symmetry

MDPI

Article

An Efficient Applications Cloud Interoperability Framework Using I-Anfis

Chithambaramani Ramalingam [1,*] and Prakash Mohan [2]

[1] Department of CSE, T.J.S Engineering College, Thiruvallur 601206, India
[2] Department of CSE, Karpagam College of Engineering, Coimbatore 641032, India; salemprakash@gmail.com
* Correspondence: chithambaramani.r@gmail.com

Abstract: Cloud interoperability provides cloud services such as Software as a Service (SaaS) or customer system to communicate between the cloud providers. However, one of the most important barriers for existing researches was to adopt the application's or data's in cloud computing environments so as to obtain efficient cloud interoperability. This paper focuses on reliable cloud interoperability with a heterogeneous cloud computing resource environment with the objective of providing unilateral provision computing capabilities of a cloud server without the help of human interaction and allowing proper utilization of applications and services across various domains by using an effective cloud environment available at runtime. Moreover, the framework uses hybrid squirrel search genetic algorithm (HSSGA) to select the relevant features from a set of extracted features in order to eliminate irrelevant data which provides advantages of low computational time and less memory usage. Thereafter, for a proper selection of cloud server with respect to the selected features, the system has developed the improved adaptive neuro-fuzzy inference system (I-ANFIS) which provides accurate server selection and helps against uncertainties caused by servers or applications. Hence, the experimental result of the proposed framework gives an accuracy of 94.24% and remains more efficient compared to existing frameworks.

Keywords: hybrid squirrel search genetic (HSSG) algorithm; improved adaptive neuro-fuzzy inference system (I-ANFIS)

check for updates

Citation: Ramalingam, C.; Mohan, P. An Efficient Applications Cloud Interoperability Framework Using I-Anfis. *Symmetry* **2021**, *13*, 268. https://doi.org/10.3390/sym13020268

Academic Editor: Theodore E. Simos
Received: 31 December 2020
Accepted: 31 January 2021
Published: 5 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the increasing need of computing resources such as server, storage, CPU, application, database and network, which enables a new computing paradigm named cloud computing. Cloud infrastructure which are available in a data centers have to be shared among organizations through pay as you go models use service anytime anywhere [1]. Cloud service can be provisioned rapidly and released with minimum human intervention in a cost effective manner. Cloud service can be scaled based on user needs [2]. Intensive need of cloud computing services such as network-demanded web application, computation resource intensive application [3] and storage dependent application [4] are drivers of this technology.

Vendors such as Amazon [5,6], IBM Cloud, Openstack, Google [7], Pivotal Web, Microsoft Azure [8] and Salesforce [9] have seen rapid growth in their cloud business for the past 10 years. Each of the cloud providers have their own format to access the cloud, which results in the consumer opting for one cloud provider. There are no widely accepted standards for accessing and deploying cloud services for cloud consumers. Multi-clouds need to work seamlessly with other cloud services in an interoperable manner [10]. Using the Service Level Agreement (SLA) format, authorization and authentication token [11], it is possible for the data and workload available in a public or private cloud to move between one cloud provider to another cloud provider which enable interoperability between clouds. Interoperability in cloud computing will have three kind of interoperable cloud

services such as platform, application and management interoperability with response to management, application and platform components [12].

In recent years, industry and academia have highlighted the need for cloud interoperable services. Many cloud standards address interoperability due to increasing demand. Cloud standards address the issue of interoperability when there is an exchange of data, applications and Virtual Machine (VMs) between different cloud providers [13]. These inconsistencies can be either specialized, for example incongruent virtualization executions (VMware, Xen and KVM), or contrary programming code (Java-based, PHP-based) or semantic. Notwithstanding that, data synchronization is a specific issue when resource components in various clouds interact, regardless of whether they are indistinguishable [14]. Such resource components regularly keep duplicates of similar information, and these duplicates should be maintained in a steady state. Interaction between clouds ordinarily have high latency, which makes synchronization troublesome [15].

Many cloud interoperability standards have been proposed based on state-of-the-art efforts, such as Distributed Management Task Force (DMTF) [16], Storage Networking Industry Association (SNIA) [17], Open Virtual Machine Format (OVF) standards [18], NIST Cloud Computing related standards [19], et cetera, to enable cloud interoperability and to be implemented soon. However, a technological challenge still remains in terms of different hypervisors for virtualization in cloud environments and platforms for developing applications and non-technological challenges in terms of the policy as well as management in cloud interoperability [20]. In order to overcome the challenges faced by existing research, this work has developed an efficient applications interoperability framework using the I-ANFIS algorithm.

Therefore, the rest of the paper is organized as follows: Section 2 reviews various frameworks developed for cloud interoperability; Section 3 briefly explains the proposed methodology for cloud interoperability; Section 4 discusses the results achieved by the proposed work based on various performance metrics, and finally; Section 5 concludes the work with future scope.

2. Literature Review

This section illustrates the various existing works or frameworks carried out for cloud interoperability and the corresponding challenges faced by the framework.

Huedo et al. [21] developed a BEACON framework which allows cloud applications and services to be deployed automatically across different data centers and different clouds with emphasis on intercloud networking and security issues and enables the provision of federated cloud infrastructures. BEACON has contributed to the OpenNebula and OpenStack cloud management platforms so as to manage the data interoperability and it is a open-source software. The management of data interoperability was modeled well. However, they had repeated data or imbalanced distribution of data, which were the big challenges in the framework.

Castane et al. [22] illustrated mOSAIC ontology for inter-cloud interoperability which was extended to create the cloud lightning ontology (CL-ontology). CL-ontology was the incorporation of heterogeneous resources and high performance computing (HPC) environments in the cloud. To help the CL-ontology, a conventional architecture was introduced as a driver to address heterogeneity in the cloud, and as a utilization case illustration of the architecture, the internal processes of the cloud lightning framework were updated and introduced to show the plausibility of joining a semantic engine to reduce interoperability issues to encourage the consolidation of HPC in the cloud. Based on the inter-cloud interoperability, the approach was developed. However, when it considered various platforms, the respective execution of an application did not match with the corresponding server environment.

Nodehi et al. [23] discussed a framework that supports interaction between different cloud resources to support inter cloud interoperability with the objective of dispatching the remaining workloads to the best cloud service accessible at runtime. Additionally, a genetic

algorithm (GA)–based employment scheduler was created as a part of the interoperability system that offered remaining job movement with the best execution at any event cost. The cloud resource selection model [24] was evaluated with an agent-based simulation. The approach used features selection techniques to provide accurate placement of the data or applications with the respective cloud server environment; but the selection of features remained more complex and time-consuming.

K. Chandrasekaran et al. [25] used service level agreement to implement interoperable cloud services. To provide sufficient cloud resources during the deficiency of resources of one cloud, it encouraged concurred administrations by the client. Adaptable resource management methodology was created for interoperability-based cloud services. At first, the service level agreement (SLA) layouts of private and public clouds were mapped with the soft term frequency inverse document frequency (TF-IDF) metric with case-based reasoning (CBR) approach. At that point, in light of the mapped SLA, various groups of cloud suppliers were framed with the assistance of the K-means clustering technique. Lastly, when one of the clouds in a cluster confronted the issue of resource deficiency, adaptable cloud resource assignment was given through the adaptive dimensional search algorithm. The developed interoperable cloud computing with SLA remains to provide unwanted resources needed for an application to get executed.

Huedo et al. [26] used an SLA-based service virtualization which a self-manageable architecture gives an approach for easing interoperable service executions in an assorted, heterogeneous, distributed and virtualized universe of cloud services. In a distributed environment, to achieve reliable and efficient service operation, a model with the combination of negotiation, brokering and deployment with SLA-aware extensions and autonomic computing principles was used. The method remained more challenging at the time of big file size or application which required installation of wide setup files.

Loulloudes et al. [27] used a utility-based approach for provisioning on demand cloud computing resource to IT solutions. Currently, cloud computing consumers, who are willing to deploy cloud applications across different cloud resource providers do not have any automated user application programming interface (API) nor support tools. The approach presented current efforts to develop an open-source cloud application management framework (CAMF) based on the Eclipse Rich Client Platform. The framework facilitated cloud application lifecycle management in a vendor-neutral way to avoid interoperability issues. The framework conflicted with a vendor in a neutral way to avoid interoperability but the technological issues reduced its efficiency.

Sehgal et al. [28] has addressed interoperability at application-level which paves the path to understanding allocation of cloud resources at distributed infrastructure level. The approach was widely used, and it had the ability to capture the primary challenges in developing distributed applications. It used MapReduce as the fundamental exemplar and it developed an interoperable implementation of MapReduce with an application programming interface (API) called Simple API for Grid Applications (SAGA) to support distributed programming. Thereafter, the work provided a canonical word count application that used SAGA-based MapReduce. At last, the performance measures were provided and analyses of the SAGA-map were done based on multiple, different, heterogeneous infrastructures concurrently for the same problem instance. As for the same problem instance, the framework was analyzed and stated to be efficient for mapping the application according to the respective servers but non-technological issues caused an obstacle in terms of providing efficient interoperability.

Thus, the above investigation illustrates various existing works to conquer cloud interoperability and to provide smooth application-based computing. However, technological challenges as well as non-technological challenges remain that cause inefficient data or application interoperability in the cloud. In order to overcome the challenge, the work has initiated efficient application interoperability in cloud computing, which meets the current trend requirements and scope towards future needs.

3. Proposed System

For the various platforms on the source and objective clouds, a separate transfer procedure has to be started to pack, copy, download, install, deploy and configure in order to allow interoperability. Nevertheless, there are open problems with how systems communicate with others when the originating cloud does not accept additional cloud services or when a specific system in the original cloud does not support the particular cloud operating system. This type of issue may create unwanted congestion of the data or applications and may affect the whole process of interaction with the cloud. In order to conquer these types of challenges, the work has proposed an efficient application or data interoperability framework containing the following phases: feature extraction, feature selection, cloud server arrangement and cloud server selection.

The proposed framework illustrated in Figure 1 provides a simple exchange of data and services among different applications hosted on different platforms and infrastructures on the cloud, and allows utilization of applications and services across various domains. Therefore, detailed discussion of each phase is done in later sections.

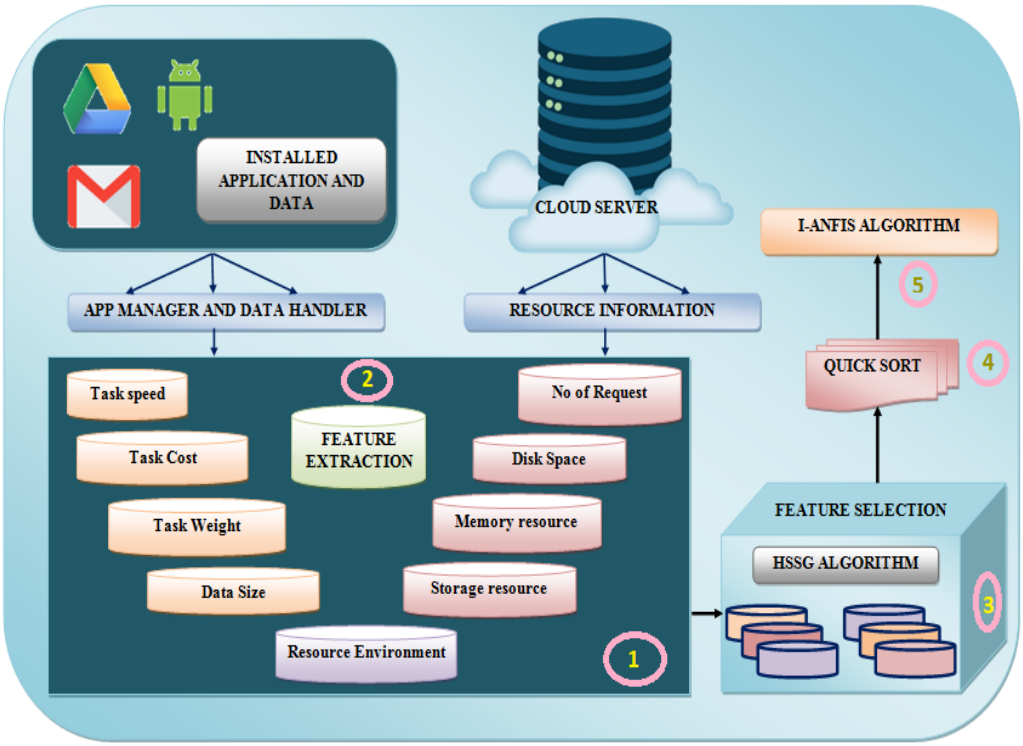


Figure 1. Proposed cloud interoperability framework.

3.1. Feature Extraction

This is the initial stage of the proposed framework. The initialization of the proposed framework with feature extraction provides various features extracted from the application or data and cloud server. Features extraction helps the work to know the important relationships among the data without losing any relevant information and also provides for identifying the redundant features so as to reduce the computational time. Some of the extracted features based on the task manager and cloud server that have been extracted are stated below:

Task Manager

The task manager helps us to know about the currently running applications on a system. The seasonal requests of multiple users are divided into a series of tasks, and these tasks are managed by a task manager. The task manager (T_m) contains several features,

such as task speed (I_1), task cost (I_2), task weight (I_3), and size of the data (I_4). These features are extracted for the process of resource allocation. The explanation for some of the features is given below:

(a) Task speed: It can be calculated by taking the ratio of the turnaround time for allocating cloud resources to the amount time taken to respond to the cloud consumer from the time of the request, which can be expressed as

$$I_1 = \frac{\Gamma_a}{\omega} \tag{1}$$

where, I_1 is the task speed, Γ_a is the turnaround time for allocating cloud resources and ω is the amount of time taken to respond to the cloud consumer from the time of the request.

(b) Task cost: The cost of the task is defined as the amount required to be paid before the request can be attained. The demand cost is determined using the equation:

$$I_2 = \tau \times \Gamma_d \tag{2}$$

where, I_2 is the task cost, τ is the rate at with cloud resource are allocated to the cloud consumer and Γ_d is the amount time taken to respond to the cloud consumer from the time of the request.

(c) Task weight: The amount of weight depends on the speed and cost of the demands, which is expressed as

$$I_3 = x \times (I_1 \times I_2) \tag{3}$$

where, x denotes a constant value and $x \in (0, 1)$.

(d) Data size: Data size of a request can be estimated from the following expression,

$$I_4 = \frac{T_s}{1 + T_s e^2} \tag{4}$$

where, I_4 denotes the user's data size, T_s represents the total size of the user's request and e denotes the allowed probability of committing an error in selecting a small representative of the user's request.

3.2. Cloud Server Features

The features of cloud servers, such as memory resources (I_5), bandwidths (I_6), disk space (I_7) and number of requests (I_8) are extracted [29]. The explanations of some of the cloud server features are given below:

3.2.1. Memory

The cloud memory in a server is used to store, retain and retrieve information by the user and it is calculated as follows,

$$I_5 = (S_L \times S_S) \tag{5}$$

where, I_5 denotes the memory of the cloud server, S_L represents the number of storage locations, and S_s denotes the size of each storage location.

3.2.2. Bandwidth

The transferring rate of data is known as bandwidth. It is calculated by taking the product of several tasks and weight of the applications.

$$I_6 = (N_t \times u_w) \tag{6}$$

where, I_6 denotes the bandwidth of the cloud server and N_t represents a number of tasks, u_w denotes the usage weight.

3.2.3. Disk Space

Disk space of the cloud server is the total used space and the amount of free space available, which is expressed as,

$$I_7 = f_s + u_s \tag{7}$$

where, I_7 denotes the disk space of the cloud server, f_s and u_s represents the free space and the used space of the cloud server.

3.2.4. Number of Requests

The total number of requests to the cloud server is denoted as,

$$I_8 = R_1 + R_2, \ldots R_n \tag{8}$$

where, I_8 denotes the number of requests sent, and $R_1 + R_2, \ldots, R_n$ denotes the requests from the user.

The extracted features are expressed in the form of an equation as stated below:

$$I_{(A,C)} = \{I_1, I_2, I_3, I_4, I_5, I_6 \ldots\ldots I_n\} \tag{9}$$

where, $I_{(A,C)}$ denotes the extraction of the features from "1" to *n*-th values of the application and cloud servers.

4. Proposed Algorithm

The feature selection process contributes to the proposed work by selecting most of the important features from the extracted features to select the cloud server in order to eliminate the irrelevant features that might reduce model accuracy and also may lead to more computational time. In order to optimize the extracted feature, the work developed a Hybrid Squirrel Search Genetic (HSSG) algorithm.

4.1. HSSG Algorithm

The HSSG algorithm [30] provides a complete summary of all the features and the important features required to suit the cloud server for meeting all the application needs. This reduces the error rate and tends to achieve a good accuracy rate. The HSSG algorithm follows the convergence theorem that is the time taken so as to achieve an optimal solution or best solution. The HSSG algorithm takes the extracted features as input and finds the best fitness value between the corresponding squirrel (features) and the server (food). Finally, the minimum value is chosen to be the best fitness value, which is an important feature to map the cloud server. The convergence time of the HSSG algorithm is faster and obtains relevant features so as to provide good accuracy rate. The HSSG algorithm follows certain steps which have been discussed in depth below.

Step 1: Random Initialization

There are n number of flying squirrels (extracted features) in a forest (working space) and the location of *i*-th flying squirrel can be specified by a vector [30]. The location of all the flying squirrels can be represented by the following matrix:

$$Y_{FS} = \begin{bmatrix} I_{1,1} & I_{1,2} & \cdots & \cdots & I_{1,d} \\ I_{2,1} & I_{2,2} & \cdots & \cdots & I_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ I_{n,1} & I_{n,2} & \cdots & \cdots & I_{n,d} \end{bmatrix} \tag{10}$$

where, $I_{i,j}$ represents the *j*-th dimension of *i*-th flying squirrel. A uniform distribution (Equation (2)) is used to allocate the initial location of each flying squirrel in the forest.

$$Y_{FS(i)} = I_L + U(0,1) \times (I_U - I_L) \tag{11}$$

where, I_L and I_U are lower and upper bounds respectively of *i*-th flying squirrel in *j*-th dimension and U (0, 1) is a uniformly distributed random number in the range (0, 1).

Step 2: Fitness evaluation

The fitness value of each flying squirrel is calculated by substituting the decision variables into the user-defined fitness function equation and the respective values are stored in arrays [30] as illustrated below:

$$F_{EVAL} = \begin{bmatrix} F_{EVAL1} & ([I_{1,1} & I_{1,2} & \cdots & I_{1,d}]) \\ F_{EVAL2} & ([I_{2,1} & I_{2,2} & \cdots & I_{2,d}]) \\ & & \vdots & & \\ F_{EVALn} & ([I_{n,1} & I_{n,2} & \cdots & I_{n,d}]) \end{bmatrix} \quad (12)$$

where $F_{EVAL} = (F_{EVAL1}, F_{EVAL2}, \ldots, F_{EVALn})$ is the fitness value for each flying squirrel location.

Step 3: Sorting, declaration and random selection

Furthermore, the quality of the food source (cloud server) based on the fitness value is sorted in ascending order.

$$Sort(F_{EVAL}) = [sorted_F_{EVAL}, sorted_index] \quad (13)$$

Based on the sorting of the food source of each flying squirrel location, the optimization algorithm consists of three trees, such as hickory tree (hickory nuts food source), oak tree (acorn nuts food source) and the normal tree, which evaluates the best nuts or food source, which is located by the flying squirrel. Within this tree, the hickory tree represents the best solution that is the minimal fitness value, which is denoted as (I_{HT}), then the next three best flying squirrels are considered to be on the acorn nut trees (I_{AT}) and the rest are considered as normal trees (I_{NT}).

Step 4: Generate new locations

Under dynamic foraging conditions, that is, in search of an efficient solution, there are three situations. In every situation, it is assumed that the predator is absent so that it encourages the flying squirrel to search for a better solution over a wide area or forest or working space [31]. The three scenarios are:

Scenario 1: The new location generated by the flying squirrels when it tends to move from acorn nut trees to hickory nut trees. The new locations can be generated as follows:

$$I_{AT}^{new} = \begin{cases} I_{AT}^{old} + \lambda_g \kappa \left(I_{HT}^{old} - I_{AT}^{old} \right), & if\ U_1 \geq P_{AB} \\ randomlocation, & Otherwise \end{cases} \quad (14)$$

where, I_{AT}^{new} denotes the newly updated value for flying squirrel of acorn nut trees to hickory nut tree, λ_g represents the gliding distance, κ states the gliding constant, U_1 is a function which returns a value from the uniform distribution on the interval (0, 1), and P_{AB} is the absence of predators.

Scenario 2: The new location generated by the flying squirrel when it tends to move from normal trees to acorn nut trees. The new locations can be generated as follows:

$$I_{NT}^{new} = \begin{cases} I_{NT}^{old} + \lambda_g \kappa \left(I_{AT}^{old} - I_{NT}^{old} \right), & if\ U_2 \geq P_{AB} \\ randomlocation, & Otherwise \end{cases} \quad (15)$$

U_2 is a function which returns a value from the uniform distribution on the interval (0, 1).

Scenario 3: The new location generated by the flying squirrels when it tends to move towards hickory tree if they have already fulfilled their daily energy requirements in this scenario, the new location of squirrels can be generated as follows:

$$\mathrm{I}_{NT}^{new} = \begin{cases} \mathrm{I}_{NT}^{old} + \lambda_g \kappa \left(\mathrm{I}_{HT}^{old} - \mathrm{I}_{NT}^{old} \right), & if\ U_3 \geq P_{AB} \\ randomlocation, & Otherwise \end{cases} \tag{16}$$

U_3 is a function which returns a value from the uniform distribution on the interval (0, 1).

Step 5: Seasonal monitoring condition

Due to seasonal conditions (outliers), the behavior of the flying squirrel may be changed. So as to cope up with the seasonal conditions and to avoid getting trapped within the local minima or solutions, a seasonal monitoring condition is introduced in the algorithm

$$Sea_C^t = \sqrt{\sum_{k=1}^{d} \left(\mathrm{I}_{AT,K}^t - \mathrm{I}_{HT,k} \right)^2} \tag{17}$$

$$Sea_{\min} = \frac{10e^{-6}}{(365)^{t/(t_m/2.5)}} \tag{18}$$

Then, the seasonal monitoring condition is checked. Under the condition of $Sea_C^t < Sea_{\min}$, the winter is over and the flying squirrels which lose their abilities to explore the forest will randomly relocate their searching positions for food sources again:

$$\mathrm{I}_{NT}^{new} = \mathrm{I}_L + Levy(n) \times (\mathrm{I}_U - \mathrm{I}_L) \tag{19}$$

where *Levy* distribution is a powerful mathematical tool to enhance the global exploration capability of most optimization algorithms

$$Levy(x) = 0.01 \times \frac{u_a \times \xi}{|u_b|^{1/\kappa}} \tag{20}$$

where, u_a and u_b are two functions which return a value from the uniform distribution on the interval (0, 1), κ is the constant and ξ is the *Levy* Index and it is calculated as:

$$\xi = \left(\frac{\Gamma(1+\kappa) \times \sin(\pi\kappa/2}{\Gamma((1+\kappa)/2) \times \kappa \times 2^{((\kappa-1)/2)}} \right)^{1/\kappa} \tag{21}$$

where, $\Gamma(\kappa) = (\kappa - 1)!$

Step 6: Crossover and Mutation

In order to achieve an accurate location of the flying squirrel which is in search of food, crossover and mutation are performed on the flying squirrel. That is, among the updated new locations of the flying squirrel (I_{HT}^{new}), a crossover point is chosen randomly. Now, within the selected crossover point, the new location of the flying squirrel is updated until the crossover point is reached. Now, these updated new locations are added the previous locations, and the mutation is performed so as to obtain better selection accuracy. Initially, two squirrels are selected with the best fitness value and after that, crossover probability $\aleph_C$ and mutation probability $\aleph_m$ are calculated using the equation below:

$$\aleph_C = \begin{cases} \aleph_{c_0}, & \mathrm{I}_{NT} \leq \mathrm{I}_{avg}, \\ \aleph_{c_1} \left(\frac{\aleph_{c_0}}{\aleph_{c_1}} \right)^{((\mathrm{I}_{HTMAX} - \mathrm{I}_{NT})/(\mathrm{I}_{HTMAX} - \mathrm{I}_{avg}))}, & \mathrm{I}_{NT} > \mathrm{I}_{avg}, \end{cases} \tag{22}$$

$$\lambda_m = \begin{cases} \lambda_{m_0}, & I''_{NT} \leq I_{avg}, \\ \lambda_{m_1}\left(\frac{\lambda_{m_0}}{\lambda_{m_1}}\right)^{((I_{HTMAX}-I_{NT}|')/(I_{HTMAX}-I_{avg}))}, & I''_{NT} > I_{avg}, \end{cases} \quad (23)$$

where, λ_{c_0} and λ_{m_0} represent the higher crossover and mutation probabilities, λ_{c_1} ($\lambda_{c_1} < \lambda_{c_0}$) and λ_{m_1} ($\lambda_{m_1} < \lambda_{m_0}$) are the lower probabilities, I_{NT} and I''_{NT} are the lower fitness values of the flying squirrel, and I_{HTMAX} and I_{avg} are the optimal and average fitness values in the population.

1. When the fitness of a flying squirrel is greater than the average one in the population (total features), it shows that the squirrel is better, and λ_C and λ_m should be correspondingly reduced.
2. When the fitness of a flying squirrel is smaller than the average one, it indicates that the squirrel has a poor performance, and λ_C and λ_m should keep their initial configurations.
3. When the fitness of a flying squirrel is close to the maximum one, λ_C and λ_m should be kept as small as possible to retain the best individual.

Step 7: Stopping criterion

The algorithm terminates if the maximum number of iterations is satisfied. Otherwise, generating new locations and checking seasonal monitoring conditions are repeated.

Thus, the HSSG algorithms tend to provide the best overall fitness value-based features so as to select the respective cloud server. The selected features are stored in an array for further processing as shown below in the equation:

$$F_{SELECT} = \begin{bmatrix} I_{1,1} & I_{3,2} & \cdots & I_{1,d} \\ I_{2,1} & I_{3,1} & \cdots & I_{2,d} \end{bmatrix} \quad (24)$$

Thus, the pseudo code for the proposed HSSG algorithm is illustrated in Algorithm 1, which gives an overall outline of the proposed HSSG method.

4.2. Cloud Server Arrangements

After selecting the features, they are sorted depending upon the fitness value of the features of application as well as cloud servers. The sorting of the features is carried out by the quick-sort algorithm. The sorting of the features helps the cloud server to improve its reliability.

The key process in quick sort is partitioning. The target of the partitions is F_{SELECT}, given an array and a fitness value of the feature of the array is chosen as the pivot, so the quick-sort algorithm puts the pivot value at its correct position in a sorted array and puts all smaller features before the pivot value and put all greater feature fitness value after the pivot value. Thus, the sorting is represented in the equation below:

$$\begin{cases} QuickSort(F_{SELECT}, low, high) \\ pivot_index = partition(F_{SELECT}, low, high); \\ \text{QuickSort}(F_{SELECT}, low, pivot_index - 1); \\ \text{QuickSort}(F_{SELECT}, high, pivot_index + 1); \end{cases} \quad (if(low < high)) \quad (25)$$

Based on the above equation, the sorting of the features has been done. Now, the features are arranged more reliably so as to avoid any errors while selecting the cloud server, which allows the full utilization of the resource and provides a good accuracy rate. Thus, the sort feature is represented in the equation below:

$$F_{SORT} = \begin{bmatrix} I_{1,1} & I_{1,2} & \cdots & I_{1,d} \\ I_{2,1} & I_{2,2} & \cdots & I_{2,d} \end{bmatrix} \quad (26)$$

Algorithm 1

Input: Extracted Features $Y_{FS} = (I_{1,1}, I_{1,2}, ...I_{1,d})$
Output: Relevant Features
Begin
Initialize the input parameters
Initialize the location of squirrel randomly
While (the stopping criterion is not satisfied)
Calculate the fitness value using,
$F_{EVAL} = F_{EVAL1}([I_{1,1}, I_{1,2}, ...I_{1,d}])$
for t = 1 to U_1
Generate new location scenario1:
if $f(U_1 \geq P_{AB})$
$I_{AT}^{new} = I_{AT}^{old} + \lambda_g K(I_{NT}^{old} - I_{AT}^{old})$
else
Random position of search space
end for
for t = 1 to U_2
Generate new location scenario2:
if $f(U_2 \geq P_{AB})$
$I_{NT}^{new} = I_{NT}^{old} + \lambda_g K\left(I_{AT}^{old} - I_{NT}^{old}\right)$
else
Random position of search space
end for
for t=1 to U_3
Generate new location scenario3:
if $f(U_3 \geq P_{AB})$
$I_{NT}^{new} = I_{NT}^{old} + \lambda_g K\left(I_{HT}^{old} - I_{NT}^{old}\right)$
else
Random position of search space
end for
Calculate seasonal constant Sea_C^t *and* Sea_{min}
if $Sea_C^t < Sea_{min}$
Randomly relocate flying squirrels using,
$I_{NT}^{new} = I_L + Levy(n) \times (I_U - I_L)$
end
Evaluate crossover probability λ_c and mutation probability λ_m
Locate the new location of squirrel on hickory nut tree obtained by crossover
and mutation is the final optimal solution
End While
End Begin

4.3. Cloud Server Selection

Cloud server selection is the most important step for cloud interoperability. Based on the sorted features the selection of a cloud server takes place. As the existing techniques faced an inappropriate cloud server selection and improper utilization of resources which caused a high error rate. In order to select the cloud server, the work uses the improved adaptive neuro-fuzzy inference system (I-ANFIS) [32]. The primary goal of I-ANFIS is to allocate an appropriate resource to the application or software with maximum resource utilization and minimum delay. The I-ANFIS provides accurate selection of the cloud server due to the Gaussian kernel membership function which tends to reduce the possibility of error. The I-ANFIS structural layers are presented below in Figure 2. The workings behind each structural layer of the I-ANFIS are stated below:

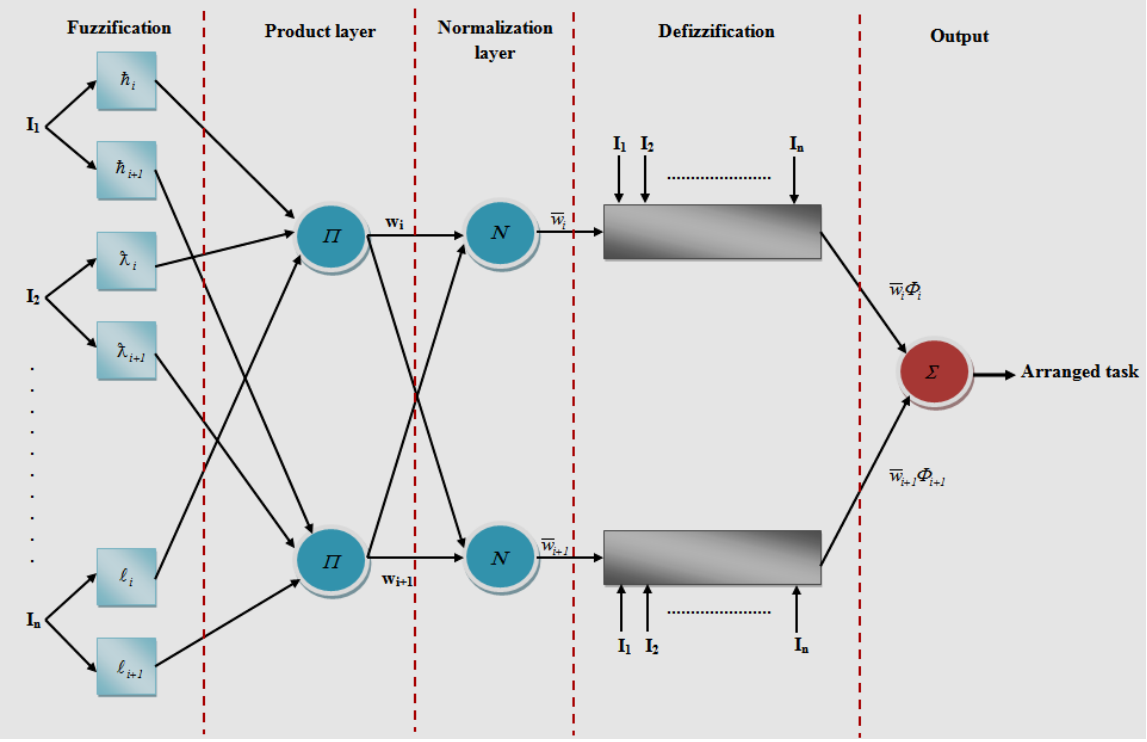


Figure 2. Structure of I-ANFIS.

Fuzzification layer: This transforms the crisp input that is the collection of the selected features of the application and the VM servers into linguistic labels with a degree of membership as explained below in Equations (27) and (28).

$$\text{Rule1}: \text{ if } \mathrm{I}_i \text{ is} \hbar_i \text{and } \mathrm{I}_{i+1} \text{is} \hat{\chi}_i, \text{ then } \Phi_i = x_i \mathrm{I}_i + y_i \mathrm{I}_{i+1} + \Im_i \tag{27}$$

$$\text{Rule2}: \text{ } \mathrm{I}_i \text{ is } \hbar_{i+1} \text{and } \mathrm{I}_{i+1} \text{is} \hat{\chi}_{i+1}, \text{ then } \Phi_{i+1} = x_{i+1} \mathrm{I}_i + y_{i+1} \mathrm{I}_{i+1} + \Im_{i+1} \tag{28}$$

where $\hbar_i$, χ_i, $\hbar_{i+1}$ and χ_{i+1} signify the fuzzy sets; and I_i and I_{i+1} denote the input set (task features and VM status details). $\Im_i$, x_i, y_i, $\Im_{i+1}$, x_{i+1} and y_{i+1} values are the parameter set and Φ is a first-order polynomial and represents the outputs of the first-order Sugeno fuzzy inference system. I-ANFIS is composed of five functional layers. The function of each layer is given as follows:

The output of each node in the first layer is defined as,

$$\prod_i^1 = \Omega_i(\Phi_i) \quad , i = 1, 2, \ldots\ldots \tag{29}$$

where, $\prod_i^1$ signifies the output of layer 1, Φ_i is the collection of input nodes and Ω_i represents the Gaussian kernel membership function (GKMF). To improve the performance of I-ANFIS the proposed work performs the Gaussian kernel membership (GKMF) function, which is computed as follows,

$$\Omega_i = \exp\left(-\frac{||\mathrm{I}_i - y_i||^2}{2\varpi^2}\right) \tag{30}$$

where, y_i is the central value and standard deviation ϖ which should be greater than 0 ($\varpi > 0$). If the Standard Deviation (SD) value is smaller, then the membership value will be accurate in identifying the features. Membership values are computed for each input value in I.

Product layer: Every node in layer 2 is a fixed node, which admits first layer input values and turns to represent fuzzy sets of respective input variables as a membership

function. The output of each node multiplies the entire signal coming towards it and represents the firing strength of a rule.

$$\prod_i^2 = w_i = \Omega_i \mathrm{I}_i \times \Omega_i \mathrm{I}_{i+1} \tag{31}$$

Here, $w_i \rightarrow$ firing strength.

Normalization layer: The output of each node in this layer represents their normalization position to the firing strengths from the previous layer.

$$\prod_i^3 = E_y(\overline{w}_i) = \frac{E_y(w_i)}{\sum_{i=1}^{n} E_y(w_i)}, \quad i = 1, 2, \ldots \tag{32}$$

Here, $E_y(\overline{w}_i)$ signifies normalized firing strength entropy value.

Defuzzification layer: Each node of this layer gives a firing strength output of the first order Sugeno-type fuzzy if-then rule as follows,

$$\prod_i^4 = E_y(\overline{w}_i). \quad \Phi_i \tag{33}$$

Output layer: It has only one node, and it calculates the sum of all the outputs coming from the nodes of the previous layer to produce the overall I-ANFIS output as in Equation (24).

$$\prod_i^5 = \sum_{i=1}^{n} E_y(\overline{w}_i).\ \Phi_i = \frac{\sum_{i=1}^{n} E_y(\overline{w}_i). \quad \Phi_i}{\sum_{i=1}^{n} E_y(\overline{w}_i)} \tag{34}$$

Here, $\prod_i^5$ is the overall output of I-ANFIS. That is to say, the output $\prod_i^5$ signifies the best cloud server that is regarded as a chosen VM to execute the application or software.

Thus, the overall outline of the proposed I-ANFIS is given in the form of pseudo code illustrated in Algorithm 2.

Algorithm 2

Input: Application and VM features
Output: Cloud Server Selection
Begin
  Initialize the linear parameters, $\Im_i, x_i, y_i$
    For i = 1 to n
      Generate fuzzy output from crisp input using,
        $\prod_{\mathrm{i}}^1 \leftarrow$ Gaussian kernal membership function $(\mathrm{I}_1 : c, \sigma) = \Omega_i(\mathrm{I}_i)$
      Calculate firing strength for each rule using,
        $\prod_{\mathrm{i}}^2 \leftarrow \Omega_i(\mathrm{I}_i) \times \Omega_{i+1}(\mathrm{I}_{i+1}) = \omega_i$
      Normalize the firing strength using,
        $\prod_{\mathrm{i}}^3 \leftarrow \frac{\mathrm{E_y}(\omega_i)}{\sum \mathrm{E_y}(\omega_i)} = o_i$
      Calculate Defuzzify the output of 3rd layer using,
        $\prod_{\mathrm{i}}^4 \leftarrow \mathrm{E_y}(o_i)(x_i(\mathrm{I}_i) \times y_{i+1}(\mathrm{I}_{i+1})) = \mathrm{E_y}(o_i) \times \Phi_i$
      Evaluate the output by converting fuzzy result into crisp output using
        $\prod_{\mathrm{i}}^5 \leftarrow \sum \mathrm{E_y}(o_i) \times \Phi_i$
    End for
End Begin

5. Discussion and Results

This section provides a detailed analysis of the proposed efficient application cloud interoperability based on various performance metrics and thereafter a comparative analysis is done with the existing methodologies in order to state its efficiency.

5.1. Performance Analysis

Based on performance metrics, such as accuracy, precision, F-measures, recall, fitness value vs. iteration, sorting time and training time, the evaluation of the proposed methods is carried out. The analysis of the proposed work is split based on proposed feature selection, sorting of features and selection of cloud server.

5.1.1. Performance Analysis of Proposed HSSG Algorithm for Selecting Features

The proposed HSSGA feature selection method is analyzed based on iteration vs. fitness value and compared with existing techniques, such as lower pollination algorithm (FPA), cat swarm optimization algorithm (CSO), genetic algorithm (GA), and squirrel search algorithm (SSA). The evaluation of iteration vs. fitness for the proposed method is tabulated.

Table 1 illustrates the evaluation of the proposed HSSG optimization algorithm based on iteration vs. fitness value to select the important features in order to achieve good accuracy and low complexity time. To elaborate its efficiency, it was analyzed with existing techniques, such as FPA, CSO, GA and SSA [33]. Basically, iteration vs. fitness states that the method adheres to best fitness value and reduces the computational time within a minimal iteration. According to that, the analysis illustrated that the proposed HSSG algorithm tends to achieve a fitness value of 102 for the 25th iteration whereas the existing FPA, CSO, GA and SSA achieve fitness values of 79, 84, 89 and 92, respectively. From the analysis, it is known that the proposed HSSG algorithm tends to achieve the best fitness value compared to existing methods. The graphical analysis of the proposed algorithm with the existing method is represented in Figure 3.

Table 1. Evaluation of proposed hybrid squirrel search genetic (HSSG) algorithm based on iteration vs. fitness.

Optimization Technique	FPA	CSO	GA	SSA	Proposed
5	35	39	41	53	63
10	50	54	57	65	73
15	59	63	68	70	86
20	67	71	75	85	96
25	79	84	89	92	102

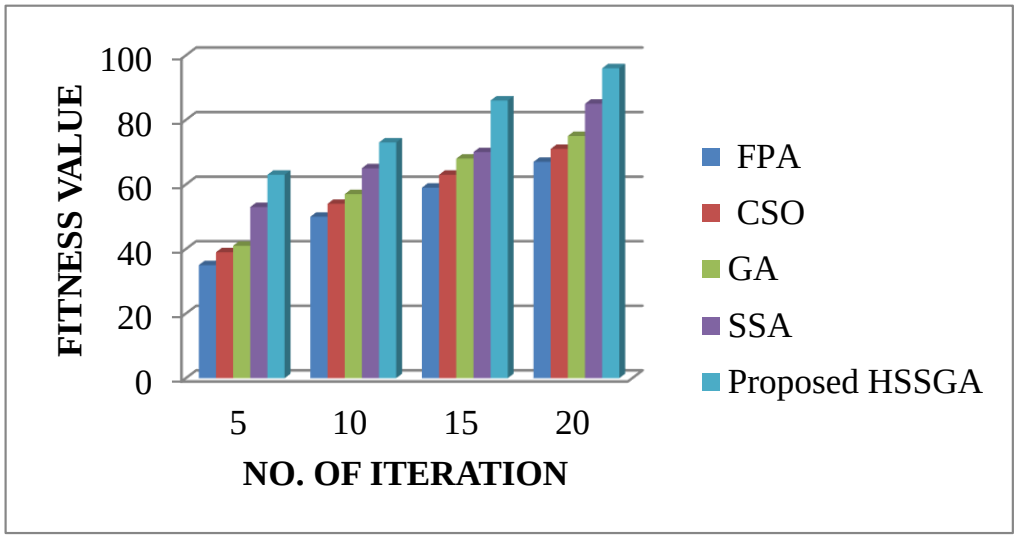


Figure 3. Comparative analysis of proposed HSSG algorithm based on iteration vs. fitness.

From Figure 3 it can be graphically analyzed and illustrated that the proposed HSSG algorithm tends to achieve better fitness outcomes for the respective iteration than the existing FPA, CSO, GA and SSA algorithms. The better fitness values within a low iteration may help to decrease the time complexity and may also help achieve good accuracy without further iteration proceeding processes.

5.1.2. Performance Analysis of Proposed I-ANFIS for Selection of Cloud Server

The proposed I-ANFIS is analyzed based on various performance metrics, such as accuracy, precision, recall, training time, memory usage and F-measure with various existing methods, such as Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Artificial Neural Network (ANN) and ANFIS. The evaluation of the performance metrics for the proposed I-ANFIS and the existing methods is tabulated.

Table 2 illustrates the analysis of various performance metrics, such as accuracy, recall, F-measure and precision, for the proposed method and evaluated against the existing methods, such as SVM, KNN, ANN and ANFIS [34]. The analysis illustrates that the proposed I-ANFIS achieves precision, recall, F-measure and accuracy values of 87.58%, 91.19%, 90.51% and 94.24%, respectively, which ranges between 87.58 and 94.24%, whereas the existing methods tend to achieve metric values ranging between 84.33 and 92.55% which is relatively low compared to the proposed method. The metrics estimate the efficiency of the proposed I-ANFIS based on true positive (TP), false positive (FP), true negative (TN) and false negative (FN). Therefore, the proposed method tends to determine the best server that suits the corresponding application features. The graphical analysis of the proposed method along with the existing method is illustrated in Figure 4.

Table 2. Evaluation of proposed I-ANFIS based on precision, accuracy, recall and F-measure.

Techniques/Performance Metrics	Existing SVM	Existing KNN	Existing ANN	Existing ANFIS	Proposed I-ANFIS
Precision	84.33	85.01	85.22	85.5	87.58
Recall	87.11	87.87	88.12	88.36	91.19
F-measure	87.02	87.33	88.57	89.91	90.51
Accuracy	88.33	89.66	91.55	92.55	94.24

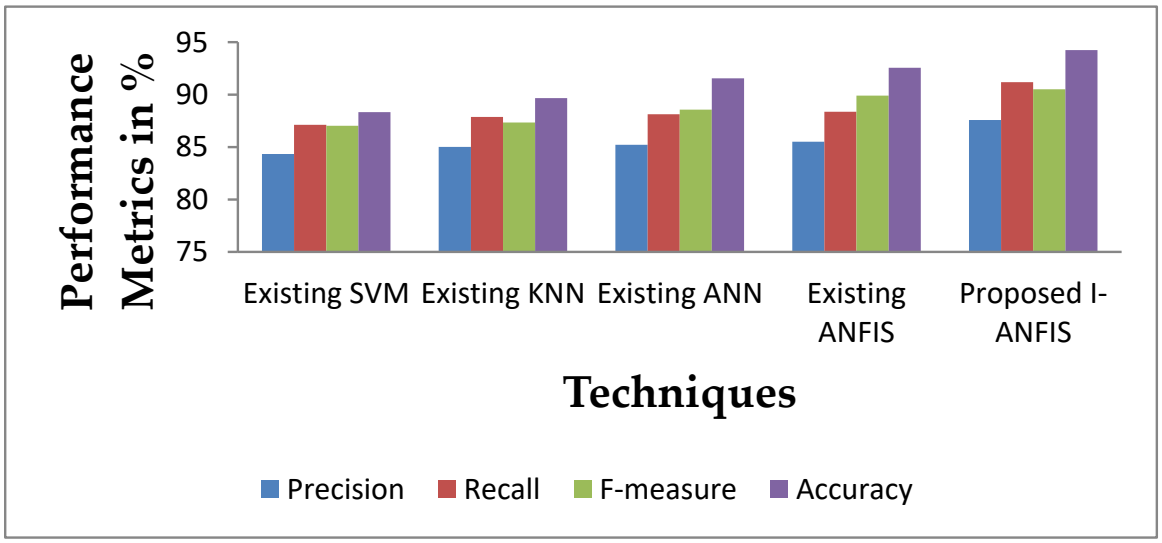
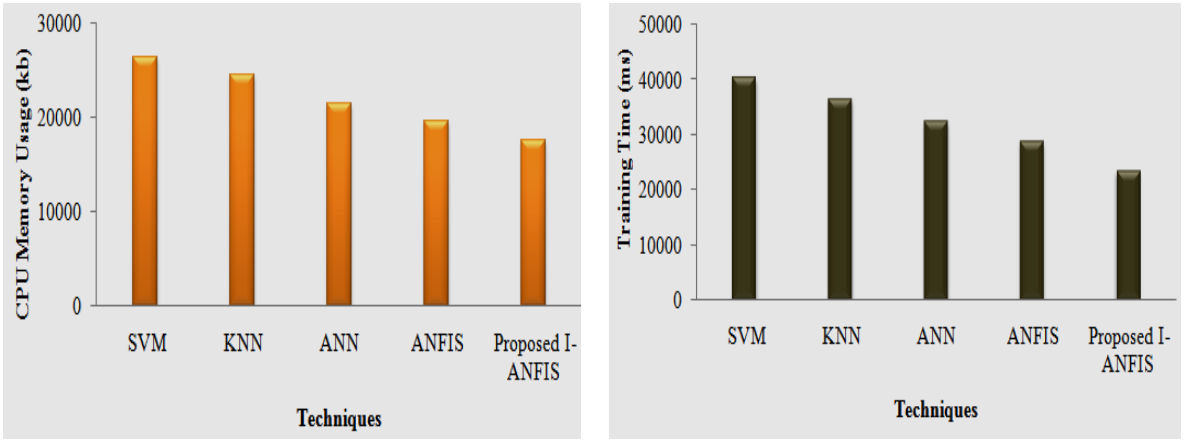


Figure 4. Comparative analysis of proposed I-ANFIS technique based on precision, accuracy, recall and F-measure.

Figure 4 illustrates the graphical analysis of the metrics, such as accuracy, precision, recall and F-measure, for the proposed I-ANFIS [35] technique with the existing SVM, KNN, ANN and ANFIS. The analysis illustrates that the proposed techniques tend to detect the cloud server with better accuracy and tends to reduce the false negative and false positive rate by achieving a better recall and precision value. However, the existing techniques tend to achieve comparatively fewer metric values.

Training Time and Memory Usage analysis of Proposed I-ANFIS

Based on training time and memory usage, the proposed I-ANFIS is evaluated with existing SVM, KNN, ANN and ANFIS. From Figure 4 it can be stated that the proposed I-ANFIS tends to utilize less training time as well as CPU and memory usage in order to perform more efficiently. Figure 5a illustrates that the proposed technique achieves a training time of 23,457 ms to train a dataset, which is comparatively better compared to existing SVM, KNN, ANN and ANFIS methods, which achieve training times of 40,325 ms, 36,457 ms, 32,457 ms and 28,745 ms, respectively. Figure 5b illustrates that the proposed technique consumes 17,515 kb of memory, whereas the existing SVM, KNN, ANN, and ANFIS technique achieve memory usage of 26,354 kB, 24,568 kB, 21,457 kB and 19,634 kB respectively, which is comparatively higher than the proposed method.



(**a**) (**b**)

Figure 5. Comparative analysis of proposed I-ANFIS technique based on: (**a**) CPU memory usage and (**b**) Training time.

Thus, the proposed I-ANFIS attempts to achieve a better selection of server based on the application and cloud server features by requiring low time as well as low usage of memory to avoid complexity.

5.1.3. Performance Analysis of Proposed Quick Sort for Sorting the Selection Features

The performance analysis of the quick-sort algorithm is done based on sorting time with the existing heap, bubble, insertion and merge sort algorithms in order to compare the time consumption between the algorithms and to state its efficiency. The graphical analysis of the sorting time for the proposed and existing algorithms is represented in Figure 6.

Figure 6 represents the graphical analysis of the sorting time taken by the proposed quick-sort algorithm and the existing heap, bubble, insertion and merge sort algorithms so as to compute the best algorithm. The sorting algorithm helps the I-ANFIS algorithm to select the cloud server with respect to the applications. There should be a low sorting time in order to reduce the computational time. According to that, the proposed methods tend to achieve a sorting time of 6322 ms, whereas, the existing heap, bubble, insertion and merge sort algorithms tend to achieve sorting times of 11,969 ms, 11,347 ms, 10,457 ms and 9478 ms, respectively, which is relatively high compared to the proposed algorithm.

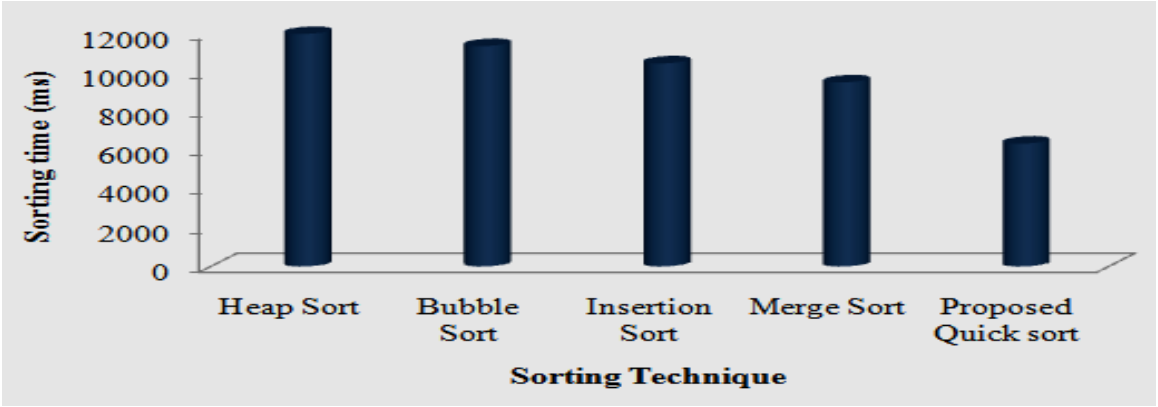


Figure 6. Comparative analysis of proposed quick sort based on sorting time.

Thus, the proposed framework for cloud interoperability achieves a unilateral provision of computing capabilities of a cloud server without the help of human interaction and allows utilization of applications and services across various domains.

6. Conclusions

The work introduces an efficient application cloud interoperability framework, which is used to manage various applications or data and to execute over the respective cloud server. The framework provides efficient solutions for the implementation of collaboration among services for better quality in providing services. In order to overcome the technological and non-technological challenges, the framework has developed a hybrid search optimization algorithm for selecting relevant features and the I-ANFIS technique to select the cloud server. It avoids most of the redundant data and utilizes less memory so as to avoid complexity or congestion. The efficiency of the proposed methods is evaluated using various performance metrics, and overall, the evaluation states that the framework achieved an accuracy rate of 94.24%, precision of 87.58%, recall of 91.19% and F-measure of 90.51%, which is highly efficient compared to existing works. Therefore, in future the proposed framework can be improvised by indulging load balancing techniques for handling multiple requests and deep learning techniques to increase the accuracy rate of selecting a cloud server.

Author Contributions: Conceptualization, C.R. and P.M.; methodology, C.R.; software, C.R.; validation, C.R. and P.M.; formal analysis, C.R.; investigation, C.R. and P.M.; resources, C.R.; data curation, C.R.; writing—original draft preparation, C.R.; writing—review and editing, C.R. and P.M.; visualization, C.R. and P.M.; C.R.; supervision, P.M.; project administration, C.R.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: We would like acknowledge K. Jayashree, CSE Department, Rajalakshmi Engineering College, Chennai, India, for providing technical support.

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

References

1. Seo, D.; Jeon, Y.-B.; Lee, S.-H.; Lee, K.-H. Cloud computing for ubiquitous computing on M2M and IoT environment mobile application. *Clust. Comput.* **2016**, *19*, 1001–1013. [CrossRef]
2. Delaram, J.; Valilai, O.F. A Novel Solution for Manufacturing Interoperability Fulfillment using Interoperability Service Providers. *Procedia CIRP* **2017**, *63*, 774–779. [CrossRef]
3. Lagar-Cavilla, H.A.; Tolia, N.; de Lara, E.; Satyanarayanan, M.; O'Hallaron, D. Interactive Resource-Intensive Applications Made Easy. In *Middleware 2007*; Lecture Notes in Computer Science; Cerqueira, R., Campbell, R.H., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4834. [CrossRef]
4. Lachmann, A.; Clarke, D.J.; Torre, D.; Xie, Z.; Ma'Ayan, A. Interoperable RNA-Seq analysis in the cloud. *Biochim. Biophys. Acta (BBA)-Bioenerg.* **2020**, *1863*, 194521. [CrossRef] [PubMed]
5. Pedone, G.; Mezgár, I. Model similarity evidence and interoperability affinity in cloud-ready Industry 4.0 technologies. *Comput. Ind.* **2018**, *100*, 278–286. [CrossRef]
6. Domanal, S.G.; Reddy, G.R.M. An efficient cost optimized scheduling for spot instances in heterogeneous cloud environ-ment. *Future Gener. Comput. Syst.* **2018**, *84*, 11–21. [CrossRef]
7. Gill, N.K.; Singh, S. A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers. *Future Gener. Comput. Syst.* **2016**, *65*, 10–32. [CrossRef]
8. Jardim-Goncalves, R.; Popplewell, K.; Grilo, A. Sustainable interoperability: The future of Internet based industrial enterprises. *Comput. Ind.* **2012**, *63*, 731–738. [CrossRef]
9. Marković, M.; Gostojić, S. A knowledge-based document assembly method to support semantic interoperability of enterprise information systems. *Enterp. Inf. Syst.* **2020**, 1–20. [CrossRef]
10. Emeakaroha, V.C.; Healy, P.; Morrison, J.P. Facilitating Cloud Federation Management via Data Interoperability. In *Smart Cities*; Springer Nature: Berlin/Heidelberg, Germany, 2017; pp. 227–253.
11. Wang, X.; Wang, L.; Gördes, R. Interoperability in cloud manufacturing: A case study on private cloud structure for SMEs. *Int. J. Comput. Integr. Manuf.* **2017**, *31*, 653–663. [CrossRef]
12. Jabbar, S.; Ullah, F.; Chakour, K.; Khan, M.; Han, K. Semantic Interoperability in Heterogeneous IoT Infrastructure for Healthcare. *Wirel. Commun. Mob. Comput.* **2017**, *2017*, 1–10. [CrossRef]
13. Mourad, M.H.; Nassehi, A.; Schaefer, D.; Newman, S.T. Assessment of interoperability in cloud man-ufacturing. *Robot. Comput. Integr. Manuf.* **2020**, *61*, 101832. [CrossRef]
14. Khalfallah, M.; Figay, N.; Da Silva, C.F.; Ghodous, P. A cloud-based platform to ensure interoper-ability in aerospace industry. *J. Intell. Manuf.* **2016**, *27*, 119–129. [CrossRef]
15. Habibi, M.; Fazli, M.; Movaghar, A. Efficient distribution of requests in federated cloud computing en-vironments utilizing statistical multiplexing. *Future Gener. Comput. Syst.* **2019**, *90*, 451–460. [CrossRef]
16. Teixeira, B.; Pinto, T.; Silva, F.; Santos, G.; Praça, I.; Vale, Z. Multi-Agent Decision Support Tool to Enable Interoperability among Heterogeneous Energy Systems. *Appl. Sci.* **2018**, *8*, 328. [CrossRef]
17. Silva, T.W.; Morais, D.C.; Andrade, H.G.; Lima, A.M.; Melcher, E.U.; Brito, A.V. Envi-ronment for integration of distributed heterogeneous computing systems. *J. Internet Serv. Appl.* **2018**, *9*, 4. [CrossRef]
18. Esposito, C. Interoperable, dynamic and privacy-preserving access control for cloud data storage when integrating heterogeneous organizations. *J. Netw. Comput. Appl.* **2018**, *108*, 124–136. [CrossRef]
19. Liu, F.; Tong, J.; Mao, J.; Bohn, R.; Messina, J.; Badger, L.; Leaf, D. NIST cloud computing reference architecture. *NIST Spec. Publ.* **2011**, *500*, 1–28.
20. Pang, L.Y.; Zhong, R.Y.; Fang, J.; Huang, G.Q. Data-source interoperability service for heterogeneous information integra-tion in ubiquitous enterprises. *Adv. Eng. Inform.* **2015**, *29*, 549–561. [CrossRef]
21. Huedo, E.; Montero, R.S.; Moreno, R.; Llorente, I.M.; Levin, A.; Massonet, P. Interoperable federated cloud networking. *IEEE Internet Comput.* **2017**, *21*, 54–59. [CrossRef]
22. Castañé, G.G.; Xiong, H.; Dong, D.; Morrison, J.P. An ontology for heterogeneous resources man-agement interoperability and HPC in the cloud. *Future Gener. Comput. Syst.* **2019**, *88*, 373–384. [CrossRef]
23. Nodehi, T.; Jardim-Goncalves, R.; Zutshi, A.; Grilo, A. ICIF: An inter-cloud interoperability frame-work for computing resource cloud providers in factories of the future. *Int. J. Comput. Integr. Manuf.* **2017**, *30*, 147–157. [CrossRef]
24. Rezaei, R.; Chiew, T.K.; Lee, S.P.; Aliee, Z.S. A semantic interoperability framework for software as a service systems in cloud computing environments. *Expert Syst. Appl.* **2014**, *41*, 5751–5770. [CrossRef]
25. Anithakumari, S.; Chandrasekaran, K. Adaptive Resource Allocation in Interoperable Cloud Services. In *Advances in Intelligent Systems and Computing*; Springer Nature: Berlin/Heidelberg, Germany, 2019; pp. 229–240.
26. Kertész, A.; Kecskemeti, G.; Brandic, I. An interoperable and self-adaptive approach for SLA-based service vir-tualization in heterogeneous Cloud environments. *Future Gener. Comput. Syst.* **2014**, *32*, 54–68. [CrossRef]
27. Loulloudes, N.; Sofokleous, C.; Trihinas, D.; Dikaiakos, M.D.; Pallis, G. Enabling Interoperable Cloud Application Management through an Open Source Ecosystem. *IEEE Internet Comput.* **2015**, *19*, 54–59. [CrossRef]
28. Sehgal, S.; Erdelyi, M.; Merzky, A.; Jha, S. Understanding application-level interoperability: Scal-ing-out MapReduce over high-performance grids and clouds. *Future Gener. Comput. Syst.* **2011**, *27*, 590–599. [CrossRef]

29. Ramasamy, V.; Pillai, S.T. An effective HPSO-MGA optimization algorithm for dynamic re-source allocation in cloud environment. *Clust. Comput.* **2020**, *23*, 1711–1724. [CrossRef]
30. Jain, M.; Singh, V.; Rani, A. A novel nature-inspired algorithm for optimization: Squirrel search algorithm. *Swarm Evol. Comput.* **2019**, *44*, 148–175. [CrossRef]
31. Zheng, T.; Luo, W. An Improved Squirrel Search Algorithm for Optimization. *Complexity* **2019**, *2019*, 6291968. [CrossRef]
32. Hu, H.; Zhang, L.; Bai, Y.; Wang, P.; Tan, X. A Hybrid Algorithm Based on Squirrel Search Algorithm and Invasive Weed Optimization for Optimization. *IEEE Access* **2019**, *7*, 105652–105668. [CrossRef]
33. Tubishat, M.; Ja'Afar, S.; Alswaitti, M.; Mirjalili, S.; Idris, N.; Ismail, M.A.; Omar, M.S. Dynamic Salp swarm algorithm for feature selection. *Expert Syst. Appl.* **2021**, *164*, 113873. [CrossRef]
34. Poul, A.K.; Shourian, M.; Ebrahimi, H. A Comparative Study of MLR, KNN, ANN and ANFIS Models with Wavelet Transform in Monthly Stream Flow Prediction. *Water Resour. Manag.* **2019**, *33*, 2907–2923. [CrossRef]
35. Stallings, R.; Samanta, B. Prediction of University Enrollment Using Computational Intelligence. In Proceedings of the 2014 IEEE Symposium on Swarm Intelligence, Orlando, FL, USA, 9–12 December 2014; pp. 1–8. [CrossRef]