symmetry

MDPI

Article

A Scalable Virtualized Server Cluster Providing Sensor Data Storage and Web Services

Mei-Ling Chiang * and Tsung-Te Hou

Department of Information Management, National Chi Nan University, Puli, Nantou 545, Taiwan; s104213509@mail1.ncnu.edu.tw

* Correspondence: joanna@mail.ncnu.edu.tw

Received: 15 August 2020; Accepted: 23 November 2020; Published: 25 November 2020

Abstract: With the rapid development of the Internet of Things (IoT) technology, diversified applications deploy extensive sensors to monitor objects, such as $PM_{2.5}$ air quality monitoring. The sensors transmit data to the server periodically and continuously. However, a single server cannot provide efficient services for the ever-growing IoT devices and the data they generate. This study bases on the concept of symmetry of architecture and quantities in system design and explores the load balancing issue to improve performance. This study uses the Linux Virtual Server (LVS) and virtualization technology to deploy a virtual machine (VM) cluster. It consists of a front-end server, also a load balancer, to dispatch requests, and several back-end servers to provide services. These receive data from sensors and provide Web services for browsing real-time sensor data. The Hadoop Distributed File System (HDFS) and HBase are used to store the massive amount of received sensor data. Because load-balancing is critical for resource utilization, this study also proposes a new load distribution algorithm for VM-based server clusters that simultaneously provide multiple services, such as sensor services and Web service. It considers the aggregate load of all back-end servers on the same physical server that provides multiple services. It also considers the difference between physical machines and VMs. Algorithms such as those for LVS, which do not consider these factors, can cause load imbalance between physical servers. The experimental results demonstrate that the proposed system is fault tolerant, highly scalable, and offers high availability and high performance.

Keywords: Internet of Things; virtualization; server cluster; load balancing; sensor data storage

1. Introduction

With the rapid development of the Internet of Things (IoT) technology, many IoT devices have emerged, and various IoT applications such as environmental monitoring, precise agriculture, smart health care, smart families, and intelligent manufacturing have been developed. With the incorporation of big data technology and IoT applications, increasing numbers of sensors are being deployed, and the amount of data that is transmitted is increasing. Designing a system to effectively support IoT applications face lots of challenges.

For IoT applications that deploy a large number of sensors to monitor changes in environmental parameters, the sensor data are characterized in terms of the periodic and continuous generation and the large amount of data transmitted. For example, as air pollution is becoming severe, a large number of sensors have been deployed to monitor air quality. These sensors detect the value of $PM_{2.5}$ for each area over a long time and periodically transmit the sensor data to the server for storage. In order to obtain more accurate values, more and more sensors are deployed, so the amount of data generated and transmitted is getting higher. Since the sensing period is short, data are generated faster, and more storage space is required over time.

On the other hand, service providers usually allow users to observe real-time changes in air quality using a Web browser. To process the growing number of sensors and their generated sensor data and provide stable Web services simultaneously, the system requires more servers to receive and process sensor data. The system must also have a fault tolerance mechanism to ensure that data are not lost and the system is highly available.

To address growing service demands, most enterprises base on the concept of symmetry of architecture and quantities in system design and use server cluster technology to construct their systems. Traditionally, server clusters use multiple physical servers to provide services and employ effective load-balancing mechanisms to distribute the load over servers. Servers can be added into or remove from the cluster on demand. This allows a high degree of scalability, excellent reliability, and high efficiency. With the maturity of computer hardware and virtualization technology, administrators can maintain and deploy systems more conveniently and efficiently [1,2]. Hence, enterprises and organizations use virtualization technology to deploy their systems. Physical server clusters are thus transformed into virtualized server clusters. Besides, virtual machines (VMs) on a physical server are independent, so administrators can construct VMs that provide a variety of services on a server to meet the diverse service needs of users and make full use of system resources.

Tenants can construct individual server clusters or rent VMs from cloud providers, such as Google Cloud Platform (GCP) [3], Amazon Web Services (AWS) [4] and Microsoft Azure [5], in line with their demand for computing resources. In order to increase the reliability and performance of services, a group of VMs can be created, and load-balancing services implemented by providers are used to distribute the workload between physical server groups.

To avoid the constraints of cloud providers and reduce security concerns about data leakage, this study designs and implements a server cluster for IoT applications. To efficiently meet the ever-increasing service demands, the system architecture should allow a high degree of scalability, high performance, and high availability for providing IoT-generated data repositories and Web services. This study describes our experience in adopting the open-source software to develop our server cluster system. The Linux Virtual Server (LVS) [6,7] and the Kernel-based Virtual Machine (KVM) [8] technologies are used to deploy a VM cluster named VM-based Cluster Providing Hybrid Service (VMC/H). LVS and KVM technologies are adopted since they are directly supported in the Linux kernel. Being operated at the kernel layer, they are shown to have excellent performance in the related studies [1,2,7]. It consists of a front-end server, also called a load balancer for dispatching requests, and several back-end servers to provide services. On each host of the VMC/H cluster, several VMs are constructed as back-end servers to provide sensor services and provide Web services to users that allow sensor data to be browsed. In order to achieve high availability, a standby front-end server is additionally set up and serves as a backup. If the primary front-end server fails, the backup server continues to provide system service. Besides, the Hadoop Distributed File System (HDFS) [9] and distributed NoSQL database HBase [10] are used to establish a storage cluster to process and store a large amount of sensor data, which mainly takes advantage of their features of high performance and high availability.

Though several open-source software is integrated and used to construct a server cluster system, nevertheless, to meet the high-performance needs, the load-balancing issue should be addressed for the VM-based server cluster because it significantly affects the use of system resources. Current load distribution algorithms provided in the LVS cluster, such as Weighted Round-Robin (WRR) [7] and Weighted Least Connection (WLC) [7], are not appropriate to be used for virtual server clusters that provide different types of services on the same physical server group. Two factors must be taken into account. The first one is the difference between physical machines and VMs. The other one is the aggregate loads of different services provided by the server clusters. Otherwise, even VMs are load-balanced, physical machines may not be.

In this study, we further explore efficient load distribution strategies for VM-based server clusters. These virtual server clusters are constructed on the same physical server group but provide different

types of services. We have designed and implemented a new load distribution strategy called VM-based Weighted Least Connections for Hybrid Service (VMWLC/H). It not only considers the difference between physical and virtual machines but also the aggregate load of different services provided by the server clusters to prevent a load imbalance between physical servers.

We further perform experiments to explore the difference between the overall system performance for the proposed VMC/H using multiple lower-cost physical servers and a single physical server with higher capability. The experimental results show that these VMC/H clusters with the VMWLC/H load distribution strategy provide scalable and reliable data storage and efficiently provide sensor services and Web browsing services simultaneously. The VMC/H system constructed using multiple physical servers performs better than the VMC/H system constructed using a single powerful physical server. Besides, the proposed VMWLC/H helps balance the load over servers and increase performance.

The contribution of this work is as follows. First, to efficiently support IoT applications that provide hybrid services such as web service and sensor data storage, we have utilized several open-source software to construct our VM cluster. We have further improved them to achieve the VMC/H system's key properties regarding scalability, fault tolerance, high performance, and virtualization. Second, we have designed and implemented a new load distribution strategy VMWLC/H for VM-based server clusters that provide hybrid services. It considers the difference between physical and virtual machines and the aggregate load of hybrid services on servers to prevent a load imbalance between physical servers. Besides, to demonstrate the effectiveness of the proposed VMC/H system for IoT applications and the VMWLC/H for load balancing, they are practically implemented and use air quality monitoring as an application.

The remainder of this paper is organized as follows. Section 2 describes the technology background and related work. Section 3 describes the design and implementation of the proposed system, including the proposed VMC/H virtual server cluster and the proposed load-balancing algorithm, to provide hybrid services in the same virtual server cluster. Section 4 describes the experimental environment and presents experimental results. Finally, Section 5 concludes.

2. Technology Background and Related Work

This study utilizes Linux Virtual Server (LVS) [6,7] technology to construct a server cluster to provide services with high performance, a high degree of scalability, and high availability. It also uses the Hadoop Distributed File System (HDFS) [9] and the Not Only SQL (NoSQL) database technology, HBase [10], to store large amounts of IoT-generated data. The LVS technology is described in Section 2.1, and HDFS and HBase are described in Section 2.2. Section 2.3 reviews related work.

2.1. Linux Virtual Server Cluster

The Linux Virtual Server (LVS) [6] is a server cluster system with high scalability and availability. The TCP/IP stack of Linux kernel is extended to support IP load balancing techniques. It is composed of the front-end server (FE) and a server pool. The FE is a load balancer responsible for forwarding and dispatching requests from clients to the appropriate server in the cluster. The server pool consists of several back-end servers (BEs) providing actual services. To clients, it looks like there is only one high-performance server with a single IP address. When the FE receives the SYN packet from a client requesting to establish a connection, it then performs the designate load-balancing algorithm to select a BE to serve the request. After the connection is established, requests from the same connection would be sent to the same BE for processing. All the work is performed inside the Linux kernel, so that the handling overhead of the load balancer is low.

LVS provides three routing mechanisms to forward packets, and direct routing (DR) was shown to have the best performance, as reported in the study [7]. As, after the BE processes the request packet, it sends the response to the client directly instead of forwarding the response to the client by way of the FE, we adopt the DR mechanism in our experimental VM clusters.

In the beginning, an administrator must designate one load balancing algorithm when setting up an LVS cluster. Several algorithms are provided in LVS, and the representatives are as follows:

- The Round-Robin scheduling (RR) selects a target BE in a round-robin manner. In RR, all BEs are treated as having equal processing capability.
- The Weighted Round-Robin scheduling (WRR) is similar to RR, whereas it needs an administrator to assign static weight values to BEs according to their processing capability. Such that servers with higher weight values will be assigned to serve more connections.
- The Least Connection scheduling (LC) selects a target BE with the least number of active connections. The FE tracks the number of active connections processed in every BE.
- The Weighted Least Connection scheduling (WLC) is similar to LC; nevertheless, it allows an administrator to assign static weight values to BEs in compliance with their processing capability. Assume that there are n BEs. For each BE i, its weight denotes W_i, and its number of active connections denotes C_i. WLC determines the target BE that has the smallest C_i/W_i value.

2.2. Hadoop and Hbase

In this study, HBase [10] built on the Hadoop [9] distributed file system is used as a database for storing sensor data. They are used because besides being an open-source software framework for distributed storage and processing of big data sets across clusters of computers, and Apache Hadoop has the features of high scalability, high availability, and high reliability. Hadoop uses master/slave architecture, which includes several modules such as Hadoop Distributed File System (HDFS) [9] and MapReduce [11]. The NameNode and Resource Manager are usually placed on the machines of the master nodes, responsible for dispatching data or work on slave nodes, while the DataNode and Node Manager are placed on the machines of the slave nodes to store data or perform assigned work. MapReduce performs parallel computing programs through YARN, a framework for job scheduling and cluster resource management, to provide big data analysis and processing capability.

HDFS is a distributed file system that provides efficient access to application data with high fault tolerance and high throughput. It has a master/slave architecture, which is composed of a single NameNode and multiple DataNodes. The NameNode is responsible for managing the file system namespace, and any changes to the file will be recorded in the NameNode. HDFS stores each file in units of data blocks, and these blocks are stored in a set of DataNodes. The NameNode is responsible for copying the data blocks of all files to generate replicas and store replicas in different DataNodes to provide fault tolerance. When a DataNode in the HDFS cluster fails, the NameNode will automatically move and copy the data blocks to other normally operating DataNode, to ensure the integrity of the data. When the client requests file access, the NameNode will query the information in metadata and assign the DataNode that stores the file to establish a connection with the client. After the connection is established, the client will directly access the file on the DataNode.

However, once the NameNode fails, the Hadoop system cannot continue to provide services. This study then utilizes Zookeeper [12] to deal with this situation. Zookeeper is an open-source project under the Apache Software Foundation, developed for distributed applications and providing consistent coordination services. The functions provided to users include server synchronization, configuration maintenance, and server grouping and naming so that servers in distributed systems can provide consistency and highly reliable services. In this study, when the NameNode (i.e., active) fails, Zookeeper will immediately notify the NameNode (i.e., standby) to continue to provide services to ensure the entire Hadoop cluster will not interrupt service.

Hbase [10] is an open-source, scalable, and distributed non-relational database modeled after Google's Bigtable [13]. It is a part of the Apache Hadoop project, which runs on top of the HDFS and can store a large amount of data in a fault-tolerant manner.

Apache Thrift [14] is a software framework for the development of cross-language services. It provides a cross-language Remote Procedure Call service, whose primary purpose is to allow programs written in different languages to be executed or communicated between different systems.

It supports several programming languages, including C++, C#, Java, Perl, PHP, Python, and Ruby. Similarly, programs written in different programming languages can be used to query or write data in HBase using the Thrift service. However, in the current HBase cluster, only a single server can provide the Thrift service, which can easily become a system bottleneck. To improve performance, this study then additionally constructs a server cluster to provide Thrift service.

2.3. Related Work

The virtual machine cluster (VM cluster) consists of several virtual machines (VMs) to provide services to clients. There are many studies related to VM clusters, such as VMs migration, VM clusters deployment, and load balancing in a set of VM clusters. Among them, VMCTune [15], a load-balancing scheme, is proposed for VM cluster based on dynamic resource allocation. VMCTune monitors resources utilization of VMs and physical machines (PMs). It supports live VM migration according to CPU, memory, and network usage, so VMs can be migrated to another PMs dynamically to optimize the resource utilization and the performance of a VM cluster.

VNIX [1] is developed to reduce resource contention between VMs on the same host. Two architectures of VM clusters based on the LVS [6,7] are proposed, i.e., centralized LVS and distributed LVS. There are multiple VM-based LVS clusters constructed on several PMs, where the front-end server (FE) and the back-end servers (BEs) are all constructed on VMs. VNIX then employs the DNS server to spread client requests to these LVS clusters with a round-robin manner for load balancing. In the centralized LVS, the FEs of all LVS clusters are centrally constructed on a single machine. Whereas, in the distributed LVS, the FE of each LVS and its associated BEs are constructed on VMs of the same host. Their experimental results showed that the throughput of VM clusters with VNIX was three times higher than traditional systems, and the error rate significantly reduced.

Our previous study [2] explores the issues of implementing server clusters based on VMs. Several kinds of VM-based server clusters with different architectures are developed and discussed, including single VM cluster, hierarchical multiple VM clusters, and distributed multiple VM clusters. They are based on the LVS, and Kernel-based Virtual Machine (KVM) [8] is used to set up VMs. Besides, load distribution algorithms that adapt the traditional Least Connections (LC) [7] algorithm to VM clusters are proposed. The experimental results showed that VM clusters with the single architecture or the hierarchical architecture obtain significantly higher performance than the distributed VM cluster that consists of multiple VM clusters with a DNS to spread the load to VM clusters. The proposed load distribution algorithms outperform the Weighted Least Connections (WLC) that does not distinguish PMs from VMs. Since only the single type of service (i.e., Web service) is concerned in server clusters, hybrid services and sensor data storage are not provided. The fault-tolerant mechanism for load balancer is not included.

Several studies [16–18] design their VM server clusters using software-defined networking (SDN). As load balancing is also critical for SDN-based server clusters, some propose new load balancing strategies to utilize cluster resources effectively. Among them, the dynamic weighted random selection (DWRS) [18] considers the real-time server loads when dispatching requests to servers with different processing capabilities. Server loads are periodically collected and reported to the controller, and server weights are dynamically updated. Underutilized servers are assigned higher weights, so they have a higher possibility of being selected to process requests. Besides, server weights need not be assigned in advance, which prevents the load imbalance due to inappropriate settings. Hybrid services, sensor data storage, and fault-tolerant mechanism for the controller (i.e., the load balancer) are beyond the scope of this paper.

Resource allocation is one of the important subjects in the IoT, and the goals are load balancing and minimizing operational cost and power consumption. A systematic literature review (SLR) is provided in the study [19], in which the resources allocation methods in the IoT and used algorithms are investigated.

It is crucial to construct a sensor data repository with high scalability, high reliability, and high efficiency. Many related works have proposed their system architectures and employed existing technologies in their applications. For example, the authors in the study [20] used Raspberry Pi to develop a portable health-care sensor tool to record the ECG changes of patients and used the MySQL [21] database to store relevant sensor data. The authors in the study [22] considered that the data generated by medical sensors is vast, and the traditional relational database cannot process large data in real time. They then used Hadoop [9] and Hive non-relational database [23] technologies for storage and processing. In their experiments with MySQL and Hive, the results showed that the use of non-relational database technology to deploy a database system for storing medical sensor data improved performance in the data processing.

The study [24] used the LVS clustering technology [6,7] and the non-relational database MongoDB [25] to establish a cloud storage system named J-TEXTDB, which is used to process and store the data generated from the nuclear fusion experiments. The experimental results showed that even if a large number of requests arrive, using the LVS cluster can effectively distribute clients' requests of data written or query and achieve load balancing among servers. However, if the system is built with a single front-end server, when it fails, the overall system will fail too.

3. System Design and Implementation

This study designs and implements a VM cluster that provides hybrid services named VM-based Cluster Providing Hybrid Service (VMC/H). We utilize several open-source software to construct it. Nevertheless, straightforwardly using this software to construct a system cannot provide efficient support for IoT applications, nor can it meet the goal of providing high availability. The system should be highly scalable and highly available to meet the increasing service demands and provides hybrid services efficiently at the same time. For large deployment of servers, virtualization technology is employed for easier management and deployment of servers. This section describes our experience, design, and implementation in achieving the above goals. The proposed VMC/H features a fault tolerance mechanism and a novel load distribution strategy.

3.1. System Architecture

In order to construct a VM-based server cluster, Kernel-based Virtual Machine (KVM) [8] technology is used to create VMs, and LVS clustering technology [6,7] is used to construct server clusters. The cores of LVS and KVM are implemented as Linux kernel modules and operated at the kernel layer. With the LVS, the OS kernel of the load balancer can perform load balancing directly at the IP level rather than at the application layer. KVM is a virtualization module running in the Linux kernel that allows the kernel to function as a hypervisor. Being directly supported in the Linux kernel, they are shown to have excellent performance and are adopted in many studies [1,2,7,24].

Figure 1 shows the VMC/H architecture, which consists of multiple physical servers (i.e., Host 1-N) on which several VMs are built. Similar to VNIX [1], all back-end servers (BEs) that provide services are constructed on VMs. In contrast, the load balancer, i.e., the front-end server (FE), is constructed inside the Linux host OS rather than directly constructed on a VM as VNIX does. That is, the host OS kernel instead of the guest OS kernel performs load balancing directly. This architecture with a kernel-level load balancer running in the host OS layer performs significantly better than the ones with a load balancer running on a VM or a DNS server, as shown in our previous study [2].

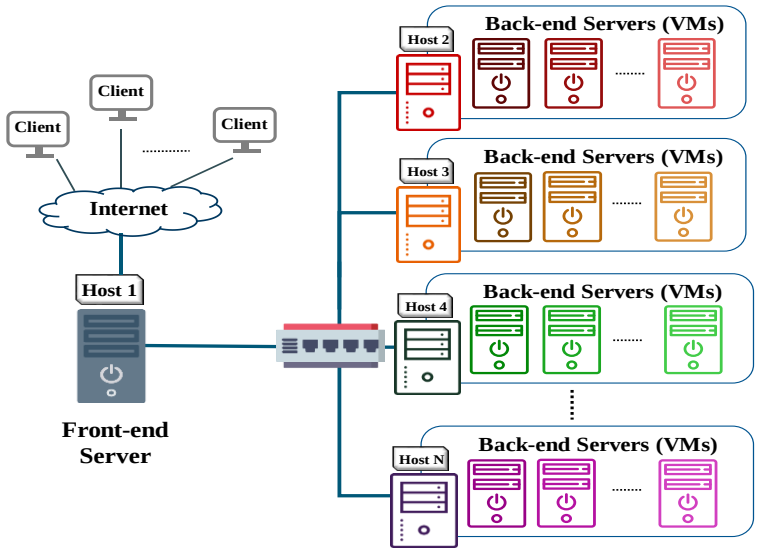


Figure 1. Linux Virtual Server (LVS) virtual machine (VM) Server Cluster.

As shown in Figure 1, the FE is the only server that receives request packets from clients. When a TCP SYN request is received to establish a connection, the FE (i.e., Host 1) selects a BE (VM) from the system to process the request using the designated load distribution algorithm and then forwards packets to servers. After processing, the selected BE directly transmits response packets to the client. The FE then dispatches the following requests from the same connection to the same BE the next time it receives a request. However, since the FE of LVS is the only server that receives incoming request packets, the overall system will fail when it fails.

To provide high availability, we additionally create another FE that acts as a backup server. Each uses Heartbeat [26] to monitor each other's heartbeats, so the overall system is tolerant of faults, and the system remains operable if the FE fails. Figure 2 shows the system architecture of VMC/H.

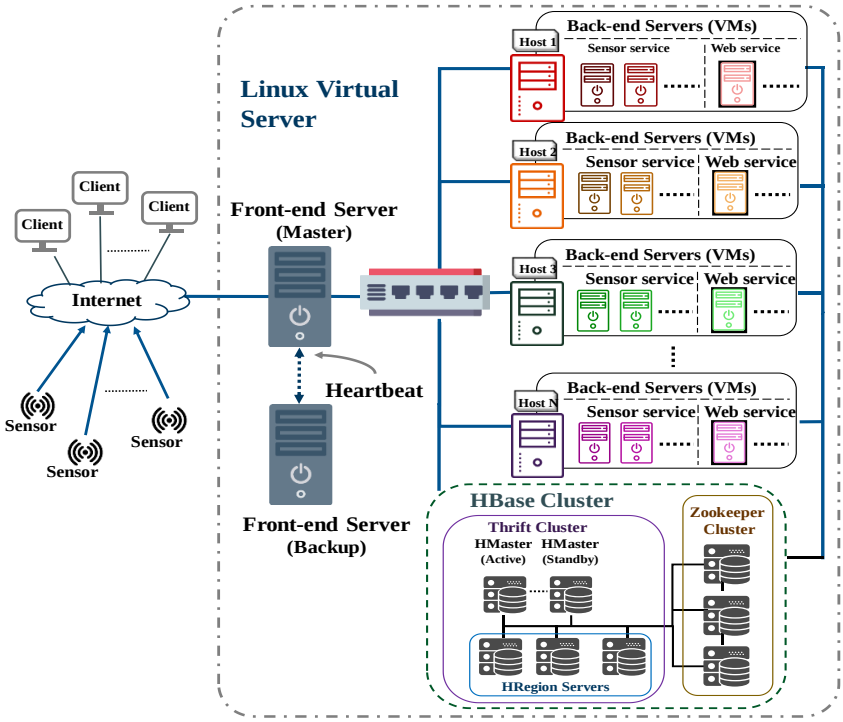


Figure 2. VM-based Cluster Providing Hybrid Service (VMC/H) on Multiple Physical Servers.

For data storage, the storage cluster is created by Hadoop [9] distributed file system and distributed non-relational database HBase [10]. It is then used to process and store large amounts of sensor data. However, once the NameNode fails, the Hadoop cluster cannot continue to provide services, as discussed in Section 2.2. This study then further utilizes Zookeeper [12] to deal with the situation due to a single NameNode failure to provide high availability. When the active NameNode fails, Zookeeper will immediately notify the standby NameNode to continue to provide services.

To provide hybrid services, initially, two groups of LVS virtual services are built in the VMC/H. One provides sensor services in the LVS server cluster, as shown in Figure 3. When the FE receives the first SYN packet from one sensor to establish a connection, it forwards the packet to a BE selected from the server pool for the sensor service. The data packets sent from that sensor are then processed by this BE and stored in the HBase cluster.

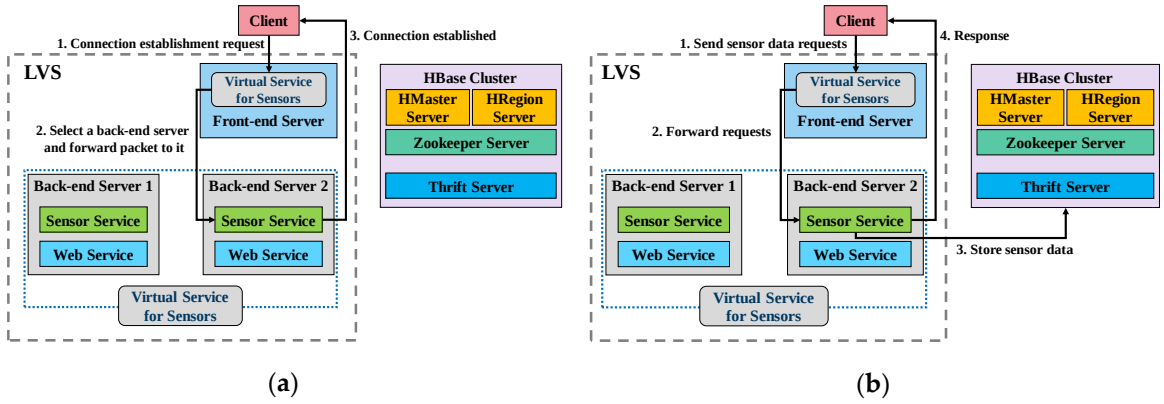


Figure 3. VMC/H Provides Sensor Services. (**a**) Connection Establishment; (**b**) Storing Sensor Data.

The second one provides Web services to browse sensor data stored in the HBase, as shown in Figure 4. To provide Web services, the FE forwards the SYN packet sent from a client to establish a connection to a BE that is selected for the Web service. Afterward, the data packets sent from that client through this connection are processed by this BE. In terms of routing mechanisms, this study uses direct routing, so when the BE finishes processing the request, the response packet is directly sent back to the client and is not forwarded through the FE. Therefore, performance is increased.

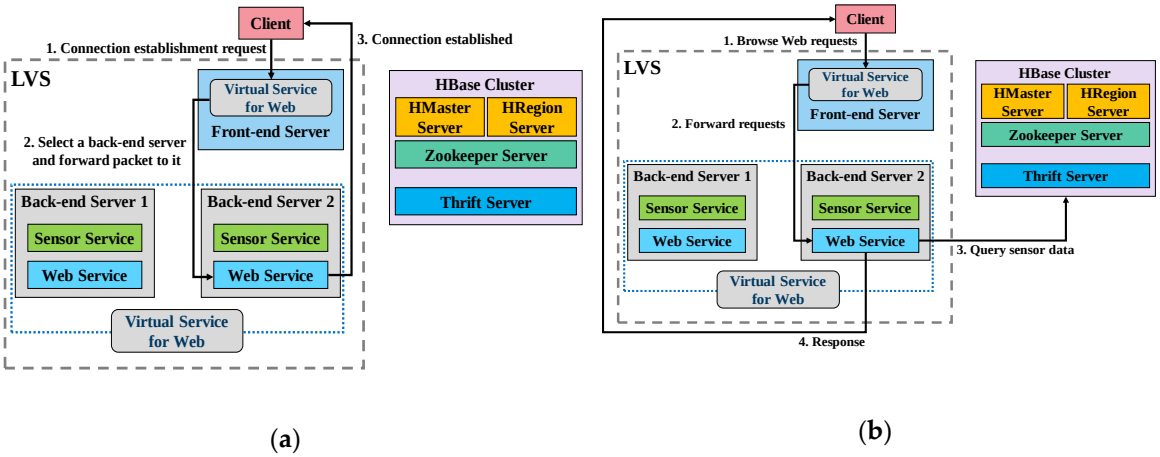


Figure 4. VMC/H Provides Web services and Sensor Data Access. (**a**) Connection Establishment; (**b**) Browsing Sensor Data.

Originally, the Thrift [14] server that provides cross-language service for accessing sensor data stored in the HBase is a single server. When handling large amounts of incoming sensor data, it easily becomes a bottleneck. To efficiently provide hybrid services, we build the third virtual service to

provide the Thrift service. As shown in Figure 5, this transforms the original Thrift server in the traditional HBase from single server architecture to a server cluster with scalability and a load-balancing mechanism. Therefore, this study increases the efficiency of Thrift by using LVS cluster technology to construct the Thrift service, which increases the overall system performance.

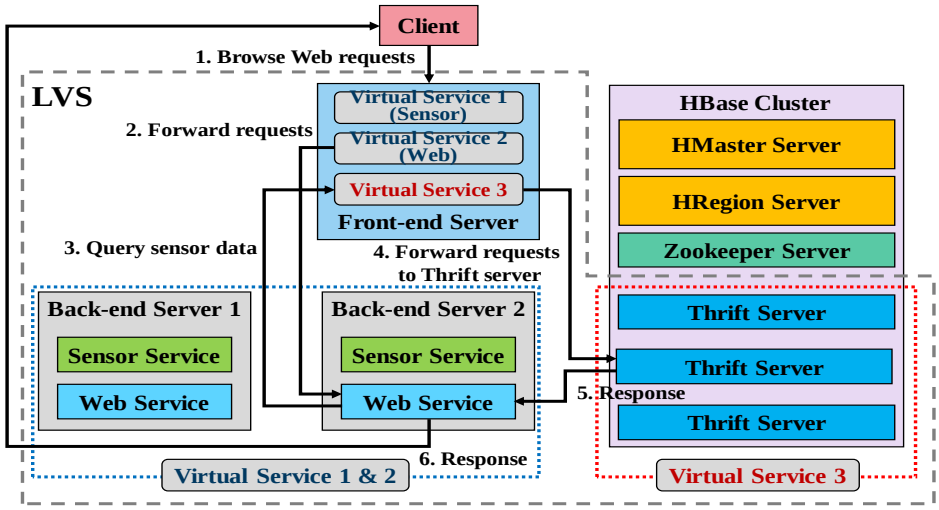


Figure 5. VMC/H Provides Three Virtual Services for Web services and Sensor Data Access.

For administrators, the choice of hardware equipment for them to deploy their systems is important. This study builds two VMC/H systems with equivalent cost to explore the difference in the performance of different implementation methods. The one denoted as VMC/H(M) uses multiple lower-cost physical servers, as shown in Figure 2. The other denoted as VMC/H(S) uses a powerful single physical server, as shown in Figure 6. It is easier for administrators to deploy and maintain VMC/H(S) than VMC/H(M). Despite the maintenance issue, the performance difference of these two systems in terms of receiving and processing a large number of packets sent continuously from sensors for an extended period is measured in Section 4. Besides, once additional physical servers are added, VMC/H(S) becomes VMC//H(M).

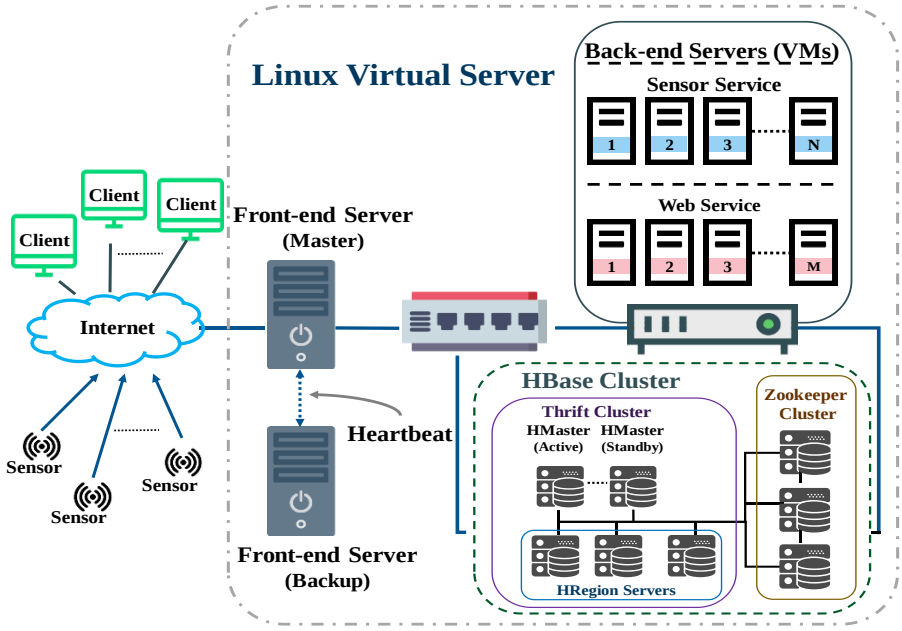


Figure 6. VMC/H on a Single Physical Server.

3.2. Virtual Machine Weighted Least Connections Algorithm for Hybrid Services

If all BEs are constructed as VMs running on a powerful single physical server, the existing load distribution algorithms provided in LVS can be directly applied to maintain load balancing among back-end servers. However, for a system such as VMC/H(M) that uses multiple VMs constructed on a cluster of physical servers and provides various services simultaneously, the load distribution algorithm must be modified and adapted for VM-based clusters. Section 3.2.1 explains the reasons, and Section 3.2.2 presents the proposed method used to balance the server loads.

3.2.1. Problem Statement

To efficiently provide hybrid services, the existing load distribution algorithms in LVS are not appropriate to be directly used on VM clusters like VMC/H(M). The reasons are as follows. Firstly, they do not distinguish between VMs and physical servers, which may lead to load imbalance among physical servers. Secondly, they are originally designed to work for the server clusters providing only a single type of service. Thirdly, the weight values assigned to back-end servers in compliance with their processing capability are integers. Since the weight granularity is based on integers, it is harder to represent the relative performance difference between physical servers exactly. Therefore, the server load may not be well balanced. They are explained in detail as follows.

The current LVS cluster technology [6] only uses IP addresses to distinguish the front-end server (FE) and back-end servers (BEs). It cannot distinguish between VMs and physical servers. Among all LVS load-balancing algorithms, the Weighted Least Connections (WLC) [7] has the best performance. The FE selects the BE with the fewest connections after considering the server's processing capability. However, running WLC in a VM server cluster like VMC/H(M) is likely to result in an imbalanced load between physical servers since only load balancing among BEs is considered, as shown in Figure 7. As a result, once many requests from clients are simultaneously forwarded to different BEs on the same physical server, the total load on this physical server is much higher than that on other physical servers.

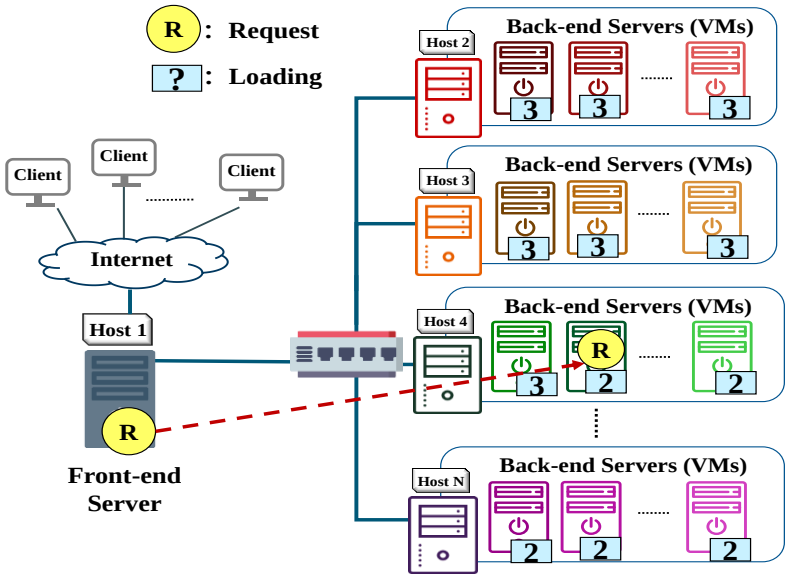


Figure 7. LVS VM Server Cluster with weighted least connection (WLC).

In our preliminary study [2] of virtual server clusters, the difference between physical servers and VMs is considered and added into WLC. Two effective load-balancing algorithms were proposed: Virtual Machine Least Connections (VMLC) and Virtual Machine Weighted Least Connections (VMWLC). VMLC separates the process of selecting the target BE into two steps, as shown in Figure 8. The first step involves the FE (i.e., Host 1 in Figure 1) receiving an SYN packet from the client to establish a connection. The target physical server with the lightest load is selected from all the physical servers

in the cluster (i.e., Host 2-N in Figure 1). For VMLC, the load on a physical server is the aggregate number of connections served by all BEs that run on it. The second step involves the FE selecting a target BE with the least connections from all BEs that run on the target physical server. Afterward, packets from the same connection are also served by this server.

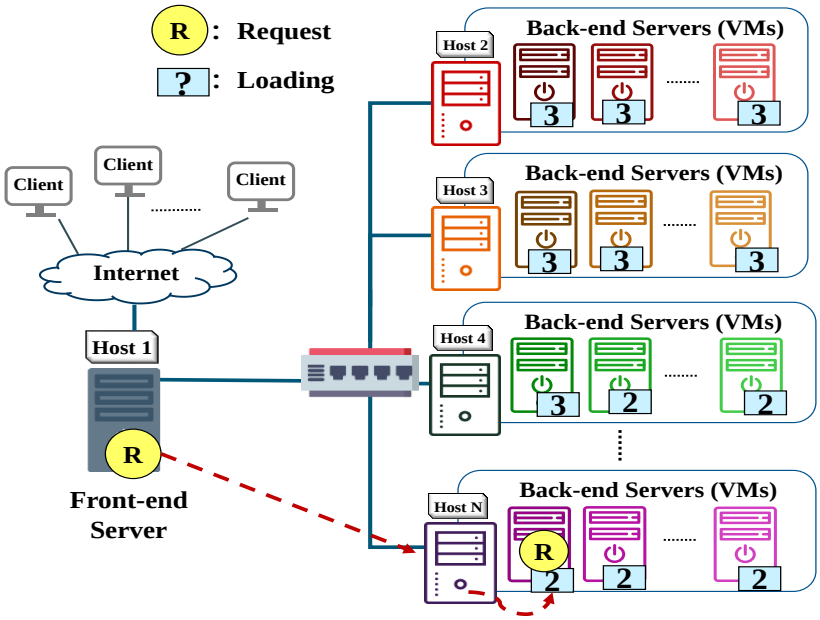


Figure 8. LVS VM Server Cluster with VMLC.

Based on VMLC, VMWLC additionally considers the different processing capabilities of physical servers in the VM Cluster (i.e., Host 2-N in Figure 1) and then assigns weights to servers. Since not every physical server is the same and the processing capabilities of VMs constructed on physical servers may not be the same, administrators can assign appropriate weight values to physical servers and the VMs according to their processing capability.

These load distribution algorithms are feasible for the LVS VM cluster that provides only a single type of service. This is because, in LVS cluster technology, an IP with a service port and a designate load-distribution algorithm is configured for an individual LVS service. For clusters that provide multiple types of services on the same physical server groups, multiple LVS services with different ports must be created. These individual LVS services are independent of each other.

For VM-based clusters such as VMC/H(M) that provide multiple types of services on the same physical server groups, an effective load distribution algorithm must consider the aggregate load of all BEs on each physical server, even though some BEs serve different types of services. This is because if the FE considers only the load of BEs that provide the same service, some physical servers may become overloaded, and others may be underutilized. An example is illustrated in Figure 9, in which VMs on Hosts 1–5 provide Web services. When the FE receives a Web request from the client to browse the sensor data, the request is forwarded to the BE on Host 1 because it provides the Web service and has the smallest load. Nevertheless, the current aggregate loads of all BEs on Host 4 and on Host 5 are lower than that of Host 1.

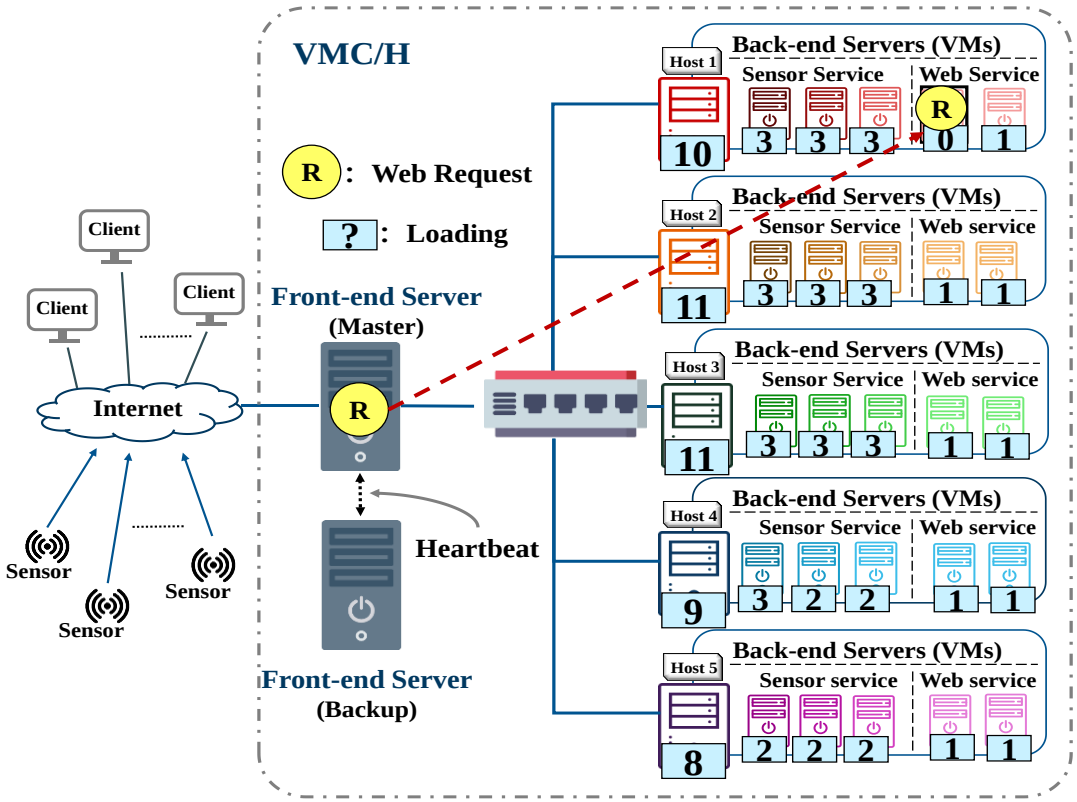


Figure 9. Load imbalance occurs when VMLC is directly applied in a VMC/H system.

3.2.2. The Proposed VM-Based Weighted Least Connections for Hybrid Services (VMWLC/H)

When the system provides hybrid services, the servers are prone to uneven load due to the nature of the services provided. For sensor service, the sensor data are collected periodically and stored in the database. The load of the server that provides sensor service does not change much, but the load of the server that provides Web service may vary greatly depending on the number of the clients. To avoid load imbalance between physical servers, we adapt VMWLC to consider aggregate loads of all BEs on the same physical servers that provide different types of services. The proposed algorithm is named VM-based Weighted Least Connections for Hybrid Service (VMWLC/H).

Figure 10 illustrates an example in which the respective aggregate loads of Hosts 1–5 are 10, 11, 11, 9, and 8. Host 5 is the least loaded host. When the FE receives a Web request from the client to browse the Web, it forwards the request to the least loaded BE on Host 5 providing the Web service.

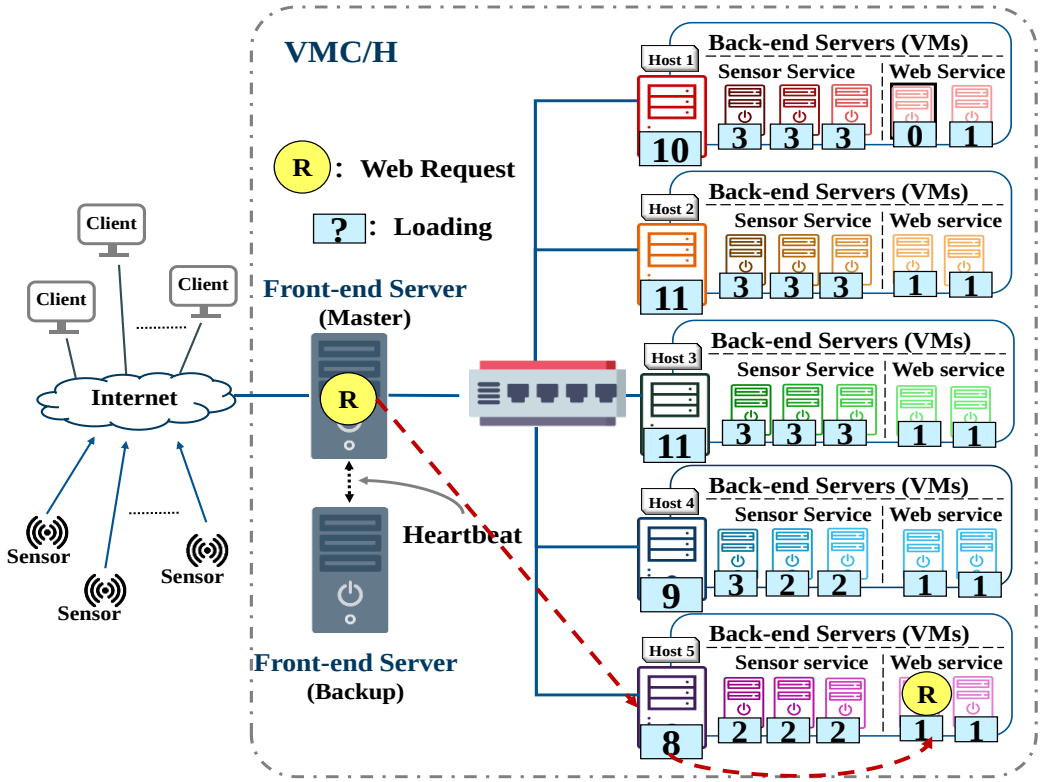


Figure 10. Web request distribution using VMWLC/H, which considers the aggregate loads.

3.2.3. Implementation of the Proposed VMWLC/H

To implement the proposed VMWLC/H, the FE of the LVS must be modified. In the proposed VMC/H system, the FE has added the tracking of the physical server on which a BE is running, so it maintains the mapping of VMs to physical servers. In the original LVS, the FE tracks the number of connections served by each BE. The number of connections represents the load. The FE is also modified to maintain the current aggregate load on physical servers. The FE then uses this information to forward incoming requests to selected BE according to VMWLC/H algorithm. Our implementation also provides flexibility to allow physical servers to be different in their processing capabilities. In addition to setting weight values to BEs, administrators can assign appropriate weight values to physical servers in compliance with their capabilities. The pseudo-code of VMWLC/H is shown in Figure 11 and described as follows.

First, when the FE receives a Web request from the client to establish a connection, it firstly selects the physical server with the least weighted aggregate load for all types of services (Lines 1–8). It then selects the least weighted loaded BE on that physical server that provides the requested service (Lines 9–15). The FE then forwards the request to the selected BE for processing (Line 16). After completion, the results will be directly sent back to the client. That is, assume that there are p physical servers. For each physical server i, there are n VMs (i.e., BEs) built on it. Let W_i be the weight value assigned to it and AC_i the number of aggregate active connections for all BEs running on it. VMWLC/H selects the physical server that has the least weighted aggregate active connections. Let server j be the one such that AC_j/W_j is the smallest. For each BE i on that selected physical server, let w_i be the weight values assigned to it and c_i the number of active connections. VMWLC/H chooses the BE that has the least weighted active connections. Let BE k be the one such that c_k/w_k is the smallest.

Algorithm : VM-based Weighted Least Connections for Hybrid Service (VMWLC/H)

Where $\boldsymbol{SVC_i}$—Virtual Service $\boldsymbol{i}$, $\boldsymbol{HOST_j}$—the physical server $\boldsymbol{j}$, $\boldsymbol{BE_k}$—Back-end Server $\boldsymbol{k}$, $\boldsymbol{HOST_{min}}$—the physical server with the least load, $\boldsymbol{KVM_i}$—the virtual server $\boldsymbol{i}$, $\boldsymbol{KVM_{min}}$—the virtual server with the least load.

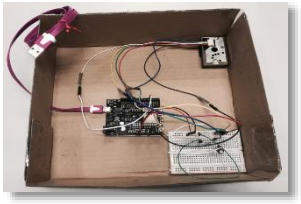
1. For each $\boldsymbol{SVC_i}$ in all Virtual Service // $\boldsymbol{i}$ is from 1 to $\boldsymbol{n}$
2. For each $\boldsymbol{BE_k}$ in $\boldsymbol{SVC_i}$'s Server Pool // $\boldsymbol{k}$ is from 1 to $\boldsymbol{m}$
3. **if** ($\boldsymbol{BE_k}$'s state == overloaded) **then**
4. continue;
5. **else**
6. calculate $\boldsymbol{BE_k}$'s load;
7. accumulate $\boldsymbol{BE_k}$'s load in the corresponding $\boldsymbol{HOST_j}$; // $\boldsymbol{j}$ is Host number and from 1 to $\boldsymbol{p}$
8. find the $\boldsymbol{HOST_{min}}$ with least ratio ($\frac{Host_j\text{'}\boldsymbol{s\ load}}{Host_j\text{'}\boldsymbol{s\ weight}}$);
9. For each $\boldsymbol{KVM_i}$ on the $\boldsymbol{HOST_{min}}$ providing this service type // $\boldsymbol{i}$ is from 1 to $\boldsymbol{q}$
10. **if** ($\boldsymbol{KVM_i}$'s state == overloaded) **then**
11. continue;
12. **else**
13. calculate $\boldsymbol{KVM_i}$'s load;
14. get $\boldsymbol{KVM_i}$'s weight;
15. find the $\boldsymbol{KVM_{min}}$ with least ratio ($\frac{KVM_i\text{'}\boldsymbol{s\ load}}{KVM_i\text{'}\boldsymbol{s\ weight}}$);
16. Forward request to $\boldsymbol{KVM_{min}}$;

Figure 11. VMWLC/H Pseudo Code.

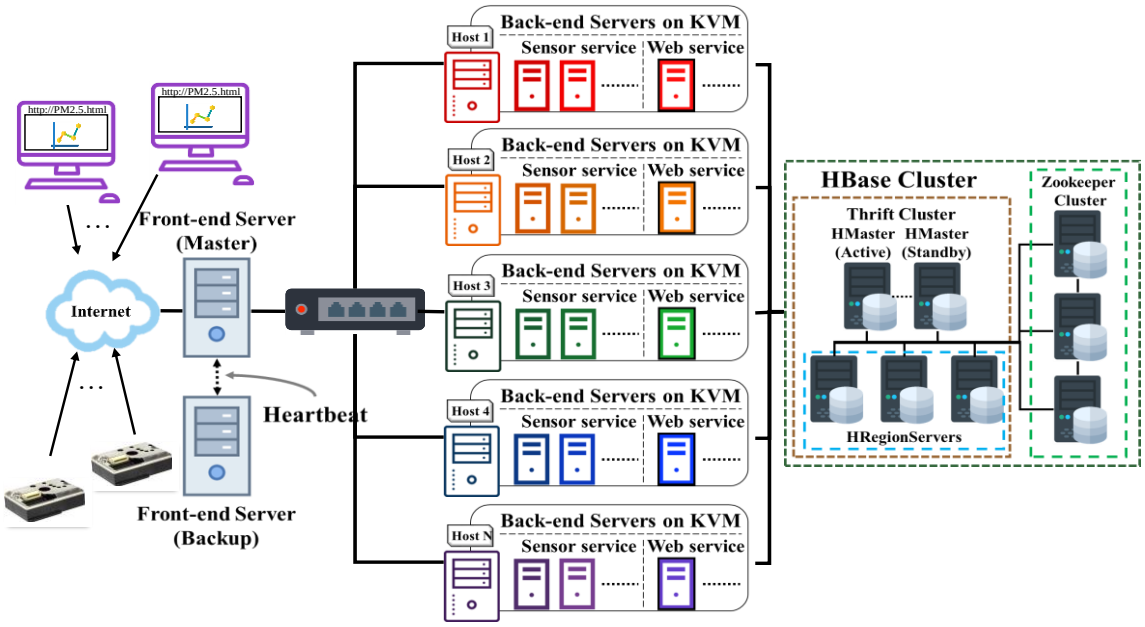
As VMWLC/H requires more calculations, for the sensor service, we use VMWLC as the load balancing algorithm in VMC/H. When the FE receives a request from a sensor, it is forwarded to the physical server with the weighted least load for sensor service, and the BE for sensor service with the least weighted connections is selected to process the request. When the FE forwards requests for Web services, it also accounts for the current load of the BEs that provide other services on the same physical server.

4. Performance Evaluation

To demonstrate the effectiveness of the proposed VMC/H system for IoT applications and the VMWLC/H for load balancing, they are practically implemented and use the air quality monitoring as an application in the experiment, as shown in Figure 12. Two services are provided, including sensor service to collect and store sensor data and Web service for browsing stored sensor data. A set of $PM_{2.5}$ sensors with wireless network connection was constructed for testing. A GP2Y1014AU0F dust sensor [27] with a WeMos D1 R2 development board [28] was used to build a $PM_{2.5}$ sensor, as shown in Figure 12a. Sensor programs were then developed, in which the sensor sends detected data periodically to the sensor service server in VMC/H. When the data are sent, the state of the connection with the sensor service server is confirmed. If the connection is lost, it is re-established, and the system then continues to detect and send sensor data.



(**a**) $PM_{2.5}$ sensor



(**b**) VMC/H system for air quality monitoring application

Figure 12. $PM_{2.5}$ sensor and VMC/H system for air quality monitoring application.

To evaluate the performance, the VMC/H system was implemented with various load-balancing strategies, such as WLC, VMWLC, and VMWLC/H. Two groups of LVS virtual services are implemented in the VMC/H system. As presented in Section 3.1, one provides sensor services, and the other provides Web services for browsing stored sensor data. We further construct two virtual server clusters that are similar in cost and system performance. This performance evaluation serves as a reference for developers when developing their systems.

Section 4.1 describes the experimental environment, and Section 4.2 describes the experimental settings. The time that is required to perform failover is measured and presented in Section 4.3. Sections 4.4–4.6 show the performance evaluation of VMC/H systems that provide sensor services, Web services, and hybrid services.

4.1. Experimental Environment

Figure 13 shows the experimental environment. Two sets of hardware devices with similar cost and system performance were used to construct VMC/H systems. LVS cluster A is constructed using multiple low-cost computers and is labeled VMC/H(M), as shown in Figure 2. The other LVS cluster B uses a Dell R730 [29] server with better performance and is labeled VMC/H(S), as shown in Figure 6. Two computers with the same specification were used to construct the FEs to provide the fault tolerance feature. In LVS cluster A and LVS cluster B, 20 VMs are deployed as BEs. 15 VMs are configured to provide sensor services, and the other 5 VMs offer Web services. Five clients are used, and these run the benchmark software Siege [30] to send Web requests to the VMC/H system. They also execute the sensor simulation program that we wrote to send sensor data to the VMC/H system. The software and hardware specifications are shown in Tables 1–3.

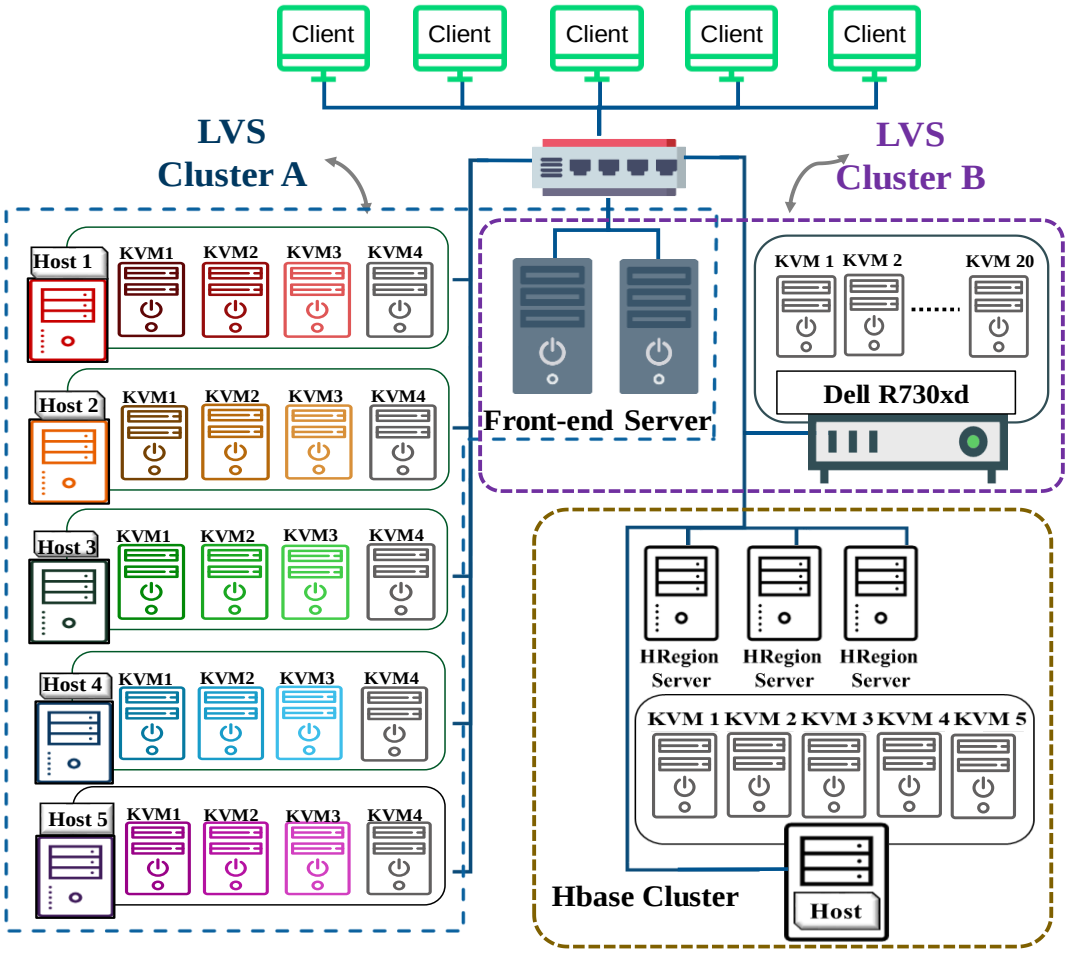


Figure 13. Experimental Environment.

Table 1. Software and hardware specifications for Cluster A.

	Front-End Server	Back-End Server			
		Host 1, 2	Host 3, 4, 5	Sensor Service VM	Web Service VM
Processor	Intel i5-4590 CPU @3.3 GHz	Intel i5-4460 CPU @3.2 GHz	Intel i5-4440 CPU @3.1 GHz	vCPU	
Memory	DDR3 8 GB	DDR3 16 GB		vMemory 4 GB	
Hard Disk	2× WD Blue 1 TB	WD Blue 1 TB		vHard Disk	
OS	Ubuntu 16.04 Server				Windows 7
Linux Kernel	4.4.1/4.4.0	4.4.0			-
Software	IPVS 1.2.1	-		Java 1.8	IIS7.5/C#/.Net
Quantity	2	2	3	-	-
Number of VMs	-	-	-	15	5

Table 2. Software and hardware specifications for Cluster B.

	Front-End Server	**Back-End Server**		
		Dell R730xd Server	**Dell R730xd Sensor Service VM**	**Dell R730xd Web Service VM**
Processor	Intel i5-4590 CPU @3.3 GHz	Intel Xeon CPU E5-2620 v4 @2.1 GHz	vCPU	
Memory	DDR3 8 GB	DDR4 80 GB	vMemory 4 GB	
Hard Disk	2× WD Blue 1 TB	5× SAS 300 GB	vHard Disk	
OS	Ubuntu 16.04 Server			Windows 7
Linux Kernel	4.4.1/4.4.0	4.4.0		-
Software	IPVS 1.2.1	-	Java 1.8	IIS7.5/C#/.Net
Quantity	2	1	-	-
Number of VMs	-	-	15	5

Table 3. Software and hardware specifications for the Client and HBase Cluster.

	HBase Cluster			**Client**
	ASUS TS500-E8-PS4	**ASUS TS500-E8-PS4 VM**	**HRegion Server**	
Processor	Intel Xeon E5-2620 V3 @2.4 GHz	vCPU	Intel i5-3470 CPU @3.2 GHz	Intel i5-3470 CPU @3.2 GHz
Memory	DDR4 32 GB	vMemory 4 GB	DDR3 4 GB	DDR3 4 GB
Hard Disk	WD Blue 1 TB	vHard Disk	Seagate 500 GB	Seagate 750 GB
OS	Ubuntu 16.04 Server			
Linux Kernel	4.4.0			
Software	-	Java 1.7/ Hadoop 2.7.2/HBase 1.1.2		Siege 3.0.8/ Java 1.8
Quantity	1	-	3	5
Number of VMs	-	5	-	-

When constructing the HBase cluster for data storage, we consider the efficiency of data access. 5 VMs were constructed on an ASUS TS500-E8-PS4 server [31]. Two of these were constructed as HMaster servers, and the other three were constructed as Zookeeper servers. The HRegion servers are mainly responsible for data storage, so they are constructed on three physical servers, instead of VMs on one physical server, to increase the speed at which data are accessed.

The operating system is Ubuntu 16.04 server edition, and LVS clustering software is IPVS [6]. We modify the core of the IPVS kernel module to implement the front-end server with VMWLC/H. Each VM is constructed using Kernel-based Virtual Machine (KVM) [8] technology. The Hadoop [9] and HBase [10] are used to provide sensor data storage.

4.2. Experimental Settings

To evaluate the performance impact of different load distribution algorithms, four VMC/H systems implemented with different load distribution algorithms were measured. As shown in Table 4, one executes WLC in VMC/H(S) systems constructed using a single physical server. VMWLC and VMWLC/H are not used because they target systems with multiple physical servers.

Table 4. Various experimental systems.

Notation	Description		
	Request Distribution Methods	**System Types**	**Note**
(A) WLC_VMC/H(S)	WLC	VMC/H(S)	20 VMs running on a single physical server
(B) WLC_VMC/H(M)	WLC	VMC/H(M)	20 VMs running on 5 physical servers
(C) VMWLC_VMC/H(M)	VMWLC		
(D) VMWLC/H _VMC/H(M)	VMWLC/H		

To evaluate the performance of VMC/H systems when providing different types of services, three different experimental workloads, including sensor requests only, Web requests only, and hybrid requests, are used in the experiments, as shown in Table 5. To serve only a single type of service, VMC/H(M) systems constructed using multiple physical servers execute VMWLC [2] or WLC [7]. VMWLC/H is used only for the VMC/H(M) that provides hybrid services.

Table 5. Various experimental workloads for evaluating various experimental systems.

Workloads	Experimental Systems	System Types	Request Distribution Methods
(A) Sensor service only	WLC_VMC/H(S)	VMC/H(S)	WLC
	WLC_VMC/H(M)	VMC/H(M)	WLC
	VMWLC_VMC/H(M)		VMWLC
(B) Web service only	WLC_VMC/H(S)	VMC/H(S)	WLC
	WLC_VMC/H(M)	VMC/H(M)	WLC
	VMWLC_VMC/H(M)		VMWLC
(C) Hybrid services	WLC_VMC/H(S)	VMC/H(S)	WLC
	WLC_VMC/H(M)	VMC/H(M)	WLC
	VMWLC_VMC/H(M)		VMWLC
	VMWLC/H _VMC/H(M)		VMWLC/H

To perform a stress test for various sensor connections, a large number of $PM_{2.5}$ sensors must be deployed. We then use Java and HBase Client API to write a multi-threaded sensor program that simulates various sensors to send sensor data to the server for storage periodically. The sensor program and sensor data format refer to the air pollutant miniature sensor system [32]. When the sensor data are received, the sensor service server stores it in the HBase cluster. The time between when all sensor service servers write the received sensor data to the HBase cluster and when all the sensor data are stored in it was recorded. Table 6 shows the sensor data format stored in the HBase.

Table 6. The sensor data format.

RowKey (IP:Port-Serial No.)	Timestamp	PM1	PM2.5	PM10	Humidity	Temperature	Illuminance	IP:Port
10.32.66.14:56002–00001	1505283500969	65	15	42	56	28	1440	10.32.66.14:56002
10.32.66.13:56004–00001	1505283504856	78	77	95	55	29	1655	10.32.66.13:56004
10.32.66.12:46880–00001	1505283505940	46	98	65	21	30	1600	10.32.66.12:46880
10.32.66.11:46880–00001	1505283507752	44	67	21	41	26	2000	10.32.66.11:46880
10.32.66.10:36540–00001	1505283500974	11	39	55	75	25	1950	10.32.66.10:36540

To avoid the server's memory cache affecting the performance evaluation, all machines were restarted before each experiment was started. Each test was conducted five times, and the average value is used to calculate the performance.

4.3. Measurement of Failover Time

This experiment verifies the fault tolerance feature for the system. It tests whether the proposed VMC/H system allows the system to recover and continue to provide services when the FE fails. This experiment uses a test time of 60 s after the sensor is turned on. The master FE is turned off at the 30th second, and the sensor is turned off at the 60th second. During the test, the sensor data per second sent by the sensor was measured on the server-side. The experimental results are shown in Figure 14. When the FE fails, the backup server recovers the system and continues to provide services in 13 s. This indicates that the backup FE successfully replaces the master, and the system successfully continues to provide services even if the master FE fails.

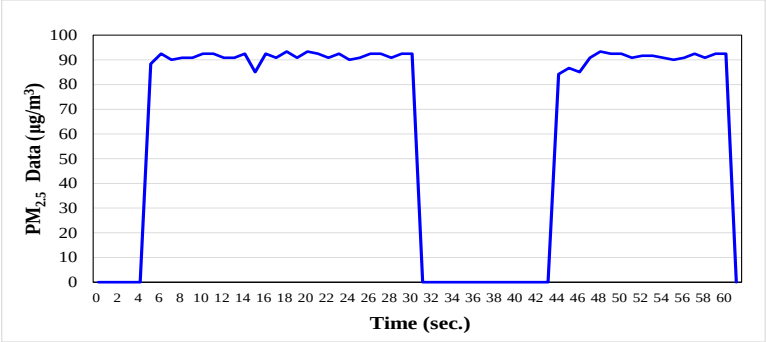


Figure 14. Time Measurement for Failover.

4.4. Performance Evaluation for VMC/H Systems Providing Sensor Services

This section presents the evaluation of performance and scalability for the VMC/H system that provides sensor services. The system performance under various amounts of sensor connections and the scalability under various amounts of VMs and physical servers are measured.

4.4.1. Performance Evaluation for Various Numbers of Sensor Connections

In this experiment, five clients are used to simulate 150, 250, 350, and 450 sensors that send sensor data at the same time. The test time is 5 min. During this time, sensors send data every 5 s; therefore, a total of 9000, 15,000, 21,000, and 27,000 sensor data are sent to the sensor service server.

The experimental results are shown in Figure 15. The average data write latency for VMC/H(S) is 261.67 s, about twice as long as that for VMC/H(M). The load is too high for a single physical server, so the VM-based sensor servers running on it have significantly reduced performance. Among them, the VMC/H(M) system using VMWLC obtains the best performance. This shows that using multiple physical servers to provide services can achieve much better performance. Using VMWLC that differentiates physical servers from virtual servers helps maintain load balance between servers, which can further increase performance. However, as the number of sensors increases more than the VMC/H systems can offer, the time delay of writing data increases significantly. Since VMC/H(S) uses only a single server, it has an upper bound for the performance limit. In contrast, VMC/H(M) can further increase performance by adding more physical servers, as shown in the scalability test for various numbers of physical servers in Section 4.4.3.

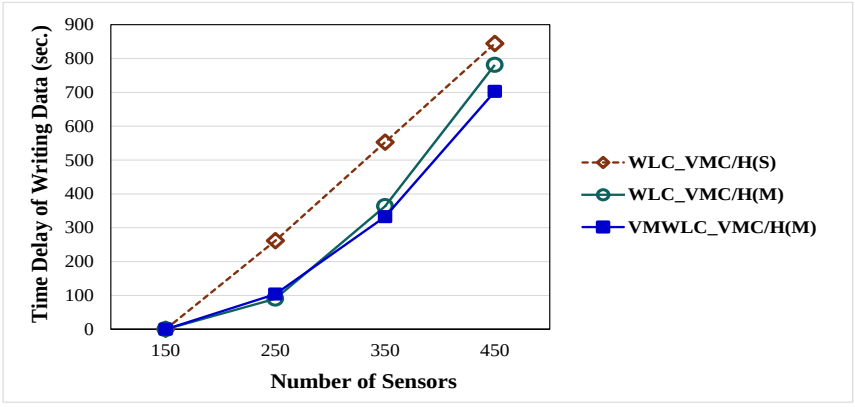


Figure 15. Various Numbers of Sensor Connections.

4.4.2. Scalability Test for Various Numbers of VMs

We have designed four test cases for the scalability test, as shown in Table 7, to determine how many sensors this VMC/H system can support and the stability of the VMC/H system for different numbers of VMs. The delay time for the sensor data to be written to the HBase cluster was measured. Figure 16 shows the experimental result for different numbers of VMs.

Table 7. Test cases for scalability test.

Test Cases	Number of VMs Providing Sensor Services	Number of Sensors
1	1	15
2	5	75
3	10	150
4	15	225

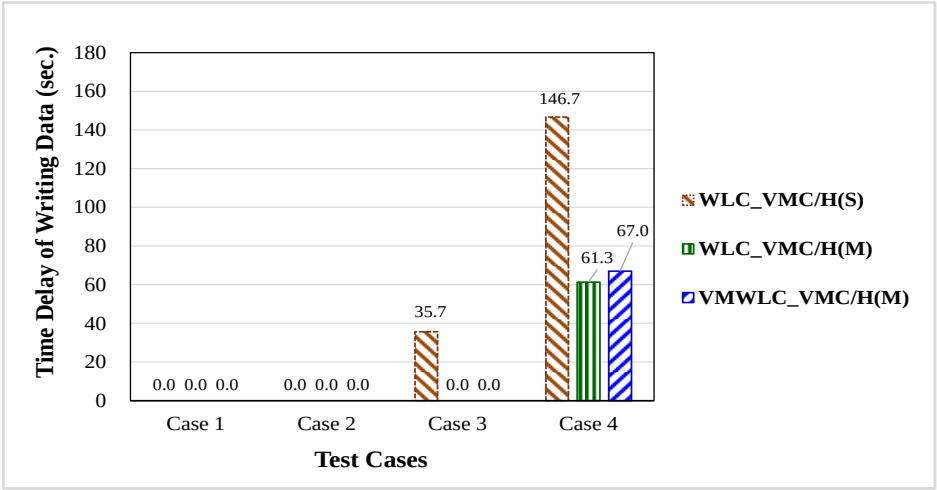


Figure 16. Scalability Test for Sensor Service for Different Numbers of VMs.

The result shows that a VM can support 15 sensors in test case 1 and without time delay of writing data. When the number of sensors increases to 75 and 5 VMs are deployed in test case 2, each sensor's data can still be processed in real time. The VMC/H(S) system begins to experience a delay in writing data in test case 3, whereas the VMC/H(M) systems begins to experience a delay in writing data in test case 4. Their delay time is much shorter than that of the VMC/H(S).

As the number of sensors is increased, the load on VMC systems increases as well. Because only one physical server in the VMC/H(S) runs all VMs and the physical server is heavily loaded, so the

entire system performance is reduced. In the VMC/H(M), since five physical servers share the load, so its performance is better than VMC/H(S) even under heavy load (i.e., test cases 3 and 4). Besides, all the data from each sensor is processed in real time in test cases 1–3.

Although increasing the number of VMs helps serve more amounts of sensors, the time delay of writing data increases as more amounts of sensors are deployed. When large amounts of sensors are deployed, adding more physical servers helps reduce the time delay, as shown in Section 4.4.3, which transforms VMC/H(S) to VMC/H(M).

4.4.3. Scalability Test for Various Numbers of Physical Servers

A scalability test was performed for the VMC/H(M) system that is constructed using different numbers of physical servers to determine how the delay time in writing data changes with the number of physical servers. In this experiment, 3 VMs that provide sensor services were established on each physical server, and five clients were used to simulate 225 sensors to send sensor data to the server simultaneously. The test ran for 5 min. Sensor data were sent every 5 s, and a total of 13,500 sensor data were sent to the sensor service server.

Figure 17 shows the experimental results that no matter which load-balancing algorithm is used, VMC/H(M) reduces the latency for writing data by adding physical servers. Adding every additional physical server in the server cluster reduces the delay time for writing data by an average of 45.0–46.4%. These data serve as a reference for administrators to extrapolate the effect when expanding the system to serve more sensors.

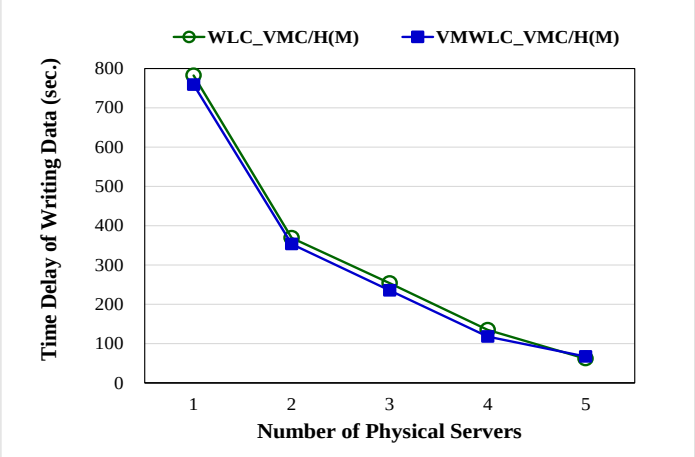


Figure 17. Scalability Test for Sensor Service for Various Numbers of Physical Servers.

4.5. Performance Evaluation for Providing Web Services

To test the website performance for servers, a dynamic Web page that accesses the latest sensor data every 5 s was established on servers. A total of 9000 pieces of data from 150 sensors were pre-created in the HBase cluster. When users query real-time sensor data using a Web browser, they query the latest sensor data of 150 sensors from the 9000 data and display them on the Web page. Five clients ran Siege [30], which simulates a total of 500 users requesting the Web service. We test three scenarios: clients sent 5000, 7500, and 10,000 Web requests for the test. The throughput was measured.

Figure 18 shows that when clients send 10,000 requests, VMC/H(M) outperforms VMC/H(S) by 41.4–55.2% in terms of throughput. VMWLC performs 9.8% better than WLC. The performance data of VMC/H systems providing Web services is similar to that of VMC/H systems providing sensor services, as shown in Section 4.4.1. VMC/H(S) is constructed using a single physical server. When too many requests are received, the physical server is overloaded, and the bandwidth of the network interface is limited, so the overall system performs poorly. VMC/H(M) using multiple physical servers to share

the load and using VMWLC that balances the loads between physical servers and virtual servers has the best performance.

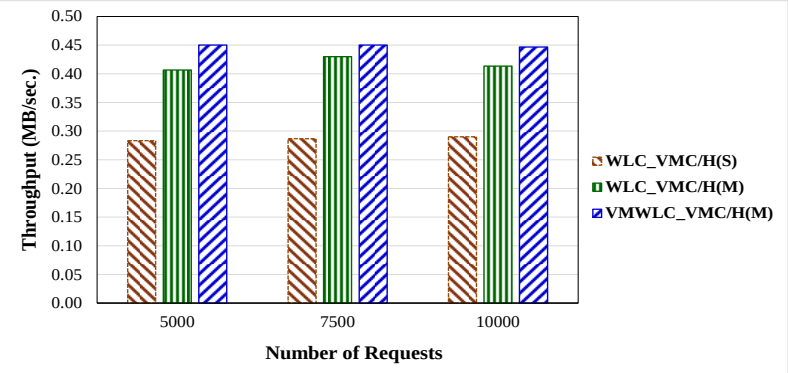


Figure 18. Performance Comparison for VMC/H Systems Providing Web Services.

4.6. Performance Evaluation for VMC/H Systems Providing Hybrid Services

The proposed VMWLC/H algorithm is mainly designed for a virtual server cluster that provides hybrid services, such as the proposed VMC/H system that simultaneously provides sensor services and Web browsing services. In this experiment, five clients are used to simulate 225 sensors that send sensor data, and they run Siege [30] that simulates a total of 5000 users requesting Web service at the same time. We tested three scenarios; the tests last for 5, 10, and 15 min. During these periods, sensors send data every 5 s, so a total of 13,500, 27,000, and 40,500 sensor data are sent to the sensor service server. The longer the test period, the more sensor data are received and stored in the HBase cluster, which affects the performance of querying sensor data through the Web browsing service.

The experimental results for VMC/H systems that use different load-balancing algorithms are shown in Figure 19. Among them, VMC/H(M) with VMWLC/H has the best performance. For VMC/H(M) system, VMWLC/H performs 3.6–9.1% better than VMWLC and 8.2–26.3% better than WLC. Furthermore, the VMC/H(M) system that uses VMWLC/H performs 35.3–62.8% better than VMC/H(S) system that uses WLC.

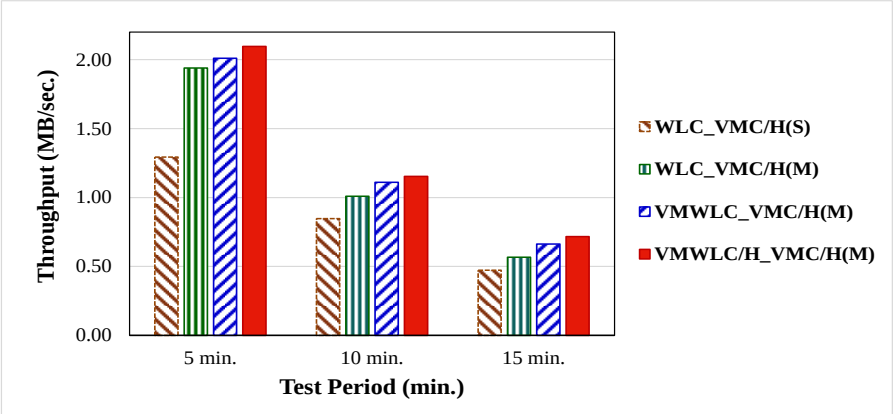


Figure 19. Performance Comparison for VMC/H Systems Providing Hybrid Services.

These results show that VMC/H(M) is more effective than VMC/H(S) because constructing all the VMs on a single physical server in VMC/H(S) results in an excessive load on the overall system. The proposed VMWLC/H algorithm considers the aggregate loads of all BEs even though they provide different services on the same physical server. Therefore, loads can be distributed to physical servers in a more balanced way. Since the overall server load is more balanced, system performance is increased.

5. Conclusions

For IoT devices and applications such as air quality monitoring that send data periodically and continuously for a long period, the system should be highly scalable and highly available to meet the increasing service demands and process and store sensor data efficiently. To efficiently provide hybrid services such as storing and browsing sensor data, this study takes advantage of several open-source software to construct the proposed VMC/H system and describes our experience in utilizing them and improving them to achieve high scalability, high performance, and high availability. Virtualization technology is used to construct server clusters, which allows easier management and maintenance of server clusters, and server clusters have the advantages of high scalability, high reliability, and high efficiency.

The VMC/H system's implementation uses LVS clustering technology, but traditional LVS load-balancing algorithms do not differentiate VMs from physical machines. They also do not consider the current loads of back-end servers that provide different services on the same physical servers, which can result in an uneven load on physical servers and a decrease in the overall system's performance. Therefore, this study designs and implements a new load distribution algorithm for VM-based server clusters that provide multiple types of services. When the front-end server forwards requests for Web services, it considers the aggregate loads of all back-end servers on all physical servers, even if they provide different services, such as sensor services and Web services. Besides, it differentiates VMs from physical machines for balancing loads between physical servers.

To demonstrate the effectiveness of the proposed VMC/H system for IoT applications and the VMWLC/H for load balancing, they are practically implemented and use air quality monitoring as the experimental application. To meet the diverse needs of users, the VMC/H system is designed to offer sensor services for collecting and storing sensor data and Web service for accessing real-time sensor data at the same time. The experimental results show that VMWLC/H helps balance the load over servers and increase performance.

Nowadays, IoT applications are becoming more and more diverse; service providers deploy more and more sensors to obtain more accurate data. With VMWLC/H, adding additional physical servers into VMC/H system can effectively distribute the load over servers and improve system performance. Our VMC/H system with VMWLC/H can serve as a reference for developers when developing their systems for IoT applications.

Currently, the VMC/H system is constructed using LVS cluster technology, KVM virtualization technology, HDFS, and Hbase distributed non-relational database. Our future work will compare its performance with that of a VMC/H system using software-defined networking [16–18] to design virtual server clusters. We will also study the use of other database technologies for sensor data storage, such as Cassandra [33], to increase the performance of the overall system. Besides, deploying the VMC/H system in the practical fields and using real workloads for performance evaluation helps subsequent work in improving system performance.

Author Contributions: Methodology, M.-L.C. and T.-T.H.; software, T.-T.H.; writing—original draft preparation, M.-L.C. and T.-T.H.; writing—review and editing, M.-L.C.; project administration, M.-L.C.; funding acquisition, M.-L.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Ministry of Science and Technology, Taiwan, grant number 107-2221-E-260-005 and 108-2221-E-260-005.

Acknowledgments: We would like to thank Jia-Jung Chung for her assistance in draft preparation. Special thanks to the anonymous reviewers for their valuable comments in improving this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shi, X.; Jin, H.; Jiang, H.; Pan, X.; Huang, D.; Yu, B. Toward scalable Web systems on multicore clusters: Making use of virtual machines. *J. Supercomput.* **2012**, *61*, 27–45. [CrossRef]

2. Chang, J.H.; Cheng, H.S.; Chiang, M.L. Design and Implementation of Scalable and Load-Balanced Virtual Machine Clusters. In Proceedings of the 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Kanazawa, Japan, 22–25 November 2017; pp. 40–47.
3. Google Cloud Load Balancing. Available online: https://cloud.google.com/load-balancing/ (accessed on 10 August 2020).
4. AWS Elastic Load Balancing. Available online: https://aws.amazon.com/elasticloadbalancing/ (accessed on 10 August 2020).
5. Microsoft Azure Load Balancer. Available online: https://azure.microsoft.com/en-au/services/load-balancer/ (accessed on 10 August 2020).
6. The Linux Virtual Server Project-Linux Server Cluster for Load Balancing. Available online: http://www.linuxvirtualserver.org (accessed on 10 August 2020).
7. Zhang, W. Linux virtual server for scalable network services. In Proceedings of the Ottawa Linux Symposium, Ottawa, ON, Canada, 13–16 July 2010.
8. Kernel-Based Virtual Machine. Available online: http://www.linux-kvm.org/page/Main_Page (accessed on 10 August 2020).
9. Hadoop. Available online: http://hadoop.apache.org/ (accessed on 10 August 2020).
10. HBase. Available online: http://hbase.apache.org/ (accessed on 10 August 2020).
11. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
12. Zookeeper. Available online: https://zookeeper.apache.org/ (accessed on 10 August 2020).
13. Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W.C.; Wallach, D.A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R.E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst. (TOCS)* **2008**, *26*, 4. [CrossRef]
14. Thrift. Available online: https://thrift.apache.org/ (accessed on 10 August 2020).
15. Zhou, W.; Yang, S.; Fang, J.; Niu, X.; Song, H. VMCTune: A Load Balancing Scheme for Virtual Machine Cluster Based on Dynamic Resource Allocation. In Proceedings of the 2010 9th International Conference on Grid and Cloud Computing, Nanjing, China, 1–5 November 2010; pp. 81–86. [CrossRef]
16. Kreutz, D.; Ramos, F.M.V.; Veríssimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
17. Neghabi, A.A.; Navimipour, N.J.; Hosseinzadeh, M.; Rezaee, A. Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. *IEEE Access* **2018**, *6*, 14159–14178. [CrossRef]
18. Chiang, M.L.; Cheng, H.S.; Liu, H.Y.; Chiang, C.Y. SDN-based server clusters with dynamic load balancing and performance improvement. *Clust. Comput.* **2020**. [CrossRef]
19. Ghanbari, Z.; Navimipour, N.J.; Hosseinzadeh, M.; Darwesh, A. Resource allocation mechanisms and approaches on the Internet of Things. *Clust. Comput.* **2019**, *22*, 1253–1282. [CrossRef]
20. Rasyid, M.U.H.A.; Pranata, A.A.; Lee, B.F.; Saputra, F.A.; Sudarsono, A. Portable electrocardiogram sensor monitoring system based on body area network. In Proceedings of the IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Nantou, Taiwan, 27–29 May 2016.
21. MySQL. Available online: https://www.mysql.com/ (accessed on 10 August 2020).
22. Rasyid, M.U.H.A.; Yuwono, W.; Muharom, S.A.; Alasiry, A.H. Building platform application big sensor data for e-health wireless body area network. In Proceedings of the 2016 International Electronics Symposium (IES), Denpasar, Indonesia, 29–30 September 2016; pp. 409–413.
23. Hive. Available online: https://hive.apache.org/ (accessed on 10 August 2020).
24. Zhang, M.; Liu, Q.; Zheng, W.; Wan, K.; Hu, F.; Yu, K. Utilizing cloud storage architecture for long-pulse fusion experiment data storage. *Fusion Eng. Design* **2016**, *112*, 1003–1006. [CrossRef]
25. MongoDB. Available online: https://www.mongodb.com/ (accessed on 10 August 2020).
26. Linux-Heartbeat. Available online: http://www.linux-ha.org/wiki/Heartbeat (accessed on 10 August 2020).
27. GP2Y1014AU0F Dust Sensor. Available online: http://www.sharp-world.com/products/device/lineup/selection/opto/dust/index.html (accessed on 10 August 2020).
28. Arduino. Available online: https://www.arduino.cc/ (accessed on 10 August 2020).
29. Dell PowerEdge Rack Servers. Available online: https://www.dell.com/en-us/work/shop/poweredge-rack-servers/sf/poweredge-rack-servers (accessed on 10 August 2020).

30. Siege. Available online: https://www.joedog.org/siege-home/ (accessed on 10 August 2020).
31. ASUS TS500-E8-PS4 Server. Available online: https://www.asus.com/Commercial-Servers-Workstations/TS500E8PS4/ (accessed on 10 August 2020).
32. Real-Time Map of Local Air Quality. Available online: https://www.airq.org.tw/ (accessed on 10 October 2020).
33. Cassandra. Available online: http://cassandra.apache.org/ (accessed on 10 August 2020).