applied sciences

MDPI

Article

Design and Implementation of an Automated Disaster-Recovery System for a Kubernetes Cluster Using LSTM

Ji-Beom Kim, Je-Bum Choi and Eun-Sung Jung *

Department of Software & Communications Engineering, Hongik University, Sejong 30016, Republic of Korea; kjbeom@g.hongik.ac.kr (J.-B.K.); b989061@g.hongik.ac.kr (J.-B.C.)
* Correspondence: ejung@hongik.ac.kr

Abstract: With the increasing importance of data in modern business environments, effective data management and protection strategies are gaining increasing research attention. Data protection in a cloud environment is crucial for safeguarding information assets and maintaining sustainable services. This study introduces a system structure that integrates Kubernetes management platforms with backup and restoration tools. This system is designed to immediately detect disasters and automatically recover applications from another Kubernetes cluster. The experimental results show that this system executes the restoration process within 15 s without human intervention, enabling rapid recovery. This, in turn, significantly reduces the potential for delays and errors compared to manual recovery processes, thereby enhancing data management and recovery efficiency in cloud environments. Moreover, our research model predicts the CPU utilization of the cluster using Long Short-Term Memory (LSTM). The necessity of scheduling through this predict is made clearer through comparison with experiments without scheduling, demonstrating its ability to prevent performance degradation. This research highlights the efficiency and necessity of automatic recovery systems in cloud environments, setting a new direction for future research.

Keywords: data protection; automatic recovery; kubernetes; cluster; LSTM

Citation: Kim, J.-B.; Choi, J.-B.; Jung, E.-S. Design and Implementation of an Automated Disaster-Recovery System for a Kubernetes Cluster Using LSTM. *Appl. Sci.* **2024**, *14*, 3914. https://doi.org/10.3390/app14093914

Academic Editor: Chilukuri K. Mohan

Received: 23 January 2024
Revised: 8 April 2024
Accepted: 1 May 2024
Published: 3 May 2024

1. Introduction

In the modern era, data have become a crucial asset and a key to competitiveness for businesses. With the expansion of digitalization and cloud technology, data continues to increase in value. In this context, data loss poses a significant threat to businesses, highlighting the need for efficient data management and protection strategies. Data protection in cloud environments is vital, enabling companies to respond to various threats such as software errors, hardware failures, cyber-attacks, and natural disasters. The maintenance of data stability and reliability has emerged as a key element of business operations.

The importance of data backup is emphasized in this context. Data backup transcends mere information copying; it involves various functions from fulfilling legal obligations to protecting against security threats, such as ransomware and hacking, and preparing for natural disasters [1–4]. Data backup strategies, such as the setting of recovery time objectives (RTOs) and recovery point objectives, play a crucial role in minimizing service disruptions and preventing data loss. These strategies are essential for organizations to protect their information assets from various risks and maintain sustainable services.

The significance of disaster recovery is an extension of data backup strategies. Disaster recovery provides more comprehensive security and stability in conjunction with data backup. Disaster-recovery solutions respond quickly and effectively to data loss or damage due to natural disasters, technical errors, and cyber-attacks. Such strategies ensure quick recovery and the provision of sustainable services, thus guaranteeing business continuity. Companies operating various applications and services in cloud environments must implement advanced disaster-recovery strategies.

Disaster Recovery, as a Service (DRaaS), supports the quick recovery of data and applications in the event of a disaster [5]. This service is applied in various environments, and its importance has been increasingly highlighted in cloud usage. Cloud-based businesses and organizations require effective disaster-recovery strategies to ensure data and service continuity. DRaaS is designed to meet this requirement by considering the flexibility and scalability of cloud infrastructure to minimize business disruptions in the event of a disaster. DRaaS in cloud environments offers many advantages, including cost efficiency, fast recovery times, and accessibility, making it an essential element in modern business environments. Consequently, research on disaster recovery in cloud environments is being actively pursued [6–10].

Backup and restoration tools for Kubernetes are crucial for protecting the data and system state of container-based applications. These tools play an important role in rapidly restoring important data and configurations in the event of a failure [11]. Their usage supports the safe backup of critical data and configurations and quick restoration in case of failure or data loss.

The Kubernetes management platform provides an environment for the integrated management of various clusters and services. With the continuous increase in the complexity of cloud services and applications, consistent cluster management and deployment have become essential. This platform enhances operational efficiency in complex environments, reduces operational burdens through resource optimization and automation, and emphasizes the need for a centralized management platform with the increase in the number of clusters. This reduces the possibility of errors and facilitates maintenance and monitoring. The Kubernetes management platform simplifies the manual management of individual clusters and supports consistent policy application, effective monitoring, and stable application deployment. This plays a significant role in enhancing the efficiency of the IT infrastructure for businesses and organizations, contributing to the achievement of business goals.

This paper presents a system that automatically recovers applications in a cluster. The system detects when a cluster loses functionality because of a disaster and automatically restores the services that were operating in the affected cluster to another cluster. Accordingly, even if a disaster occurs and the cluster loses functionality, automatic recovery is immediately initiated without the need for user intervention. Conventional disaster recovery involves delays due to human intervention [12], which can be minimized in an automated recovery, thus providing the advantage of faster recovery. Compared to manual recovery, an automated system responds immediately and quickly restores services to normal. In addition, it prevents mistakes that can occur during manual operations and ensures consistent recovery.

Our research has two technical contributions. First, it automates the entire recovery process, i.e., from event detection, including disasters and attacks, to querying backup files, selecting clusters for restoration, and executing restoration tasks. To automate all these functions, the system integrates Kubernetes management platforms with Kubernetes backup and restoration tools. Thus, user-intervention time is eliminated. This automation prevents mistakes that can occur during manual operations because of the complexity of the recovery process and ensures consistent recovery. In addition, automated recovery uses pre-allocated resources to perform tasks most optimally, thereby improving the system's overall performance and stability.

Second, when selecting a cluster for the restoration task, machine learning is used to predict the CPU usage required for selecting a cluster. While clusters can be selected using algorithms or rules, the time taken for restoration tasks must also be considered. Therefore, machine learning is used to predict the CPU usage of clusters to select the cluster for successful restoration. This minimizes service disruption with quick recovery times, positively affecting RTOs. Finally, the automated-recovery solution reduces management complexity and enhances the overall system stability by providing consistent recovery

procedures across various systems or applications. This approach prevents mistakes that can occur during manual operations and ensures consistent recovery.

The remainder of this paper is structured as follows. Section 2 reviews the current related research, highlighting the importance of recovery automation in cloud environments and presenting the unique aspects of our research. Section 3 describes the architecture of the proposed system, which integrates Kubernetes management platforms with backup and restoration tools. Section 4 introduces the data used in the experiments and explains the preprocessing methods used. Section 5 discusses the Long Short-Term Memory (LSTM) and how to train time-series data on the LSTM. Section 6 details the experimental environment and methods used. In Section 7, we present and analyze the experimental results. Finally, conclusions are provided in Section 8.

2. Related Work

Sousa et al. [13] explored two important concepts in cloud software engineering: "automatic recovery" and "job scheduling". Automatic recovery refers to the automatic restoration of services through an orchestration manager in the event of system failures, enhancing the reliability of cloud services and minimizing the response time to failures. Job scheduling involves the efficient allocation and scheduling of resources in the cloud environment, optimizing system performance, and reducing operational costs. These functionalities are essential for the stable and efficient management of the cloud infrastructure, especially in environments requiring high availability and quick recovery times. The superiority of automatic recovery over manual recovery is demonstrated through experiments that simulate various failure scenarios and measure the reduction in the service-restoration time during automatic recovery. In manual recovery, users must identify failures and decide on the type of system recovery, thus causing delays and errors. In contrast, automatic recovery precisely monitors the service status and attempts recovery automatically when the orchestration manager detects a failure, thus enhancing reliability. Container deployments, including automatic recovery, are considered to be part of the service-development process, restoring containers to a normal state after failures. The experimental results show that the orchestration manager continuously monitors the status of services and automatically restarts services upon detecting issues, thereby reducing service downtime, and enhancing system reliability, demonstrating that rapid service recovery is possible without manual intervention. This automatic recovery mechanism proves crucial in the operation of cloud-based services.

Yu et al. [14] focused on the automatic recovery of applications within aerospace ground systems, based on cloud computing. They emphasized developing and implementing recovery services to counter software errors. The core of the recovery service is the automatic recovery capability of applications, which is aimed at improving stability and availability in the cloud computing environment. They explored the technical details related to software recovery strategies and provided experimental evaluations of the recovery time and capability. In addition, measures to automate application recovery concerning software failures in cloud computing environments were designed and implemented, including strategies for various recovery scenarios. Moreover, the efficiency and performance of these strategies were validated using real experiments. Yu at al. also presented an automated approach to maintain continuous access portals and ensure business continuity after application recovery, allowing users to use the service continuously without being aware of the recovery process. Their research, which focused on the automation of application recovery in cloud environments, presented a different approach from those of previous studies. While most research [15–17] has focused on data and system recovery, this study focused on automatic recovery at the application level, offering a new direction for enhancing the stability and the availability of applications in cloud environments.

Previous studies have presented diverse approaches to automatic recovery in Kubernetes and cloud computing environments. Our research focuses on automatic recovery at the cluster level. In contrast to the research by Sousa et al. [13], which was focused on

individual services or tasks, our approach involves the conducting of automatic recovery by targeting clusters, thus restoring applications from one cluster to another. Yu et al. [14] emphasized automated application recovery in aerospace ground systems based on cloud computing. This study proposes and implements automatic recovery functions at the application level in response to software failures. It addresses application recovery strategies and experimentally evaluates recovery time and capability. Compared to the technique used in the work presented in [14], which required standby server resources, our research enhances resource-utilization efficiency by conducting recovery in operational clusters. Yu et al. [14] addressed automatic recovery from the functional loss of applications in the same environment, whereas we propose cluster-level automatic recovery, which automatically restores applications to a different cluster when the original cluster loses functionality.

In the realm of commercial cluster and cloud management, AWS Elastic Disaster Recovery (AWS DRS) [18] and Google Cloud's disaster recovery [19] methods are designed to enhance the resilience and availability of systems and applications in cloud environments. By leveraging these services, businesses can achieve minimal downtime and data loss in the event of outages or disasters, aligning with the Recovery Time Objectives (RTOs) and Recovery Point Objectives (RPOs), which are critical for business continuity management. To meet the RTOs and RPOs, AWS DRS continuously synchronizes data across multiple servers to enable rapid recovery, meeting the RTOs and RPOs in the event of a disaster. This is usually called active/active strategy because a source and target cluster are always active for real-time synchronization. However, this leads to additional resource use and increased costs. Google Cloud's disaster recovery supports similar features.

In our study, we assume a cluster environment, deploying an active/passive strategy where backup data is restored to the target cluster only when there is a problem with the source cluster. Three distinctive features are as follows: (1) source cluster data are continuously transferred to backup storage to meet RPO; (2) target clusters are not in a standby state and are not only dedicated to a source cluster restore. They may have up-and-running services. (3) This strategy does not require complex data synchronization or migration before the recovery process, thus allowing for more efficient resource utilization.

Our proposed method and module can be integrated with a snapshot-based recovery method, which creates and stores a copy of the data at a specific point in time, allowing for the system to be quickly restored when necessary. This offers the advantage of cost and resource efficiency while meeting the RTO and RPO requirements, although it may not be as efficient as real-time recovery. Nonetheless, it is very suitable for minimizing resource usage and reducing idle resources that may occur during data migration.

The choice between an active/active strategy using real-time data migration and active/passive strategy using snapshots, which is the assumed backup environment in our study, depends on several factors. Data migration supports the rapid recovery of complex systems, meeting complex situations and business continuity requirements. In contrast, the snapshot approach is more suitable for systems prioritizing cost, resource efficiency, and simplicity in the recovery process. To determine the optimal disaster recovery strategy, one must consider the system's characteristics, requirements, and cost-effectiveness. This decision-making process should also include technical indicators like the Recovery Time Objective (RTO). Therefore, for organizations aiming to minimize resource usage and maximize cost efficiency, the active/passive strategy, using our proposed method and module, can be an excellent alternative to the AWS or Google strategy. Avoiding complex data synchronization and migration while enabling quick and effective recovery when necessary is particularly suitable for scenarios prioritizing cost and operational efficiency. The AWS multi-region access point failover control is a high-availability feature that automatically switches data access requests to another AWS region within minutes. This ensures resilience and business continuity through multi-region architecture. Google's disaster recovery scenarios offer cold, warm, and real-time concurrent services depending on the RTO, but even real-time concurrent services, as well as warm sites, require standby resources. In contrast, unlike AWS and Google services, our system's active/passive

strategy does not rely on standby resources, allowing for operational services to exist, which differentiates it.

3. Design

This section explains the structure of the automatic recovery system, implemented by integrating Kubernetes backup and restoration tools with the Kubernetes management platform. This system automatically transfers the applications of a cluster to another cluster if the original cluster experiences a disaster and loses its connection to the Kubernetes management platform. The automatic recovery system is added to the cluster state, monitoring part of the Kubernetes management platform, and integrates by installing Kubernetes backup and restoration tools in the environment.

The proposed automatic recovery system operates on the Kubernetes management platform and continuously monitors the state of a cluster. In the event of a disaster, the system is capable of monitoring and detecting events to identify the situation. Once a disaster is detected, the system performs resource comparison by comparing the resources of the affected cluster with those of other clusters to identify a cluster with superior resources. Subsequently, it verifies the name of that cluster and selects a target cluster for performing the restoration work based on the CPU usage of that cluster. Finally, the system executes the restoration process by using the backup files of the disaster-affected cluster in the selected cluster. This entire process comprises four main components, and Figure 1 shows the flowchart of the components of the proposed automatic recovery system. The structure and operation of each component are as follows.

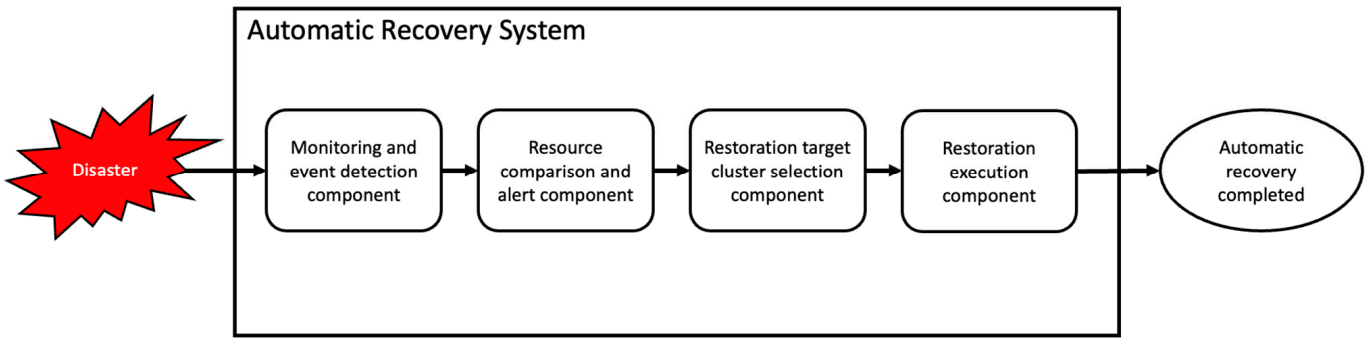

Figure 1. Flowchart of the Components of the Automatic Recovery System.

- Monitoring and event-detection component: The Kubernetes management platform monitors the clusters it manages. In this study, we enhanced this feature to detect events when a managed cluster becomes disconnected and then transmit the name of that cluster to the resource-comparison component. Figure 2 shows the flowchart of the monitoring and event-detection component.
- Resource-comparison-and-alert component: By using the name of the disaster-affected cluster transmitted by the monitoring and event-detection component, the allocated CPU core counts of that cluster and the other managed clusters are compared to check if any cluster has more CPU cores than the disaster-affected cluster. If none of the other clusters match this criterion, the restoration procedure is halted and an alert is sent to the user, as proper restoration cannot be achieved. In contrast, if clusters are found with more CPU cores than those of the affected cluster, the names of such clusters are retrieved for the restoration process and sent to the Restoration target cluster-selection component. As there can be more than one cluster with more CPU cores than the affected cluster, multiple cluster names can be transmitted to the Restoration target cluster-selection component. Figure 3 shows the flowchart of the Resource-comparison-and-alert component.

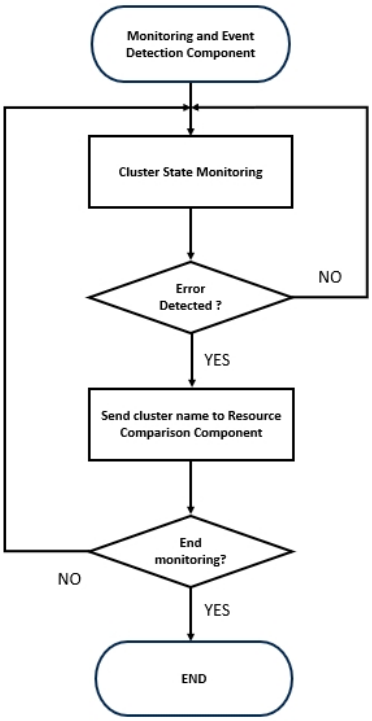

Figure 2. Flowchart of the monitoring and event-detection component.

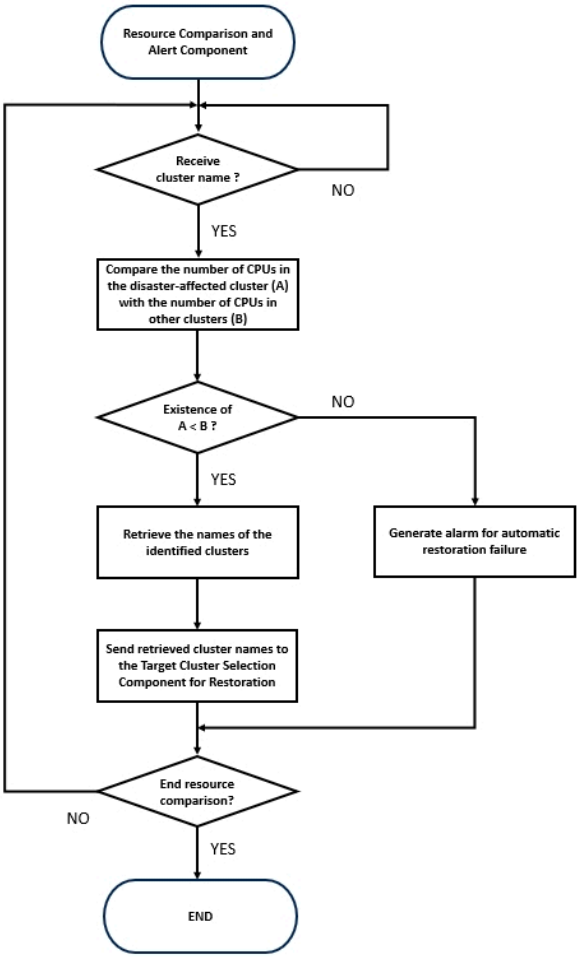

Figure 3. Flowchart of the Resource-comparison-and-alert component.

- Restoration target cluster-selection component: The names of the clusters transmitted through the Resource comparison component are used to query their current CPU-utilization rates. The queried current CPU-utilization rates of these clusters are passed to a machine-learning model as the parameters to predict CPU utilization, and the predicted CPU-utilization rates are returned. The cluster with the lowest predicted CPU-utilization rate among all predicted clusters is selected. Then, the name of the selected target cluster for restoration is transmitted to the Restoration-execution component. Figure 4 shows the flowchart of the Restoration target cluster-selection component.

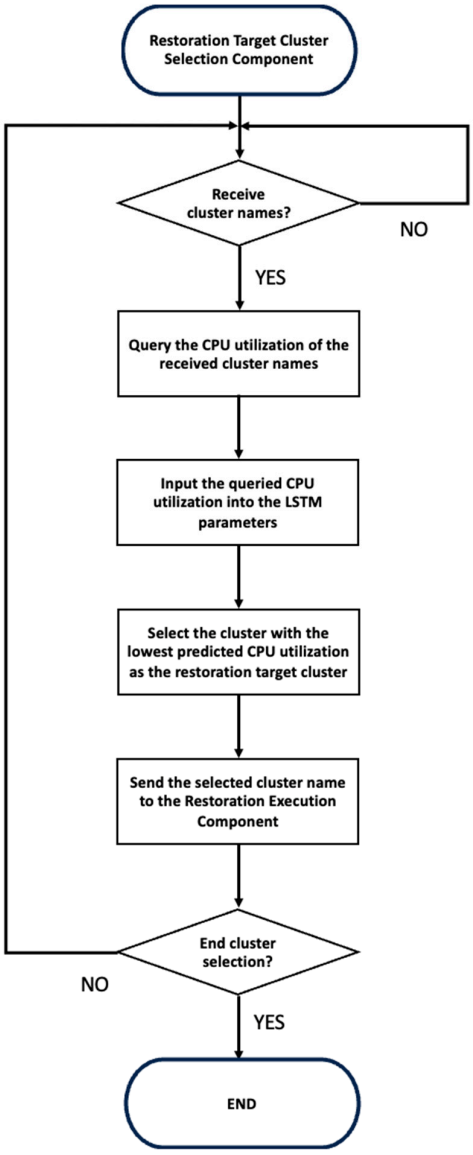

Figure 4. Flowchart of the Restoration target cluster-selection component.

- Restoration-execution component: The cluster selected as the target for restoration and transmitted through the Restoration target cluster-selection component executes the restoration command of the Kubernetes backup and restoration tool. This restoration command includes the location of the backup file of the disaster-affected cluster. When the restoration command is executed in the target cluster, the backup file is retrieved from its storage location and is used to restore applications and other components. Figure 5 shows the flowchart of the Restoration-execution component.

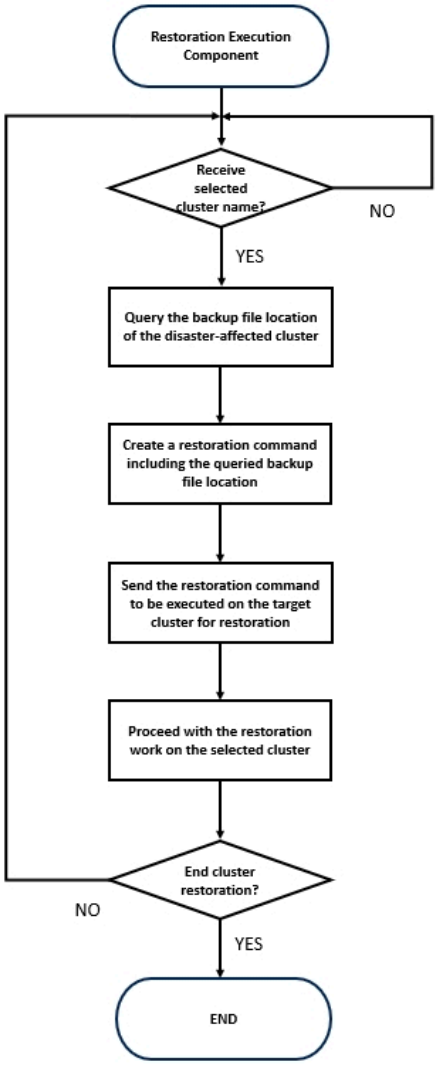

Figure 5. Flowchart of the Restoration-execution component.

4. Experimental Data

4.1. Google Cluster Trace Dataset

We used the Google Cluster Trace dataset [20], specifically the ClusterData-2011_2 version, to train our cluster-state-prediction model. This dataset records the activity of a single cluster during May 2011, encompassing approximately 12,500 machines, 650,000 jobs, and 20 million tasks. The tasks mentioned herein refer to Linux programs executable on machines. The dataset includes detailed trace information about the behavior of jobs and tasks, resource allocation, and the activities of the machines in the cluster. The dataset is categorized into various tables, including the Machine Event, Machine Attributes, Job Event, Task Event, Task Constraints, and Task Resource Usage Tables. Each table provides the following information:

- Machine Events Table: This table comprises one or more records of every machine in a cluster. Most of the records describe the machines present at the start of the trace. Event types include addition, removal, and update, and the CPU and memory capacities of each machine are standardized. The platform ID denotes the micro-architecture and the chipset version of the machine; machines with the same ID can differ in terms of clock speed or number of cores.
- Machine Attributes Table: This table comprises key-value pairs representing the machine's characteristics, including the kernel version, clock speed, and IP address. Values are expressed as strings if not in integer form, and "1" indicates a missing value.

- Job Events Table: This table includes the time, ID, type, user, and scheduling information of a job. Information about active (RUNNING) or pending (PENDING) jobs is also recorded, with each job containing scheduling constraints. The scheduling class contains the latency information of the job, and job names are provided as encrypted strings, repeated for multiple runs of the same program.
- Task Events Table: This table contains information such as timestamps, missing details, job ID, task index, machine ID, event type, username, scheduling class, priority, CPU cores, RAM, and local disk space requests. A task's priority is inversely proportional to its numerical value, with higher numbers indicating a higher priority. "Free" denotes low priority, "production" is high priority, and "monitoring" is a priority for monitoring other low-priority tasks. Resource requests indicate the maximum CPU, memory, and disk space that a task can use, and exceeding these limits can restrict the task.
- Task Resource Usage Table: This table includes information such as the start and end times of the measurement period, job ID, task index, machine ID, CPU usage, memory usage, disk I/O time, and cache usage. It contains essential data for understanding the actual resource usage in the cluster, such as the average CPU usage, normalized memory usage, average disk I/O time, and average local disk space usage.

These tables represent CPU-related resource usage data as normalized values. This normalization adjusts to a relative scale based on the maximum resource capacity of all tracked machines, with the maximum value standardized at 1.0. The CPU usage is measured in core seconds per second; for instance, if a job consistently uses two cores, the usage rate is 2.0 core seconds per second [20].

4.2. Preprocessing

According to Bi [21], the resource usage data in the Google Cluster Trace dataset is highly non-linear, exhibiting erratic and highly variable characteristics. Figure 6 visualizes this by normalizing the CPU rate data by using the min–max normalization method and representing it in 5 min intervals, showcasing the irregular values of the CPU rate.

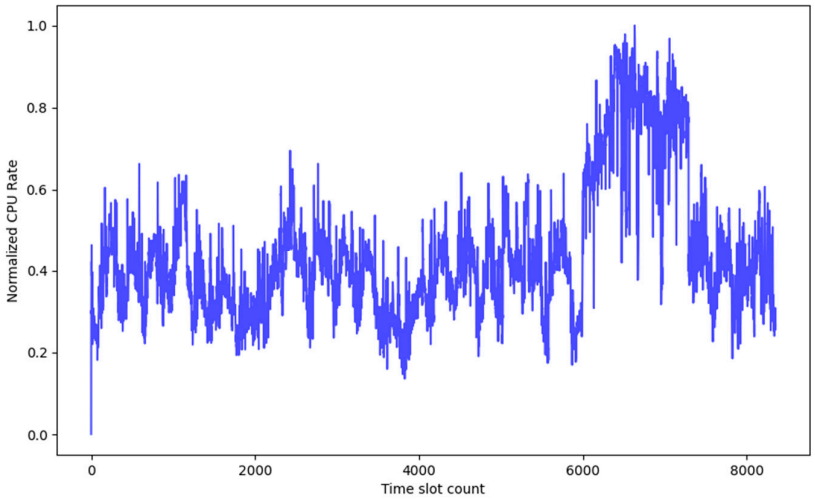

Figure 6. Time series of CPU usage.

Like previous studies focusing on CPU utilization [21,22], data preprocessing was initiated by extracting the start and end times, as well as CPU-usage data, from the task resource usage table. As the Google Cluster Trace dataset does not provide direct CPU use data for the cluster, the extracted data were aggregated. We then identified the earliest measurement start time and the latest measurement end time, creating 8352-time slots at 5 min intervals based on this range. Next, the CPU utilization for each time slot was aggregated according to the start and end times of each task. This preprocessed data was

converted to a time series, batched at 5 min intervals, and then used to analyze trends in the CPU utilization of the cluster.

5. LSTM-Based Scheduling

Where to restore is totally dependent on our scheduling strategy/policy. Our scheduling goal is to recover the services on a failed source cluster on a new target cluster "safely". Here, we assume that recovery safety is equivalent to restore to a target cluster with the "most" available resources, because the lack of resource has more chances of not satisfying SLAs (Service Level Agreements).

Typical recurrent neural networks (RNNs) may encounter challenges in capturing long-range dependencies within the data [23,24]. LSTM [25] is an advanced recurrent neural network that effectively remembers the data sequences from previous time steps for future applications. The reason for choosing LSTM is that this model integrates gates and memory lines to effectively learn both long-term and short-term dependencies in the data [26]. Due to these reasons, LSTM is commonly used in time series data prediction. In contrast to other models, LSTM excels in solving the gradient vanishing problem that occurs in long sequence data and has strengths in detecting and learning temporal dependencies [27]. With these characteristics, LSTM demonstrates robust performance in capturing complex patterns and various time intervals in time series data. Many variations to improve time-series data prediction have been proposed [28].

5.1. Sliding Window

The lookback window represents the duration of past data provided to the model, with the current time as the reference point. The model is trained and performs predictions using data within this period. For example, if the lookback window is 24 h, the model is trained and predicts based on data from the current time up to 24 h ago.

The forecasting horizon refers to the time interval into the future that the model aims to predict, with the current time as the reference point. It determines how far into the future the model intends to make predictions. For instance, if the forecasting horizon is 6 h, the model performs predictions for the timeframe from the current time to 6 h ahead.

Applying the sliding window technique to chaotic long-term time series data, such as the Google Cluster Trace Dataset, offers several advantages:

- Non-linear Behavior Detection: Chaotic data often exhibits distinct non-linear dynamic patterns. Utilizing the sliding window allows the model to capture and learn these non-linear patterns.
- Temporal Dependencies: Chaotic time series data involves crucial temporal dependencies. Sliding window considers data within specific periods, leading to a more accurate understanding of the temporal dependencies.
- Adaptability and Model Generalization: Sliding window aids the model in adapting to the dynamic nature of the data. Chaotic data can be challenging for prediction, but through the sliding window, the model can learn and generalize patterns within given periods.

We explored the optimal lookback window and forecasting horizon through experiments on the Google Cluster Trace Dataset. In conclusion, the best prediction performance was observed with a lookback window of 3 and a forecasting horizon of 1.

5.2. Prediction Model Architecture

Figure 7a illustrates the process of the LSTM model learning time series data. It depicts the process of constructing and predicting time series data using the LSTM model. Initially, the time series data is normalized to the [0, 1] range through Min–Max scaling. Subsequently, the data is divided into X values and Y values by setting the Lookback Window and Forecasting Horizon of the sliding window. Here, X values correspond to data from past times, while Y values correspond to data for future times. The data is then split into training and testing sets through Data Split, with a split ratio of 0.2. The LSTM

sequentially receives X values from the training data, and Y values are used as labels in supervised learning. The LSTM model consists of LSTM layers and a Dense layer. The first LSTM layer has an input shape of (None, 3, 128), which means the input sequence length is 3 and there are 128 features at each time step. This LSTM layer has 128 units, and the total number of parameters is 66,560. The second LSTM layer takes the output of the previous LSTM layer as its input, with an output shape of (None, 128). This LSTM layer has 128 units, and the total number of parameters is 131,584. Finally, a Dense layer follows, with an output shape of (None, 1). This layer has one unit, and the total number of parameters is 129.

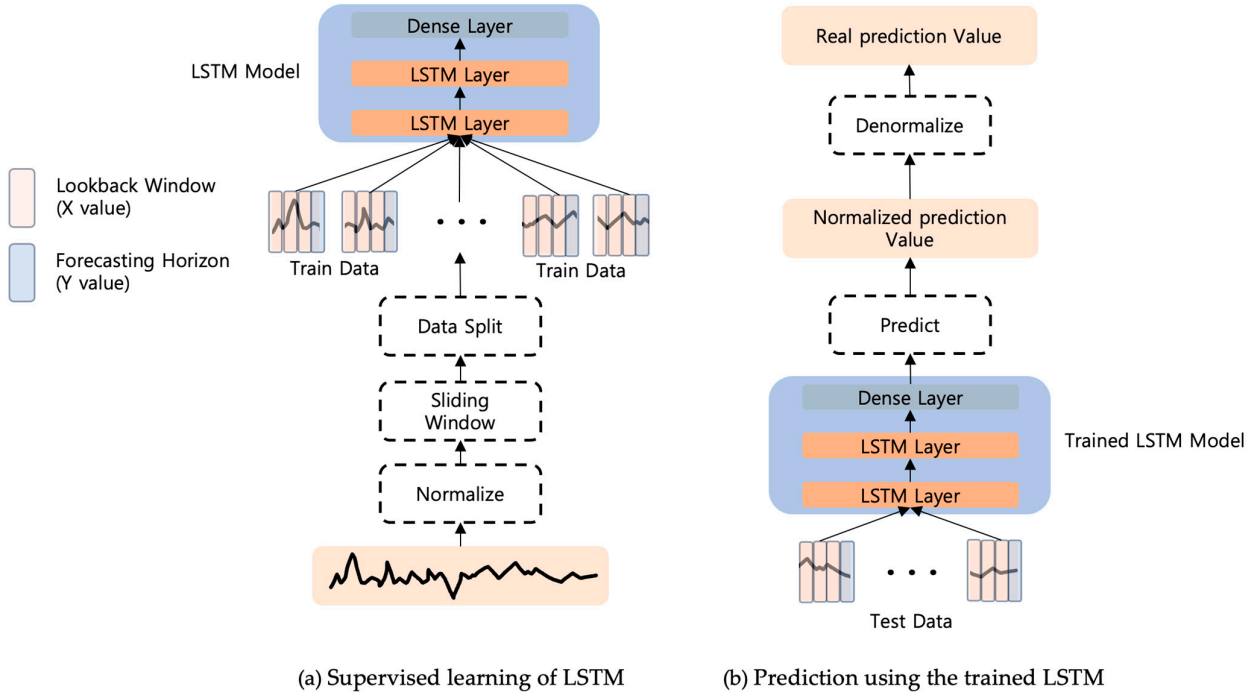

Figure 7. LSTM workflow. (**a**) Supervised learning of LSTM with normalization, sliding window, data split; (**b**) Prediction using the trained LSTM with denormalization.

We have set RMSE as the loss function for LSTM training and used the Adam Optimizer to minimize it. The generated LSTM model has a total of 198,273 trainable parameters, and each model undergoes training for 50 epochs.

Figure 7b illustrates the process of using the trained model to make predictions. The X values from the Test Data are sequentially input into the trained LSTM model, and the model predicts the Y values corresponding to the X values through the prediction process, representing the CPU rate. As these predictions are normalized, they undergo a Denormalization process to convert them back into actual prediction values.

Therefore, in our research, we apply the LSTM model to the scheduling of time series data. This model effectively detects non-linear behavior patterns, understands temporal dependencies, and adapts to chaotic long-term time series data through adaptability and model generalization.

6. Experiment

In this study, we conducted two experiments. The first is an experiment on the automatic recovery of a cluster, and the second is a scheduling experiment that uses LSTM. Both experiments were conducted in a cloud environment. The primary objective of this study was to demonstrate the efficiency and performance of Kubernetes cluster automatic recovery according to the recovery time and prove the necessity of scheduling using machine learning.

The experimental environment uses AWS EC2 instances and S3 buckets, with the EC2 instances running Ubuntu Server 22.04 LTS (HVM), SSD Volume Type. The open-source tools, Rancher and Velero, are used in the Kubernetes management platform and Kubernetes backup and restoration tools, respectively. The open-source tools, Rancher and Velero, cannot afford to perform automatic recovery themselves. Our novel module, well integrated with those tools, implements (1) our proposed automatic recovery mechanism and (2) efficient restoration to most stable target clusters using LSTM-based available resource prediction.

6.1. Automated Cluster Recovery

Figure 8 shows the Rancher configuration for the automatic cluster-recovery experiment. In the figure, green represents the systems we added, orange represents Velero, and blue represents components of Rancher. The Kubernetes cluster running Rancher is referred to as the Rancher cluster. Although Rancher can operate as a container on a single server, in our experiment, it runs within a Kubernetes cluster because of the use of Velero. Clusters 1 and 2, linked to the Rancher cluster, each comprise one master node and two worker nodes. The master node is a t2.medium EC2 instance type with 2 vCPU, 4 GB memory, and 30 GB storage, and the worker nodes use t2.small EC2 instances with 1 vCPU, 2 GB memory, and 30 GB storage.

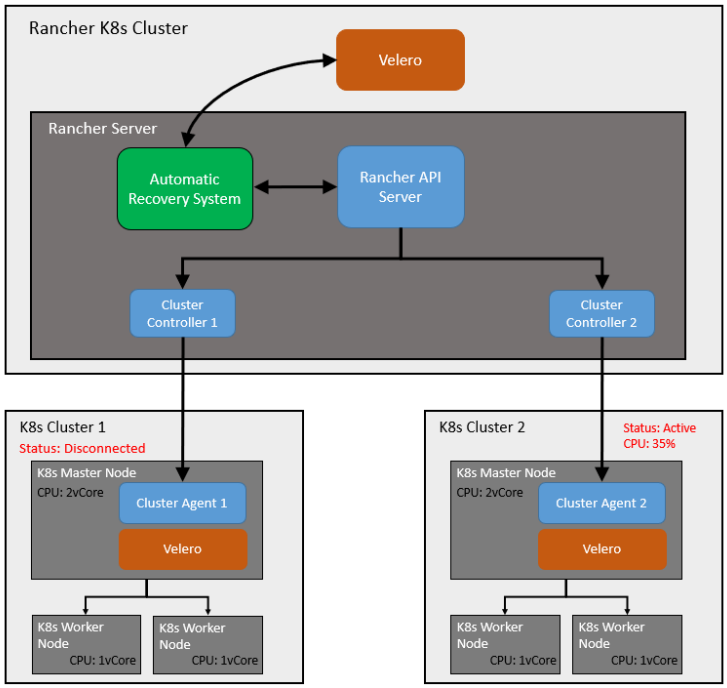

Figure 8. Architecture for the automated cluster-recovery experiment.

Figure 9 shows the flowchart of the cluster auto-recovery experiment. In the figure, green represents the systems we added, orange represents Velero, and blue represents components of Rancher. Solid arrows represent the flow of backup file data, while dashed arrows indicate communication, commands, and similar actions. Velero, connected to the AWS S3 bucket, is installed in all clusters, including Rancher Cluster, Cluster 1, and Cluster 2. In step 2, if Cluster 1 loses functionality due to a disaster, Rancher detects this. In step 3, the detected status of Cluster 1 changes to disconnected, triggering the auto-recovery system. In step 4, the auto-recovery system uses Velero to look up the latest backup file of Cluster 1. In step 5, the restore command, including the name of this backup file, is transmitted to Cluster 2. In step 6, Cluster 2 executes the received command, using the backup file in the S3 bucket to restore the application that was running in Cluster 1 to Cluster 2.

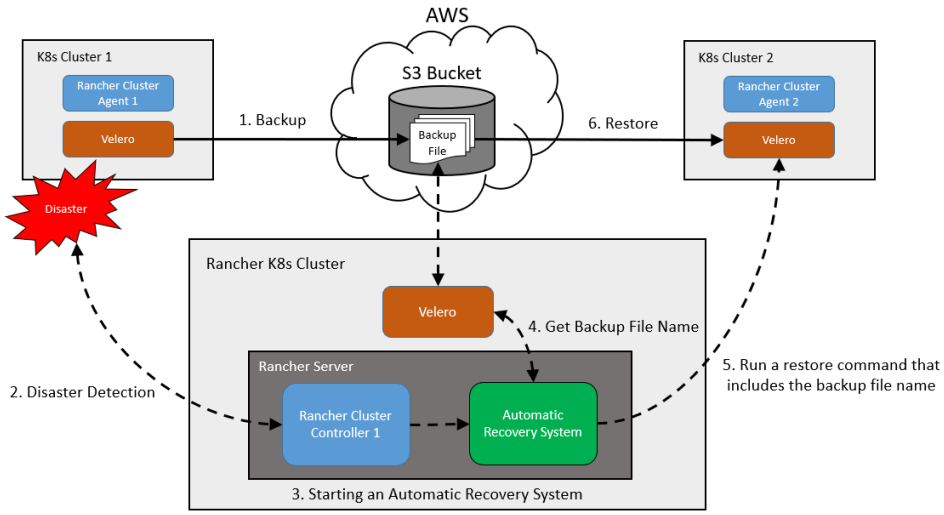

Figure 9. Flowchart of the automated cluster-recovery experiment.

The automatic recovery experiment was conducted as follows. First, Nginx was run on Cluster 1, and a backup file was created in the AWS S3 bucket by using the Velero backup command. Then, the master node of Cluster 1 was stopped to disconnect it from the Rancher cluster. When the disconnection is detected by the Rancher cluster, the proposed automatic recovery system is initiated for the cluster. Because Clusters 1 and 2 both have a total of 4vCPU allocated, Cluster 2 is a viable cluster for restoration. In this experiment, because Cluster 2 was the only target cluster available for restoration, the restoration proceeded to Cluster 2 without scheduling. First, Velero in the Rancher cluster queried the latest Nginx backup file in the S3 bucket and sent the restoration command, including the name of the backup file, to Cluster 2. Cluster 2 executed the received command, restoring the Nginx running on Cluster 1 by using the backup file in the S3 bucket.

In this experiment, excluding the process of creating the backup file, the time taken from when the master node of Cluster 1 was stopped, from the creation of an artificial disaster scenario to when the Nginx on Cluster 2 was completely restored was measured. This experiment was repeated 10 times to measure the time. The time taken from executing the Velero restoration command in Cluster 2 to the completion of the restoration is referred to as the restoration time. The time utilized by the process, excluding the restoration time, is attributed to the proposed system. The measurement and analysis of these times are explained in Section 7.

6.2. LSTM-Based Scheduling

Our experiment is based on LSTM-based scheduling, utilizing a model trained on Google Cluster Trace data [20]. In the work presented in [20], it is stated that "Most resource utilization measurements and requests have been normalized, including: (1) CPU (core count or core seconds/second); (2) memory (bytes); (3) disk space (bytes); (4) disk time fraction (I/O seconds/second). For each of the foregoing, we compute separate normalizations. The normalization is a scaling relative to the largest capacity of the resource on any machine in the trace (which is 1.0)". In this study, we also assume that all CPU resources are homogeneous and normalized regarding maximum consumption as in Google Cluster Trace data [20].

The data includes cluster CPU usage rates at 5 min intervals, enabling the training of a model to predict the cluster CPU usage rate 5 min into the future. The model, built on insights from five experiments, analyzes CPU usage rates over the past 15 min to predict the rate for the next 5 min. The choice of predicting the CPU usage rate for the next 5 min is due to the high volatility in the Google Cluster Trace data. Through LSTM-based scheduling, the algorithm selects clusters with higher stability compared to algorithms considering only the current CPU usage.

To validate this, we conducted two types of experiments. In the first experiment, we utilized test data comprising 20% of the total data from Figure 6. For each data point, we compared the current CPU utilization value with the value 5 min ahead. This process started from the 6681st data point and iterated through the last data point, calculating the RMSE (Root Mean Square Error) metric. In the second experiment, we constructed an xLSTM model, following the predictive model architecture presented in Section 5.2. After hyperparameter tuning, we obtained the best parameters as follows: lookback window size = 3; prediction window size = 1; batch size = 64; learning rate = 0.0001; trainable LSTM parameters = 789,761; and training epoch = 100. The model was trained on data from the past 15 min corresponding to the lookback window value of 3 to predict CPU utilization 5 min ahead (prediction window value of 1). We compared the predicted values with the actual values in the test data and calculated the RMSE metric.

As a result of the experiments, the RMSE value in the second experiment utilizing LSTM improved by 2.21% compared to the first experiment. This outcome validates the excellence of CPU utilization prediction using LSTM. Based on these results, we applied the experiment to real-world environments.

In the experiments, Google Cluster Trace data is applied to predict CPU usage rates in the current environment. The cluster's CPU usage rates are adjusted at 5 min intervals, aligned with the patterns in the Google Cluster Trace data. This process aims to validate the model's prediction accuracy by adapting it to the real environment. In the event of a disaster, the model mimics data patterns, predicts CPU usage based on applied data in the actual cluster environment, and selects the cluster with the lowest predicted CPU usage rate for recovery.

Figure 10 shows the Rancher configuration for the LSTM-based scheduling experiment. In the figure, green represents the systems we added, orange represents Velero, and blue represents components of Rancher. Each of the six clusters linked to the Rancher cluster comprises one master node and two worker nodes. The master node is a t2.medium EC2 instance with 2 vCPU, 4 GiB memory, and 30 GiB storage, whereas the worker nodes are t2.small EC2 instances with 1v CPU, 2 GiB memory, and 30 GiB storage. The CPU-utilization rates of the clusters were varied, which was intentional for the scheduling test.

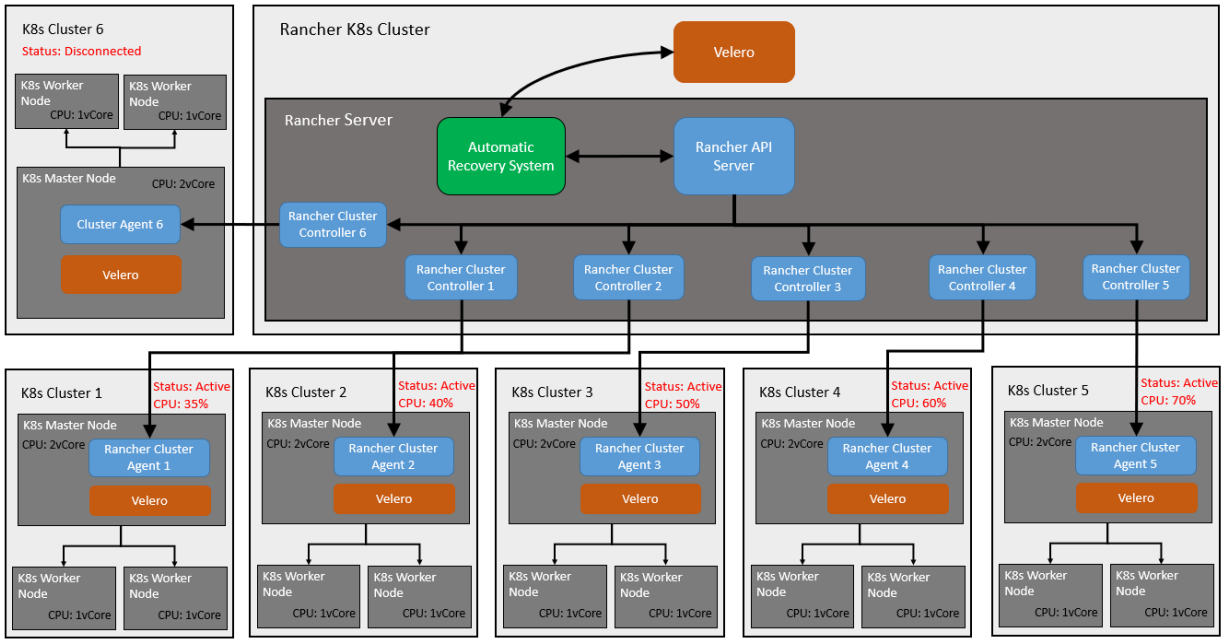

Figure 10. Architecture used in the LSTM-based scheduling experiment.

Figure 11 illustrates a YAML file for a pod operating to adjust the CPU-utilization rates of the clusters. This pod uses a very resource-light busybox image. The "requests" and

"limits" represent the CPU usage request and maximum limit, respectively. By adjusting these values and running them, the CPU-utilization rates of each cluster can be manipulated. In the example, the "requests" and "limits" were set to 200 m to achieve 5% utilization of the total 4vCPU of the cluster.

```
apiVersion: v1
kind: Pod
metadata:
  name: dummy-pod
  namespace: dummy
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "while true; do sleep 1; done"]
    resources:
      requests:
        cpu: "200m"
      limits:
        cpu: "200m"
```

Figure 11. Dummy application YAML.

Figure 12 shows the flowchart of the LSTM-based scheduling experiment. In the figure, green represents the systems we added, orange represents Velero, and blue represents components of Rancher. Solid arrows represent the flow of backup file data, while dashed arrows indicate communication, commands, and similar actions. Velero, connected to the AWS S3 bucket, is installed in all clusters. In step 1, Cluster 6 uses Velero to create a backup file of the application running in it in the S3 bucket. In step 2, if Cluster 6 loses functionality due to a disaster, Rancher detects this. In step 3, the detected status of Cluster 6 changes to disconnected, triggering the auto-recovery system. In step 4, the auto-recovery system uses Velero to look up the latest backup file of Cluster 6. In step 5, the restore command, including the name of this backup file, is transmitted to the target cluster selected through LSTM-based scheduling. In the figure, it is transmitted to Cluster 1, which has the lowest predicted CPU usage. In step 6, the target cluster executes the received command, using the backup file in the S3 bucket to restore the application that was running in Cluster 6 to the target cluster.

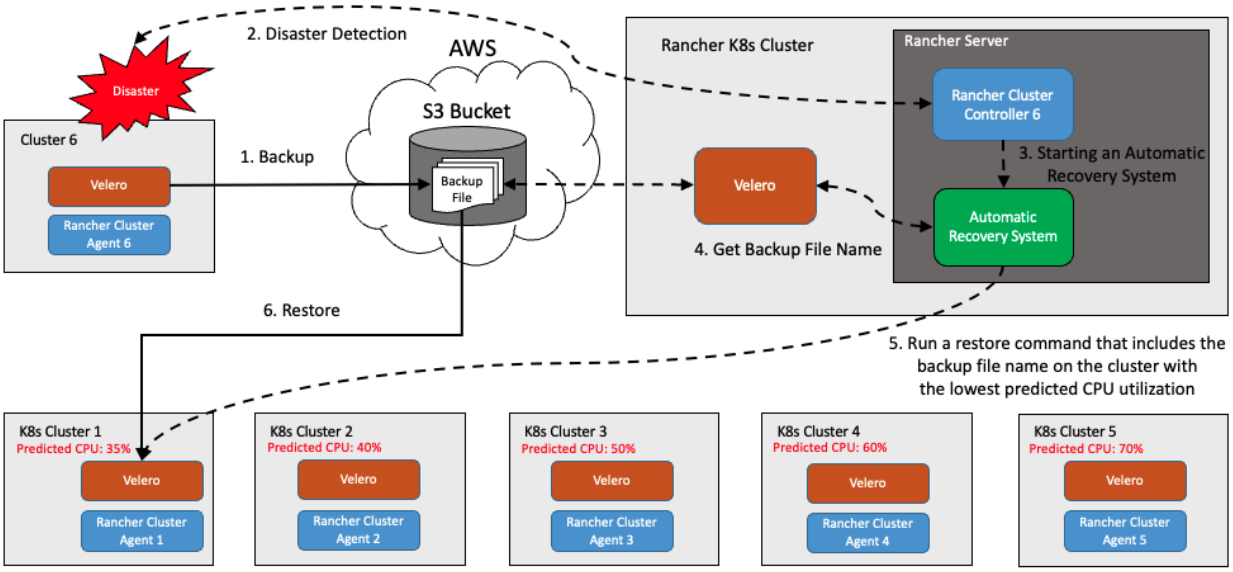

Figure 12. Flowchart of the LSTM-based scheduling experiment.

The procedure of the LSTM-based scheduling experiment is as follows. First, a dummy application is run on Cluster 6. This running application is backed up to the AWS S3 bucket by using the Velero backup command. Then, the master node of Cluster 6 is stopped to disconnect it from the Rancher cluster. After the Rancher cluster detects the disconnection, our automatic recovery system is initiated for the cluster. As Cluster 6 contains a total of 4vCPU, and Clusters 1, 2, 3, 4, and 5 each have 4vCPU allocated, they are all potential restoration targets. In this experimental environment, multiple target clusters are available for restoration, and thus scheduling is conducted. The current CPU-utilization rates of the clusters are used as input values for the LSTM-based CPU prediction. The cluster with the lowest predicted CPU-utilization rate is selected as the target cluster. Then, Velero in the Rancher cluster queries the latest backup file of the dummy application in the S3 bucket and sends the restoration command, including the name of the backup file, to the target cluster. The target cluster executes the received command, restoring the dummy application running on Cluster 6 by using the backup file in the S3 bucket. In our study, this experiment was conducted 10 times, measuring the CPU-utilization rates of the clusters each time. Additionally, an experiment in which the target cluster was randomly selected instead of using LSTM-based scheduling was conducted 10 times in the same manner. These two experiments are compared and analyzed in Section 7.

7. Results and Discussion

7.1. Automated Cluster Recovery

In Table 1, A represents the time taken to restore Nginx in our environment, B is the time taken for automatic recovery, and A − B is the time excluding the restoration time from the total time spent on automatic recovery. The restoration of Nginx in our environment takes 20 s, which is the time required for the restoration operation to be executed. In the automatic recovery experiment, the restoration was completed in an average of 27 s across 10 trials, within a range of 20–34 s. Excluding the time taken for the restoration operation, an additional 0–14 s was added. This time is attributed to the delay caused by Rancher's 15-s interval for detecting cluster disconnections. However, even if a user were to perform manual recovery, the inevitable delay would occur only after a Rancher detects cluster disconnection. Excluding this inevitable delay, the actual additional delay caused by the automatic recovery operation is determined to be less than 1 s. Therefore, the experiment proves the efficiency of automatic recovery by eliminating the delay that is caused by user intervention in a manual recovery process.

Table 1. Results of the time taken to recover Nginx.

Case	Recovery Time (A)	Restoration Time (B)	(A − B)
1	23	20	3
2	34	20	14
3	28	20	8
4	26	20	6
5	32	20	12
6	22	20	2
7	23	20	3
8	25	20	5
9	30	20	10
10	27	20	7
AVG	27	20	7

7.2. LSTM-Based Scheduling

In this section, the importance of CPU-utilization prediction is explored in the automated-recovery process. First, an experiment selecting the target cluster for restoration based on CPU-utilization prediction by using LSTM is compared with an experiment randomly selecting the target cluster. In each experiment, restoration operations were conducted 10 times on five clusters, starting from the same initial state, and the variations in CPU utilization in each cluster were analyzed. According to Gusev et al. [29], when CPU utilization exceeds 80%, the occurrence of system performance degradation is highly possible. Thus, maintaining a stable level of CPU utilization in the cluster is important. This study thoroughly analyzed the results of both experiments, presenting a comparative analysis.

Table 2 shows the results of randomly selecting a target cluster among five clusters and performing restoration 10 times. The red text in the table shows which cluster the restoration at that point is scheduled for. The first restoration occurred in Cluster 5, demonstrating the highest initial CPU utilization, followed by several other restorations. After 10 restoration operations, Clusters 4 and 5 reached 85% CPU utilization, indicating a high probability of performance degradation. Moreover, CPU utilization across clusters demonstrates significant variation.

Table 2. Results of randomly scheduling clusters.

Restoration Count	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Initial State	35%	40%	50%	60%	70%
1	35%	40%	50%	60%	75%
2	40%	40%	50%	60%	75%
3	40%	40%	50%	65%	75%
4	40%	40%	50%	70%	75%
5	40%	40%	50%	70%	80%
6	40%	40%	50%	70%	85%
7	40%	40%	55%	70%	85%
8	40%	40%	55%	75%	85%
9	40%	40%	55%	80%	85%
10	40%	40%	55%	85%	85%
Clusters with CPU utilization under 80%	O	O	O	X	X

Table 3 presents the results of selecting the target cluster for restoration based on CPU-utilization prediction with LSTM and conducting 10 restorations. The red text in the table shows which cluster the restoration at that point is scheduled for. The first restoration began in Cluster 1, which has the lowest initial CPU-utilization rate, and subsequent restoration operations targeted clusters with the lowest rate of CPU utilization at each point. After completing 10 restoration operations, none of the clusters exceeded 80% CPU utilization. Moreover, all clusters maintained a stable state, with CPU utilization between 55% and 70%.

The results of these two experiments demonstrate the importance of scheduling based on CPU-utilization prediction. As shown in Table 2, in Experiment 1, restoration was started in Cluster 5, which had high initial CPU utilization, leading to performance degradation. Eventually, CPU utilization in Clusters 4 and 5 reached 85%, increasing the risk of performance degradation and showing an imbalance in utilization among clusters. In contrast, in Experiment 2, (Table 3) restoration was started in clusters with lower CPU utilization and LSTM for prediction, which helped to maintain system balance. All clusters

maintained a stable state, not exceeding 80% CPU utilization, and a utilization rate between 55% and 70% indicated efficient resource usage.

Table 3. Results of scheduling clusters based on CPU-usage prediction by using LSTM.

Restoration Count	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Initial State	35%	40%	50%	60%	70%
1	40%	40%	50%	60%	70%
2	45%	40%	50%	60%	70%
3	45%	45%	50%	60%	70%
4	50%	45%	50%	60%	70%
5	50%	50%	50%	60%	70%
6	55%	50%	50%	60%	70%
7	55%	55%	50%	60%	70%
8	55%	55%	55%	60%	70%
9	60%	55%	55%	60%	70%
10	60%	60%	55%	60%	70%
Clusters with CPU utilization under 80%	O	O	O	O	O

Additionally, we compared five prediction methods: Random Selection, Lowest (5 m ahead), Lowest (10 m ahead), LSTM (5 m ahead), and LSTM (10 m ahead). Table 4 presents the RMSE for these five methods. Random Selection shows the results of randomly selecting a cluster from the five clusters. Lowest (5 m ahead) involves selecting the cluster with the lowest CPU usage among the five and predicting the CPU usage for the next 5 min based on the current value. Similarly, Lowest (10 m ahead) predicts the CPU usage for the next 10 min using the same approach. LSTM (5 m ahead) utilizes 15 min of data from the five clusters to predict the CPU usage for the next 5 min, while LSTM (10 m ahead) uses 30 min of data to predict the CPU usage for the next 10 min. As shown in Table 4, LSTM demonstrates approximately 2.2% lower RMSE than Lowest.

Table 4. Random Selection vs. Lowest Selection vs. LSTM-based Selection.

Metric	Random	Lowest (5 m Ahead)	Lowest (10 m Ahead)	Lowest (20 m Ahead)	LSTM (5 m Ahead)	LSTM (10 m Ahead)	LSTM (20 m Ahead)
RMSE	3085.83	312.30	472.20	608.09	305.54	461.75	621.31

This comparison proves that prediction-based scheduling by using LSTM is crucial for resource management and performance degradation prevention in cloud environments. When the prediction model was used to determine the order of restoration operations, it enhanced the overall system performance, increased resource usage efficiency, and minimized the risk of potential performance degradation.

LSTM is widely used for its strong ability to model long-term dependencies. However, due to its intrinsic recurrent approach, there is a tendency for computational and memory usage to increase as the sequence length becomes longer. This limitation can also be observed in Table 4. For example, the RMSE for a prediction made 5 min later using LSTM improved by about 50.8% compared to the RMSE for a prediction made 20 min later. Therefore, this constraint can lead to problems in large datasets or real-time applications.

Moreover, the scalability issue of LSTM models is also worth noting in relation to parallelization. Due to its sequential nature, parallelizing LSTMs is generally challenging,

which can slow down training speeds in large datasets. These limitations can restrict the practical applicability of LSTMs in real-world scenarios.

Therefore, future research needs to consider the Transformer architecture. Transformers can effectively address long-term dependency issues using attention mechanisms and are efficient in parallel processing [30]. Transformer architectures for time-series data prediction have been recently studied and leveraging them could overcome the limitations of LSTMs [31,32].

7.3. Scalability

In our current environment, the time taken to schedule five clusters is less than 1 s. Judging from these results, scheduling more clusters would not result in significant delays. Even with more target clusters, LSTM predictions can be parallelized easily. The more critical issue would be the training of multiple customized LSTM models for a large number of target clusters. Still, the issue will also be resolved with dedicated training servers conducting online learning through periodic new data collections from target clusters.

Suppose we apply our proposed methodology to a multi-resource (e.g., CPU and GPU) cloud environment. In that case, we can collect more detailed information about services running on a failed cluster and train more customized LSTM prediction models. For example, a service requiring 0.1 GPU resources and 0.5 CPU resources, where the values are all normalized, can be relocated to a target cluster having enough GPU and GPU resources by inferring GPU and CPU resource consumption prediction simultaneously. Either two separate LSTM models for GPU and CPU, respectively, or a single LSTM model predicting two types of resources at the same time can be developed. Likewise, any different types of resources can be handled.

8. Conclusions

This study focused on the efficiency of a Kubernetes cluster automatic recovery system and the effectiveness of scheduling methods using LSTMs. Compared to previous studies that mainly explored automatic recovery and restoration at the service or application level, this study experimentally verified the feasibility and effectiveness of automatic recovery by targeting the entire cluster. The contributions of this study are two-fold: the reduction in recovery time and the efficient utilization of resources. In addition, the scheduling method based on CPU-utilization prediction by using LSTM can optimize the selection of the target cluster for restoration, contributing to the overall system performance and stability.

Consequently, this research significantly contributes to the design and operation of automatic recovery systems in cloud environments, opening new horizons for future research. In addition, we emphasize the need for further research on the applicability of automatic recovery in various cloud environments and system optimization by integrating AI technologies. This could stimulate research and innovation to advance cloud computing.

Author Contributions: Conceptualization, J.-B.K. and E.-S.J.; methodology, J.-B.K. and E.-S.J.; software, J.-B.K.; validation, J.-B.K. and E.-S.J.; formal analysis, J.-B.K.; investigation J.-B.K. and J.-B.C.; resources, J.-B.K. and E.-S.J.; data curation, J.-B.K. and J.-B.C.; writing—original draft preparation, J.-B.K. and J.-B.C.; writing—review and editing, J.-B.K.; visualization, J.-B.K.; supervision, E.-S.J.; project administration, E.-S.J.; funding acquisition, E.-S.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work was also supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2021-0-02082, CDM_Cloud: Multi-Cloud Data Protection and Management Platform).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

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

References

1. Menard, P.; Gatlin, R.; Warkentin, M. Threat Protection and Convenience: Antecedents of Cloud-Based Data Backup. *J. Comput. Inf. Syst.* **2014**, *55*, 83–91. [CrossRef]
2. Landry, B.J.L.; Koger, M.S. Dispelling 10 common disaster recovery myths: Lessons learned from Hurricane Katrina and other disasters. *J. Educ. Resour. Comput.* **2006**, *6*, 6. [CrossRef]
3. Schroeder, B.; Damouras, S.; Gill, P. Understanding latent sector errors and how to protect against them. ACM Trans. *Storage* **2010**, *6*, 1–23. [CrossRef]
4. Nath, S.; Yu, H.; Gibbons, P.B.; Seshan, S. Subtleties in Tolerating Correlated Failures in Wide-area Storage Systems. In Proceedings of the Third Conference Networked Systems Design & Implementation—(NSDI '06), San Jose, CA, USA, 8–10 May 2006; Volume 3.
5. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing*; NIST Special Publication 800-145; NIST: Gaithersburg, MD, USA, 2011. [CrossRef]
6. Tomas, L.; Kokkinos, P.; Anagnostopoulos, V.; Feder, O.; Kyriazis, D.; Meth, K.; Varvarigos, E.; Varvarigou, T. Disaster Recovery Layer for Distributed OpenStack Deployments. *IEEE Trans. Cloud Comput.* **2020**, *8*, 112–123. [CrossRef]
7. Harwalkar, S.; Sitaram, D.; Jadon, S. Multi-cloud DRaaS using OpenStack Keystone Federation. In Proceedings of the 2019 International Conference on Advances in Computing and Communication Engineering (ICACCE), Sathyamangalam, India, 4–6 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6. [CrossRef]
8. Sato, T.; He, F.; Oki, E.; Kurimoto, T.; Urushidani, S. Implementation and Testing of Failure Recovery Based on Backup Resource Sharing Model for Distributed Cloud Computing System. In Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 22–24 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–3. [CrossRef]
9. Zhang, J.-h.; Zhang, N. Cloud Computing-based Data Storage and Disaster Recovery. In Proceedings of the 2011 International Conference on Future Computer Science and Education, Xi'an, China, 20–21 August 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 629–632. [CrossRef]
10. Wood, T.; Cecchet, E.; Ramakrishnan, K.K.; Shenoy, P. Disaster Recovery as a Cloud Service: Economic Benefits & Deployment Challenges. In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, 22–25 June 2010; pp. 22–25.
11. Poniszewska-Marańda, A.; Czechowska, E. Kubernetes Cluster for Automating Software Production Environment. *Sensors* **2021**, *21*, 1910. [CrossRef] [PubMed]
12. Yu, H.; Xiang, X.; Zhao, Y.; Zheng, W. BIRDS: A Bare-Metal Recovery Systemfor Instant Restoration of Data Services. *IEEE Trans. Comput.* **2014**, *63*, 1392–1407. [CrossRef]
13. Sousa, T.B.; Ferreira, H.S.; Correia, F.F.; Aguiar, A. Engineering Software for the Cloud: Automated Recovery and Scheduler. In Proceedings of the 23rd European Conference on Pattern Languages of Programs, Irsee, Germany, 4–8 July 2018; ACM: New York, NY, USA, 2018; pp. 1–8. [CrossRef]
14. Yu, X.; Wang, D.; Sun, X.; Zheng, B.; Du, Y. Design and Implementation of a Software Disaster Recovery Service for Cloud Computing-Based Aerospace Ground Systems. In Proceedings of the 2022 11th International Conference on Communications, Circuits and Systems (ICCCAS), Singapore, 13–15 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 220–225. [CrossRef]
15. Challagidad, P.S.; Dalawai, A.S.; Birje, M.N. Efficient and Reliable Data Recovery Technique in Cloud Computing. *Internet Things Cloud Comput.* **2017**, *5*, 13–18. [CrossRef]
16. Jun, Y.; Lihong, Y. The Cloud Technology Double Live Data Center Information System Research and Design Based on Disaster Recovery Platform. *Procedia Eng.* **2017**, *174*, 1356–1370. [CrossRef]
17. Wang, L.; Harper, R.E.; Mahindru, R.; Ramasamy, H.V. Disaster Recovery for Cloud-Hosted Enterprise Applications. In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 27 June–2 July 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 432–439. [CrossRef]
18. Google Cloud Documentation. Available online: https://cloud.google.com/docs (accessed on 17 March 2024).
19. Disaster Recovery Service—AWS Elastic Disaster Recovery—AWS. Available online: https://aws.amazon.com/disaster-recovery/ (accessed on 17 March 2024).
20. Reiss, C.; Wilkes, J.; Hellerstein, J.L. Google Cluster-Usage Traces: Format + Schema. Google, 2 September 2023. Available online: https://github.com/google/cluster-data (accessed on 3 September 2023).
21. Bi, J.; Li, S.; Yuan, H.; Zhou, M. Integrated deep learning method for workload and resource prediction in cloud systems. *Neurocomputing* **2021**, *424*, 35–48. [CrossRef]
22. Kumar, J.; Singh, A.K.; Buyya, R. Self directed learning based workload forecasting model for cloud resource management. *Inf. Sci.* **2021**, *543*, 345–366. [CrossRef]
23. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef] [PubMed]
24. Kolen, J.F.; Kremer, S.C. Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies. In *A Field Guide to Dynamical Recurrent Networks*; IEEE: Piscataway, NJ, USA, 2001; pp. 237–243.
25. Song, B.; Yu, Y.; Zhou, Y.; Wang, Z.; Du, S. Host load prediction with long short-term memory in cloud computing. *J. Supercomput.* **2018**, *74*, 6554–6568. [CrossRef]
26. Lim, B.; Zohren, S. Time-series forecasting with deep learning: A survey. *Phil. Trans. R. Soc. A.* **2021**, *379*, 20200209. [CrossRef] [PubMed]

27. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
28. Luo, Z.; Qi, R.; Li, Q.; Zheng, J.; Shao, S. ABODE-Net: An Attention-based Deep Learning Model for Non-intrusive Building Occupancy Detection Using Smart Meter Data. In *Smart Computing and Communication*; SmartCom 2022 Lecture Notes in Computer Science; Qiu, M., Lu, Z., Zhang, C., Eds.; Springer: Cham, Switzerland, 2023; Volume 13828. [CrossRef]
29. Gusev, M.; Ristov, S.; Simjanoska, M.; Velkoski, G. CPU Utilization while Scaling Resources in the Cloud. In Proceedings of the 4th International Conference on Cloud Computing, GRIDS, and Virtualization, Valencia, Spain, 27 May–1 June 2013.
30. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. *arXiv* **2017**, arXiv:1706.03762.
31. Chang, C.; Wang, W.-Y.; Peng, W.-C.; Chen, T.-F. LLM4TS: Aligning Pre-Trained LLMs as Data-Efficient Time-Series Forecasters. *arXiv* **2024**, arXiv:2308.08469v5.
32. Li, Q.; Luo, Z.; Qi, R.; Zheng, J. DeepTPA-Net: A Deep Triple Attention Network for sEMG-Based Hand Gesture Recognition. *IEEE Access* **2023**, *11*, 96797–96807. [CrossRef]