactions on Parallel and Distributed Systems 16 (3), 2005: 219–233

[68] R. Bashroush, I. Spence, P. Kilpatrick, T.J. Brown, A generic reference software architecture for load balancing over mirrored web servers: NaSr case study, in: The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005: 132

[69] T. Abdelzaher, K.G. Shin, N. Bhatti, Performance guarantees for web server and systems: A control theoretical approach, IEEE Transactions on Parallel and Distributed Systems 13 (1), 2002: 80–96

[70] T. Schroeder, S. Goddard, B. Ramamurthy, Scalable web server clustering technologies, IEEE Network (May–June), 2000: 38–45

[71] V. Cardellini, E. Casalicchio, M. Colajanni, S. Tucci, Mechanisms for quality of service in web clusters, Computer Networks 37 (6), 2001: 761–771

[72] V. Cardellini, E. Casalicchio, M. Colajanni, M. Mambelli, Web-switch support for differentiated services, ACM SIGMETRICS Performance Evaluation Review 29 (2), 2001

[73] V. Cardellini, E. Casalicchio, M. Colajanni, Ph.S. Yu, The state of the art in locally distributed web-server systems, ACM Computing Surveys (CSUR) 34 (2): 2002: 263–311

[74] Liu Anfeng, Chen Zhigang, Zeng Zhiwen, Zeng Biqing, Web cluster load balancing algorithm based on data mining, Computer Engineering and Applications, 2003, Issue 25

[75] Zhu Changwu, Dai Shangping, Liu Zhi, Web cluster load balancing method based on genetic algorithm, Journal of Guangxi Normal University (Natural Science Edition), 2006, Issue 04

[76] Li Shuangqing, You Lian, Cheng Daijie, Content-based load balancing strategy for web cluster systems, Journal of Chongqing University (Natural Science Edition), 2003, Issue 05

[77] Zhang Wensong, LVS, http://www.linuxvirtualserver.org/zh/lvs4.html

[78] Li Guojie, Strategic Thinking on Supercomputing and Capability Services. Information Technology Letters, 2005, 3(1): 1-8

Author Biographies

Feng Binquan: Master' s student, Institute of Computing Technology, Chinese Academy of Sciences
Song Ying: Ph.D. student, Institute of Computing Technology, Chinese Academy of Sciences
Sun Yuzhong: Professor, Institute of Computing Technology, Chinese Academy of Sciences
Email: yuzhongsun@ict.ac.cn

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