Journal of Applied Research and Technology

www.jart.icat.unam.mx

Journal of Applied Research and Technology 23 (2025) 108-119

Original

Linux cluster programming for high availability with an Oracle instance

J. I. Vega-Luna[a]* • G. Salgado-Guzmán[a] • J. F. Cosme-Aceves[a] • V. N. Tapia-Vargas[a] • E. A. Andrade-González[b]

[a]*Digital Systems Area, Electronics Department, Universidad Autónoma Metropolitana (UAM), Mexico City, Mexico*
[b]*Communications Area, Electronics Department, Universidad Autónoma Metropolitana (UAM), Mexico City, Mexico*

Received 06 09 2024; accepted 09 26 2024
Available 04 30 2024

Abstract: This paper presents programming performed in a cluster of two Linux servers to ensure the high availability of an Oracle instance. The problem of database downtime is proposed to be solved when a component of a server or the entire server fails and there is no service to the database users. The objective was to develop programming to start, stop, and monitor the status of an Oracle instance on one of the two Linux servers in a cluster in case of contingency or maintenance of the other server. The methodology followed divided the programming into five modules: cluster manager, package manager, monitoring module, Host Bus Adapter (*HBA*) port manager, and network port manager. High availability was achieved by creating a package that contained the Oracle instance and resources to start it on the cluster servers. The package is assigned an IP address to which users of the instance are connected. The contribution of this work is to provide a low-cost solution, compared to existing commercially similar systems, with a quick response and easy implementation that allows a company or institution to continue working after a hardware or software failure. The startup time of the Oracle instance package after contingency on a server was 20 s, which was the time when the cluster application was not available to the user. The results of the methodology were a cluster that eliminates the points of failure represented by *LAN* ports, *HBA*s, hard disk drives and an entire server on which a mission-critical application is running. Without the use of the cluster, the user must wait for the repair of the failed component(s) to restore the company's operation. The cost of the cluster is 10% of the cost of an equivalent commercially available solution. The application failover time is 20 s, which is one-tenth of the time achieved in other solutions that use a proprietary operating system.

Keywords: Cluster, contingency, high availability, Linux, Oracle, servers.

*Corresponding author.
E-mail address: vlji@azc.uam.mx (J. I. Vega-Luna).
Peer Review under the responsibility of Universidad Nacional Autónoma de México.

1. Introduction

In recent years, the development of information and communication technologies has led to the growth of companies and institutions, allowing them to process information more efficiently and use increasingly sophisticated applications that require computing resources. This has created the need for proactive measures to protect critical information and applications, reduce and avoid losses owing to risks and contingencies in the operating environment, and ensure the provision of services to clients and users. These measures include Business Continuity Plans (*BCP*) and Disaster Recovery Plans (*DRP*) (Ferdousi et al., 2020; Mudassir et al., 2021; Wan & Zhu, 2020). Both aim to restore an organization's operations after contingency. The *DRP* is part of the *BCP*, and its mission is to restore the systems, services, and critical IT infrastructure that supports the business to minimize the effects of a disaster and to resume the operation of the company. The *BCP* is the methodology used by the organization when it cannot work normally after a disaster and recovers critical processes in the operation of the business. A *BCP* is a set of plans that includes Business Recovery Plans (*BRP*), Occupant Emergency Plans (*OEP*), Continuity of Operations Planning (*COP*), Incident Management Plan (*IMP*), and *DRP*. A *BCP* includes all operational processes in an organization (Aziz & Jambari, 2019; Tomás et al., 2020). Disasters are due to natural factors, such as hurricanes, tornadoes, earthquakes, and floods, as well as power failures, programming errors, hardware and software updates, hardware failures, and attacks, such as sabotage, theft of information, or damage to equipment (Tian et al., 2023).

The *DRP* is a plan that indicates the strategy and actions to follow to reactivate the technological infrastructure and restore the critical services or applications of the organization after a contingency, without affecting the information to resume the operation as soon as possible through high availability and recovery schemes. When a disaster occurs and there is no *DRP*, the consequences and cost of recovery of the operation are much higher than those of the timely development of the *DRP* (Petrenko, 2021).

The consequences of a disaster are costly, can damage the technological infrastructure, and even put the existence of the company at risk. According to data from the IDC consulting firm, organizations that cannot resume operations within ten days of a disaster will probably not survive (Schwartz & Goodwin, 2022; Smith, 2020). The characteristics of a *DRP* depend on the needs of each organization; therefore, it is important to meet two requirements to determine its efficiency. The first is the Recovery Time Objective (*RTO*) and the second is the Recovery Point Objective (*RPO*). *RTO* indicates the acceptable time to restore services after a disaster, and *RPO* specifies the data recovery point in time, that is, the acceptable amount of information loss (Faramondi et al., 2024).

When a *DRP* is designed, it is important to carry it out under international norms and standards in companies that manage data centers and provide *DRP* and *BCP* services. Almost all standards, such as NIST SP 800-34 (NIST, 2024) and ISO/IEC 24762 (Strawser, 2019), require, among other things, that the base component of a *DRP* be a computer cluster that allows business continuity after a contingency.

A computer cluster is a set of at least two servers, called nodes, tightly coupled across a data network with sufficient hardware redundancy that collaborates to keep mission-critical applications available to users. A cluster is part of a *DRP* and provides a high availability of mission-critical applications, reducing the impact of downtime owing to contingencies, disasters, unplanned events, and scheduled maintenance tasks without compromising information integrity and performance. Depending on the design and scope of the *DRP*, cluster nodes can be in the same datacenter, in different datacenters, in the cloud, or by using a hybrid schema (Liu et al., 2021).

Each node has redundancy in disks, disk controller cards or Host us Adapters (*HBA*) and network ports, so that in the event of the failure of one of these components, its function is performed by the redundant component. If one of the nodes fails, the mission-critical application that runs on it starts at another node in the cluster. The programming that implements a cluster and controls the redundancy of components and its operation is divided into five modules: cluster manager, package manager, monitoring module, *HBA* port manager, and network port manager.

The functions of the cluster manager are as follows: 1) start and stop the cluster and nodes; 2) register the operational status of its components; and 3) manage the heartbeat signal. The two operating schemes of a cluster are active/active and active/standby. First, one of the nodes runs mission-critical applications and the other is used as a backup when the first node fails. In the second scenario, the applications run on both nodes, allowing load balancing and more efficient use of the computing system.

The package manager has the function of starting and stopping packages on the cluster nodes. A package groups a mission-critical application, the storage resources it must run, and a volatile or relocatable IP address. The package can start at any node in the cluster, thus making the application highly available. Application users connect to the relocatable IP address of a package. The package and application can be moved between nodes in the cluster, which is useful when a node fails or when it needs to be released for maintenance and to continue serving users. Typically, the application is a

productive database, and the package manager is responsible for starting, stopping, and verifying its operation through the monitoring module (Saxena et al., 2022).

The network port manager is responsible for controlling the use of these ports and ensuring high availability of network access. This allows you to configure, as stated above, an active/active or active/standby design on the network ports. Similarly, the function of the *HBA* port manager is to provide high-availability access to application disks from the cluster nodes.

The objective of this study is to program a two-node cluster and a package that incorporates an Oracle instance and a database. The cluster is composed of two Linux servers as shown in Figure 1. It can be used in small and medium-sized organizations that need to provide continuous services to clients and users.

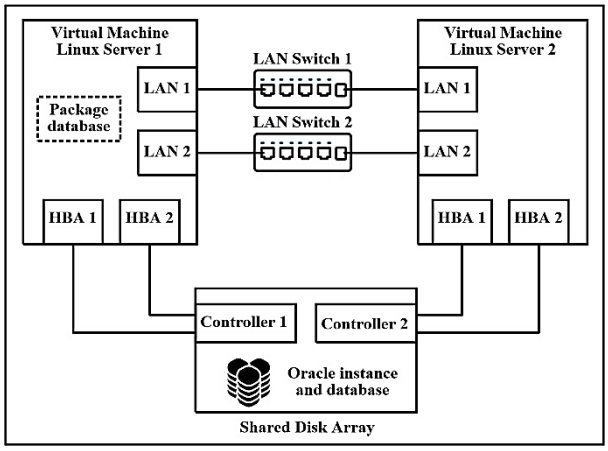

Figure 1. Cluster with two Linux servers.

The cluster incorporates a shared disk array between the nodes, and the redundancy of the *HBA* and network ports. The Oracle 21c instance and database were installed on the disk array, both of which constitute the mission-critical application. The Oracle database manager was used because it is the most used in organizations, and the Linux operating system because it is the most currently used in business servers owing to its robustness, low cost, and support from all server manufacturers, including some free distributions. The cluster servers are virtual machines.

Currently, there are solutions and products on the market from different hardware and software providers used to configure and implement a cluster of computers with a Linux operating system. For example, the following solutions are available: high-availability add-on from Red Hat (Red Hat, 2024), HPE Serviceguard for Linux (HPE, 2024), and Site Recovery Manager from VMware (VMWare, 2024), among others. The software and consulting services used to implement the cluster with these solutions have an excessive cost, which in some cases is higher than that of the operating system. Most vendors have a software tool or kit to integrate a specific application or database into a cluster and make it highly available. This tool incurs additional costs and increases the cost of the cluster implementation (Oracle, 2020).

However, the work and research carried out in recent years with computer clusters has the objective of developing solutions for the management of graphics, creating CPU and GPU clusters, (Al Badawi et al., 2021; Qu et al., 2023; Mai et al., 2022) and implementing hybrid supercomputers used in the field of big data to manage and analyze large amounts of data for scientific applications (Mahmud et al., 2023). Other developments have created techniques to improve the data access on the shared disk arrays of a cluster (Gu et al., 2024). Most recent studies that use a cluster do so using a vendor solution to have highly available applications, such as web servers, where pages are divided into subsets dispersed across the cluster nodes (Jiménez et al., 2021; Purohit et al., 2023), data mining (Zhang et al., 2022), machine learning (Kauffmann et al., 2024), Apache Hadoop Namenode Failover (Serek et al., 2023), and databases aimed at information management in the cloud (Malhotra et al., 2023).

The cost of commercially available equivalent solutions is 10 times more than the one presented in this work. No work has been done on presented here aimed at small- and medium-sized organizations, whose contributions are as follows: maintaining the continuity of the services provided through the database, minimizing the risks derived from the failure of a server, guaranteeing access to database content, and providing a short recovery time. The programming performed in this study allows the mission-critical application, composed of a database, to be started on any of the cluster nodes, which is useful when it is necessary to release one of the nodes. This makes it possible to perform planned preventive or corrective hardware or software maintenance activities such as updating the operating system, installing patches, changing network ports, changing controller cards, firmware updates, or simply re-initializing the computer. The management of the cluster, nodes, and packages can easily be performed through the command line.

Management of the cluster, nodes and packages can be easily done via the command line. The commands are short, and the syntax is simple, allowing you to perform operation and maintenance tasks on the hardware and software or react quickly to any failure of the cluster components. This also has the advantage of being able to use a text terminal without the need for a graphical one, which also reduces the cost of the solution.

2. Materials and methods

The methodology used to develop the work consisted of dividing it into five modules: the cluster manager, package manager, monitoring module, *HBA* port manager, and network port manager. Figure 2 shows the architecture of the cluster, which is composed of two servers connected to a shared disk array where a mission-critical application is installed. This figure shows the main contribution of this work: eliminating, through redundancy in hardware components, single points of failure that can cause no access to the application: Oracle instance and database.

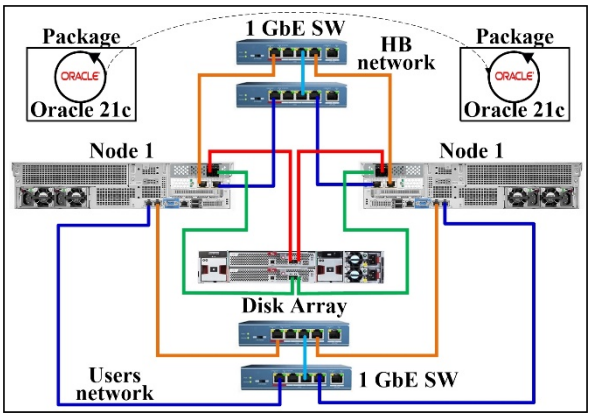

Figure 2. Cluster components diagram.

The program to implement the cluster is composed of a main program and the five modules indicated above. Each module is a child process that is created and invoked in the background of the main program, with the exception of the monitoring module. Programming was performed using shell scripts, and communication between the main program and modules was performed through sockets.

The cluster nodes were HPE Proliant DL180 Gen 10 servers, on which a virtual machine was created using VMware vSphere 8.0. VMware was used to install an ESXi hypervisor that hosted the virtual machine on each server. The SUSE Linux Enterprise Server 15 SP5 operating system was installed on the virtual machine. The virtual machines in the cluster were monitored and managed using the VMware vCenter Server. Virtual machines were used because they are a secure and efficient hybrid cloud platform that many small and medium-sized businesses currently have. Oracle 21c and the database were installed on an HPE D3610 12Gb SAS shared disk array. This device has 12 bays, of which four were used to install 6.4 TB Solid State drives configured in RAID 10, and has a usable disk capacity of 12.8 TB. The array has two controller cards or an *HBA* with two SAS-3 ports each and achieves a bandwidth of 12Gb/s. The ports of one controller were connected to one of the cluster nodes, and the ports of the other to the second node to implement high availability in access to the shared disk array and start the cluster application on any of the nodes. The nodes have a two-port 12Gb/s SAS PCI 3.0 HBA controller to connect to the SAS-3 controller of the disk array. The connection between the nodes and the array is made using the Serial Attached *SCSI* (*SAS*) communication standard, the successor to the parallel Small Computer System Interface (*SCSI*) standard. *SAS* uses *SCSI* commands to transfer information to serial devices.

The cluster is connected to the network of end users or clients of the application and to the private heartbeat Local Area Network (*LAN*). The servers have two network cards with two ports of 1 GbE each. Each card is connected to one of the data networks of the cluster. Through the heartbeat signal, the nodes periodically exchange information regarding their functional status. Therefore, this signal is critical for the operation of the cluster. To avoid delays in this signal, a private *LAN* was used for the heartbeat communication.

2.1. Cluster manager

A cluster administrator must create two configuration files. The first is the cluster configuration file, which sets the following parameters: the cluster name, node name, and heartbeat signal timer value. The second is the package configuration file, in which the following parameters are established: name, primary node, alternate node, path of the package's start and stop scripts, IP address, and the application processes to be monitored. The states of the cluster, nodes, and packages are stored in a cluster-status database. The status of each of these elements can be "*running*" or "*halted'*.

The program allows the cluster administrator user to execute one of the following commands in the node console: *cluster start/stop*, *cluster status*, *node start/stop name*, or *package start/stop*. Both commands invoke execution of the main program. This creates four child processes that implement the five modules of the cluster, except for the monitoring module, which is created when the package begins, as explained later.

The cluster start/stop command is used to start or stop the cluster. If the argument *start*, the main program creates and starts the child process of the cluster manager, which is responsible for the communication between the nodes through the heartbeat signal, startup of the nodes, and startup of the package in the primary cluster node.

To manage the heartbeat signal, the cluster manager process creates and starts a similar process on the other node such that both communicate periodically through a private network. The first process requests the second to join the cluster and waits for the response to be no longer than the timer value indicated in the cluster configuration file. If the other node accepts the request, the cluster manager requests

the package manager to start the latter at the primary node and the cluster is formed. Next, the manager assigns the cluster, nodes, and package the "*running*" state, updates the cluster status database, stored on each node, and displays its contents in the console of the node where the command was executed. Finally, both processes enter a continuous loop where they communicate periodically via the heartbeat signal and wait for a command to be sent by the main program. Once the cluster and package have been started, the main program returns the prompt to the cluster administrator user on the command line and continues running in the background waiting for another command. The flowchart in Figure 3 shows the actions performed by the cluster manager at cluster startup.

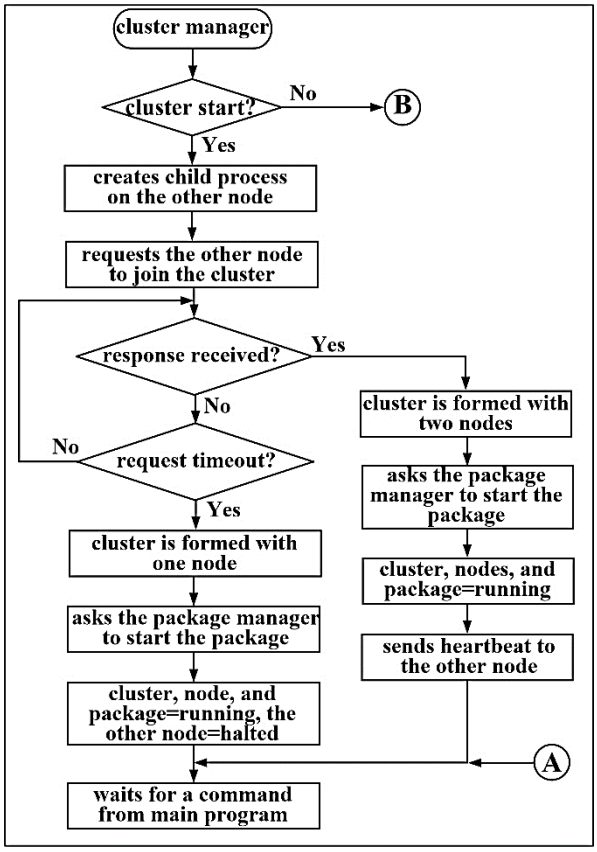

Figure 3. Cluster manager flowchart, cluster start.

If the cluster manager cannot start the process at the second node, the cluster will not be automatically formed. This may be because there is no communication through the heartbeat network, the other node is turned off, or it is not responding owing to some abnormal circumstances. In this scenario, the cluster manager process requests that the package manager start the package on the functional node and the cluster is formed with only one of the nodes. Next, the cluster manager assigns the cluster, node, and package the "*running*" state, assigns the unresponsive node the "*halted*" state, updates the cluster status database, and displays its contents in the console of the node where the program is executed. Finally, the process enters a continuous loop where it waits for a command from the main program. Subsequently, the main program returns the prompt to the user on the command line and continues to run in the background, waiting for another command from the administrator. This requires the cluster user administrator to review the unresponsive nodes and determine the problem.

When the cluster is formed with two nodes, and if at any time the heartbeat signal timer expires in one of the nodes, the cluster manager process does not receive a response, the latter assumes that the other node is not functional and has failed. This causes the cluster manager to check if the unresponsive node is running the package. If so, the cluster manager asks the package manager to start it on the functional node, which is known as failover, and provides high availability in the application, moving the package from one node to the other. The failed node is assigned the "*halted*" state, the package is assigned the "*running*" state, the cluster status database is updated, the sending of the heartbeat signal is stopped, and the cluster manager process enters a cycle in which it waits for a command sent by the main program.

However, if the requested command is cluster stop, the main program requests that the cluster manager process stop the package. To accomplish this, the manager determines the node where the package is running and requests that the package manager stop it. Next, it stops sending the heartbeat signal between the cluster nodes, assigns the "halted" state to the cluster, nodes, and package, terminates the process, and returns the control to the main program. The flowchart in Figure 4 shows the actions taken by the cluster manager to stop a cluster.

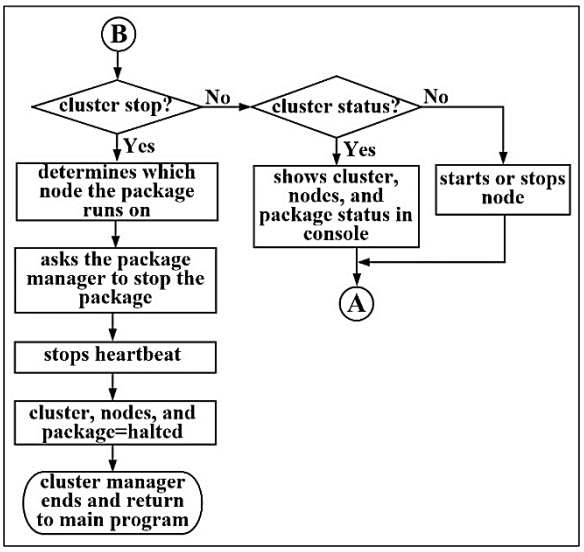

Figure 4. Cluster manager flowchart, cluster stop.

If the command executed by the user is cluster status, the main program requests the cluster manager process to display the status of the cluster, nodes, and packages in the node console. The process displays the content of the cluster status database in the node console and returns to wait for a command to be sent by the main program.

If the requested command is node start name, where the third argument is the name of the node, the main program requests that the cluster manager integrate the node into a cluster. The tasks performed by this manager create a similar process in the node to be integrated for the communication of the heartbeat signal, request that the created process join the cluster, and wait for the response to this process.

If the node to be integrated accepts the request, the cluster is created, the manager assigns the integrated node the "*running*" state, updates the cluster status database on each node, and displays its contents in the console of the node where the command is executed. The two processes then enter a loop in which they periodically communicate their functional status via the heartbeat signal and wait for a command sent by the main program. After integrating the node into the cluster, the main program returns the prompt to the user on the command line and continues to run in the background while waiting for another command.

If the requested command is a node stop name, the main program requests the cluster manager to stop or remove the node from the cluster. This action is useful when it is necessary to perform maintenance tasks on the node, which causes the cluster manager to verify whether the node to be stopped is running the package. If so, the manager asks the package manager to stop doing so. Set the stopped node and the package "halted" state, update the cluster status database, display its contents in the console of the node where the command is executed, terminate the cluster manager process that runs on the stopped node, and send the heartbeat signal. The cluster manager process of the node that remains in the cluster enters a loop in which it waits for a command to be sent by the main program.

2.2. Package manager

The high availability of the mission-critical application was implemented by creating a package that grouped the resources required by the application. These consist of storage drives where the Oracle instance software and database are installed and the IP address is assigned to the package. To allow the application to run on any of the nodes in the cluster, it must not refer to the host names of the nodes or their IP addresses. You must use the host name and IP address assigned to the package in the operating system */etc/hosts* file. In addition, the IP address of the package was replaced by the *listener.ora* and *tnsnames.ora* files of the Oracle instance.

To manage the package, the cluster administrator user can use *package start/stop* commands in the console. When this command is executed, the main program, which runs in the background on the cluster nodes, invokes the package manager process, passing the action indicated in the command as a parameter: start or stop.

When the cluster administrator user executes the package start command, the main program invokes the package manager process, and the latter performs the following tasks: 1) activate the volume groups and mount the file systems used by the instance and the database; 2) set the IP address of the package to the network port of the node where the package is starting; 3) initialize the Oracle instance environment variables set in the *oracle.conf* file, and 4) check that it is running the *LISTENER* process. If so, it stops and then starts because it may be in an inconsistent state and unable to receive database access requests. If not, just start it. 5) start the Oracle instance by executing the *sqlplus startup open* command, 6) start the monitoring module, and 7) terminate the process and return control to the main program.

The Oracle instance environment variables indicate the following information: instance name, HOME directory, kernel parameter file, and Oracle processes to monitor. The Linux and Oracle actions and commands executed at the startup and shutdown of the package are recorded in a cluster log file called *cluster.log*. Figure 5 shows the flowchart used to perform the package manager tasks.

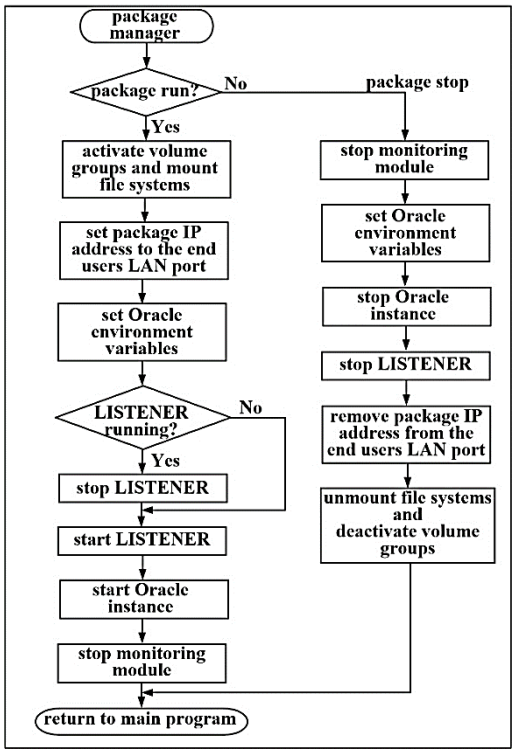

Figure 5. Package manager flowchart.

The IP address assigned to the package is called volatile, virtual, or relocatable because it moves with the package between the cluster nodes. This is so that the Oracle instance does not depend on the physical IP address of the cluster nodes, but on the IP of the package.

When the package stop command is executed, the main program invokes the package manager process at the node where the package is executed. This manager performs the tasks executed at startup in reverse order: 1) finds and stops the process associated with the monitoring module, 2) stops the Oracle instance by executing the *sqlplus shutdown immediate* command, 3) stops the *LISTENER* process, 4) removes the package's IP address of the node's network port, 5) unmounts file systems and disabled volume groups from the package, and 6) terminates the process and returns control to the main program.

2.3. Monitoring module

The function of this module is to periodically monitor the Oracle instance and the status of the associated processes. The monitoring time was configurable and set to 60 s by default. The default monitored processes are as follows: *xe_pmon_SID*, *xe_smon_SID*, *xe_dbw0_SID*, *xe_ckpt_SID*, *xe_lgwr_SID*, and *xe_reco_SID*, where *SID* is the ID of the Oracle instance. Users can modify the times and names of the monitored processes in the *oracle.conf* package configuration file.

This module is invoked by the package manager process during the startup of the latter. The monitoring module verifies the existence of a file called *DEBUG*. The presence of this file suspends monitoring of the Oracle instance and processes and can be created by the user to indicate that the Oracle instance is under maintenance. When performing maintenance tasks, the Oracle instance can be stopped and started by the user, and therefore, should not be monitored so that the package is not stopped, and the resources of the Oracle instance are available in the user's tasks. Upon the completion of instance maintenance, the user must remove the *DEBUG* file. If this file exists, the monitoring module does not perform any action and remains in a cycle, thereby verifying the existence of the file.

If a *DEBUG* file does not exist, the monitoring module enters a cycle in which it reviews the status of the instance and its processes. To perform the first task, the *sqlplus* command is executed: *select status from v\\\$instance*. If the result of this command is an error message containing string *ORA-*, the module assumes that the instance is not running correctly, records the error message in the cluster log file, requests the package manager to stop it, starts it on the other node, finishes the monitoring process, and returns control to the main program. The package terminates because the Oracle instance does not work correctly and access to the database cannot be guaranteed. It starts on the other node to continue providing services to database clients and has a high availability of mission-critical applications.

If the Oracle instance works correctly, the monitoring module periodically accesses the process dispatcher of the operating system to determine the status of the processes of the instance. If any of these processes are not running or fail, the process in this module performs the same tasks that it executes when the instance is not running correctly, as indicated in the previous paragraph. This allows the user to provide a high availability of the Oracle instance while reviewing and resolving the issue at the node where the instance or instance processes fail. Figure 6 shows the flowchart used in programming to perform the tasks of the monitoring module.

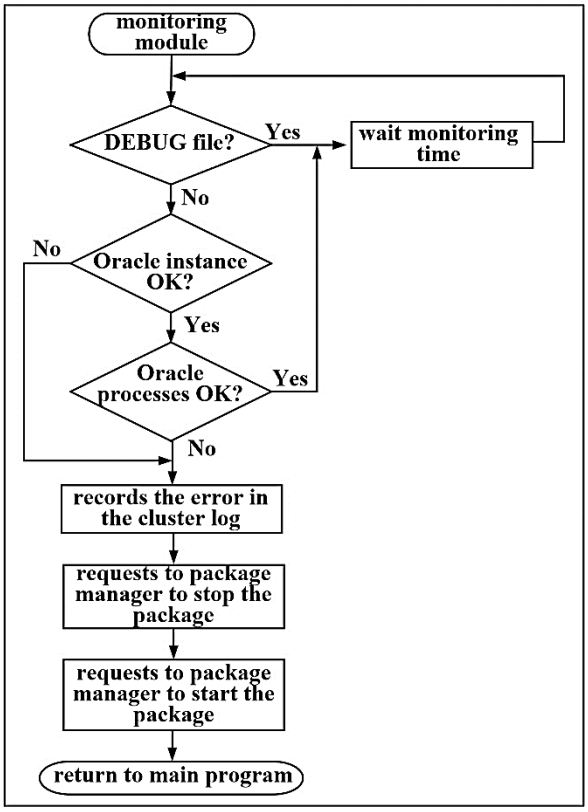

Figure 6. Monitoring module flowchart.

The Oracle software and database were installed on four disks of the cluster's shared array configured in RAID 10 to protect the information physically-

2.4. HBA port manager

The two ports of the *HBA* controller on the nodes were configured to operate in round-robin mode and balance access to the disk array. When one port fails, data access is performed through another port and by backing each other. The process that implements the *HBA* port management module is invoked

at the cluster startup and is in a continuous cycle of monitoring the status of each port every second through the execution of the multipath command. When a port fails, the process displays an error message on the console of the node where the port fails. When both ports fail, it shows the error message in the node console. If the package is running on that node, it requests the package manager to stop it and start it on another node, fail over, because the node where the ports fail will not be able to access the application. When recovering one or both *HBA* ports after a failure, the process displays a message on the node console indicating the previous event. The *HBA* port manager process is terminated by the cluster manager when a node or cluster stops.

2.5. LAN port manager

Each cluster node has two network cards with two 1 GbE ports. The four ports are connected to different *LAN* switches, and the switches are connected to each other through another port. Four switches were used: two for connection to the data network of the clients and end users of the application, and two to form the private network of the heartbeat signal. Thus, there is high availability of network ports and switches. The two network ports of each node (*eth0* and *eth1*) connected to the clients' network were configured to form a group or bond in round-robin mode (*bond0*); when the two ports were working, they balanced network access on each server, and when one failed, access to the network was carried out by the one who continued to work, supporting each other. Figure 7 shows the */etc/network/interfaces/bond0* file, used for the configuration of the *bond0* group, which includes the network ports *eth0* and *eth1*.

```
iface bond0 inet static
    address 192.168.0.100
    netmask 255.255.255.0
    network 192.168.0.0
    gateway 192.168.0.1
    slaves eth0 eth1
    bond_mode 0
    bond_miimon 100
    bond_downdelay 200
    bond_updelay 200
```

Figure 7. bond0 configuration file.

This *bond0* is assigned to the IP address of the server and that of the package when the latter runs on the server. The two network ports used for the connection to the private network were configured similarly to the previous ports to form a second group. The process that implements this module is in a cycle that monitors the status of the network groups and ports every second through the execution of the if link show command. When a network port fails, the process displays an error message on the console of the node where the port fails. When both ports fail, it displays the error message in the console of the node. If the package is running on that node, it requests the package manager to stop it and start it on another node, fail over, because users will not be able to access the application on the node where the ports fail. When a network port or group of ports recovers after a failure, the process displays a message in the node console indicating the previous event. The network port manager process is terminated by the cluster manager when a node or cluster stops.

3. Results and discussion

Before starting the cluster tests, a database with a table was created, using the script indicated in Figure 8. Both the Oracle software and the database are contained in a 100 GB file system, which is shown in the output of the Linux *df* command in Figure 9.

```
#!/usr/bin/sh

sqlplus / as sysdba
CREATE TABLE TEST (
   ID integer PRIMARY KEY,
   NAME varchar(50),
   ADDRESS varchar(50),
   SALARY integer
```

Figure 8. Script used to create the table in the test database.

```
~]$ df
Filesystem                1K-blocks      Used Available Use% Mounted on
/dev/mapper/vg_kvm-lv_root  18618236   4357360  13315112  25% /
tmpfs                         380376       288    380088   1% /dev/shm
/dev/vda1                     495844     77029    393215  17% /boot
dev/vda2/lvoracle
                            10097152  686161 1322873   34% /oracle
```

Figure 9. File system created for Oracle package database and table.

Four sets of tests were performed. The first group aimed to determine the package movement time from one node to another in a controlled environment. Each tests consisted of stopping the package on one node and starting it on the other without any open Oracle user sessions or database transactions. During testing, the package was stopped and started correctly within 20 s. Subsequently, the same test was performed several times, adding a 100 GB file system to the package in each test until 20 file systems were reached. As expected, in each test, the package startup time increased because before mounting each file system, the *mount* command checks the consistency of the file system. Figure 10 shows the *oracle.conf* package configuration file used in these tests.

```
#!/usr/bin/sh
########################################
# Oracle configuration file oracle.conf
########################################
INSTANCE_TYPE=database
ORACLE_HOME=/oracle/PRO/112_64
ORACLE_ADMIN=orapro
SID=PRO
START_MODE=open

set -A LISTENER_NAME
LISTENER_NAME[0]=
set -A MONITOR_PROCESSES
MONITOR_PROCESSES[0]=xe_pmon_${SID}
MONITOR_PROCESSES[1]=xe_smon_${SID}
MONITOR_PROCESSES[2]=xe_dbw0_${SID}
MONITOR_PROCESSES[3]=xe_ckpt_${SID}
MONITOR_PROCESSES[4]=oxe_lgwr_${SID}
MONITOR_PROCESSES[5]=xe_reco_${SID}
```

Figure 10. Oracle configuration file.

To start the package, the package start command was executed, which invokes the script shown in Figure 11 to start the Oracle instance.

```
#!/usr/bin/sh
######################
# Oracle Start Script
######################

${ORACLE_HOME}/bin/lsnrctl start
sqlplus / as sysdba
startup
```

Figure 11. Oracle start script.

To stop the package, the package stop command was executed, which invokes the script shown in Figure 12 to stop the Oracle instance.

```
#!/usr/bin/sh
######################
# Oracle Stop Script
######################

sqlplus / as sysdba
shutdown immediate

${ORACLE_HOME}/bin/lsnrctl stop
```

Figure 12. Oracle stop script.

After starting the package, the Linux df command was executed to verify that the file system has been mounted by the package, as shown in Figure 9. Additionally, the Oracle instance was accessed to connect to the database and check the status (*OPEN*) of the instance, as shown by the script used and the Oracle output in Figure 13.

```
#!/usr/bin/sh

sqlplus / as sysdba
select PRO, STATUS from
v$instance;


[test@ora ~]$

      PRO             STATUS
 ---------------- ------------
  ora21c             OPEN
```

Figure 13. Oracle instance check script.

To complete the test, the database was accessed to insert a record into the table created to verify the Oracle functionality using the script shown in Figure 14.

```
#!/usr/bin/sh

sqlplus / as sysdba
INSERT INTO TEST (ID, NAME, ADDRESS, SALARY)
VALUES (5,'Name Test5','Mexico no. 5',
5000);

-- Show data Table

SELECT * from TEST;
```

Figure 14. Script to insert a record into the database.

The results of these tests are indicated in the graph in Figure 15, which shows that the package startup time increased proportionally in each test. As you can see, as the file system mounted by the package grows, the package startup time increases slightly.

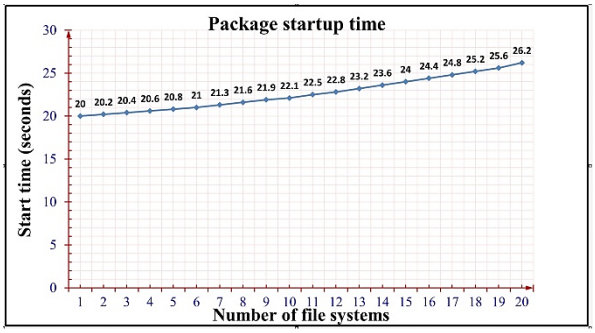

Figure 15. Package startup time.

The second group of tests aimed to determine the time consumed by the startup of the package after the node where it was running suffered contingencies. To perform these tests, 20 Oracle user sessions were initially opened, where the *INSERT*, *DELETE*, *UPDATE*, and *COMMIT SQL* statements were executed to insert, remove, and update the database records. While users were performing the above actions, power was

disconnected from the node where the package was running, and it automatically started or moved to another node, which is known as failover. The startup time of the package with only a 100 GB file system was 27 s, which is longer than the time of the first test of the previous group. Before opening the database, the Oracle manager had to complete transactions that were pending or rollback at the node where the contingency occurred. Subsequently, the test was repeated with 30, 40, and 50 sessions of users who accessed the database, and it was observed that the package startup time increased proportionally. To complete this group of tests, the node was connected to electrical power, and once it started, the node was integrated using the node start command. The package was then stopped and started on the recovered node to verify proper access to the database and perform a process known as failback.

The package shutdown was 10 s, which is less than the startup time, since during the shutdown the system checks the consistency of the file systems. The cluster shutdown was also tested in this part, which took 15 s.

The purpose of the third group of tests is to verify the operation of the *HBA* port manager. To perform these tests, a program was created to execute a cycle in which a database table was read and written. Subsequently, one of the ports was disconnected from the node's *HBA* port controller, and it was verified that access to the table was not interrupted. Subsequently, both ports were disconnected from the controller and it was confirmed that the package was automatically moved to another node. Next, the ports were connected and verified from the node's operating system, they were recovered, and the package did not return to the node where the ports were disconnected. Previous tests have been performed on two nodes of cluster.

The purpose of the last group of tests was to check the operation of the *LAN* port manager. To perform these tests, the program from the previous group of tests was executed, and one of the switches of both the end-user *LAN* and the heartbeat *LAN* was turned off. It was verified that access to the table was not interrupted.

This test simulated the failure of the *LAN* port *eth0*, however, users continued to access the Oracle instance through the *bond0* group, which only has the *eth1* port active (*up*), as shown in the cat */proc/net/bonding/bond0* command output in Figure 16.

Subsequently, the two switches are connected, and it is verified that the failed ports are recovered from the operating system of the node. The last two sets of tests demonstrated the functionality of redundancy in the *HBA* controller ports and network ports.

Other cluster architectures have only two *LAN* ports on the nodes and two switches. These are simpler and less expensive than the cluster in this work, since one of the ports is connected to a *LAN* switch and the second to the other switch. There is redundancy in the *LAN* ports. They use only one *LAN* to transmit both the cluster Heartbeat signal and the user data. However, there is a risk when there is a high load on the user data that causes delay or loss of the Heartbeat signal and therefore the packet moves from one node to the other or the cluster goes down due to the loss of the heartbeat signal. Considering the above, this work improves the existing clusters by incorporating two *LAN* segments, one to transmit the user data and the second to transmit the Heartbeat signal. With this scheme there is redundancy in ports and *LAN* segments and higher performance in the user data segment. In this respect, the cost of the cluster is not significantly higher than existing ones, since the cost of the additional *LAN* ports and switches is not significant compared to that of the nodes and the disk array. Additionally, the cost of the *LAN* architecture proposed in this work is lower compared to the losses that a company may have when it does not have access to its applications because the cluster is down.

```
~]$ cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1

Bonding Mode: fault-tolerance (balance-rr)
Primary Slave: eth0 (primary reselect
always)
Currently Active Slave: eth1
MII Status: up
MII Polling Interval (ms): 100
Up Delay  (ms): 200
Down Delay  (ms): 200

Slave Interface: eth0
MII Status: down
Speed: Unknown
Duplex: Unknown
Link Failure Count: 2
Slave queue ID: 0

Slave Interface: eth1
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 1
Slave queue ID: 0
```

Figure 16. Status of bond0 group after eth0 LAN port failure.

4. Conclusions

In this study, a highly available application was obtained by implementing a computer cluster composed of two virtual machines with a Linux operating system. The application, which is composed of an Oracle database, can be run on any of the cluster nodes to provide service to users when any of the nodes are undergoing maintenance or have a failure. The cluster is not a fault-tolerant or non-stop system because, if a contingency occurs on the node where the application is executed, it can be started on the other node, which takes 20 s, a time in which the end users do not have access to it. The advantage of a highly available application is that it runs on a

cluster of computers, whose cost is much lower than that of nonstop systems or solutions.

The programming performed in this work allows small- and medium-sized business users to have Linux servers with sufficient redundancy, at least in network ports and disk controller cards, and to have high availability of an Oracle database, ease of use, and implementation without making high investments. Cluster administration was performed using a command line.

The contributions of the presented cluster are the following: 1- it is a high availability solution whose implementation and maintenance cost is much lower than non-stop solutions and lower than commercially available cluster architectures, 2- in case of failure of a network port, an *HBA*, hard disk drives or one of the servers, companies can continue normal operation, 3- when it is necessary to perform some preventive or corrective maintenance activity on a server, the application that was running on it can be moved to the other node and continue offering the service to users, 4- each node of the cluster was implemented in a virtual machine of a physical server, which allows creating other virtual machines on the server to install other operating systems and applications, 5- the operation scheme is active-active, which allows the nodes to back each other up and in normal operation balance the load by integrating other packages and applications, most commercial solutions use an active-standby scheme, and 6- a low-cost license of the Linux operating system was used, the most common platform today for mid-level and enterprise applications, which contributes to reducing the cost of the cluster and facilitating upgrades and maintenance, since there are a variety of hardware and software tools and applications for this operating platform. Other solutions use a proprietary operating system. The cost of the cluster is one-tenth that of an equivalent commercially existing one. The most significant contribution is that the user can compare the cost of the cluster against the costs and losses that must be incurred in the event of a failure of the server where their mission-critical application is running.

Future work will involve two tasks. The first is to integrate a Graphical User Interface (*GUI*) with the cluster manager. This is to be able to view the status of the nodes, applications, network ports, and *HBA* ports, and react in a timely manner to the failure of these components. The second action is that the package manager can start, stop, and monitor more than one application, which will bring the benefit that both cluster nodes can run applications simultaneously and have an active-active scheme on the nodes. Finally, the programming performed in this study has the advantage that the cluster nodes are virtual machines, which allows other virtual machines with different operating systems and applications to be installed on each physical server, maximizing the use of physical infrastructure and investment.

Conflict of interest

The authors do not have any type of conflict of interest to declare.

Acknowledgements

The authors wish to thank the Electronics Department of Universidad Autónoma Metropolitana-Azcapotzalco for supporting this study.

Funding

This work was supported by UAM.

References

Al Badawi, A., Veeravalli, B., Lin, J., Xiao, N., Kazuaki, M., & Khin Mi Mi, A. (2021). Multi-GPU Design and Performance Evaluation of Homomorphic Encryption on GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 32(2), 379-391. https://doi.org/10.1109/TPDS.2020.3021238

Aziz, N., & Jambari, D. (2019). Information Management Procedures for Business Continuity Plan Maintenance. *International Conference on Electrical Engineering and Informatics*, 489-495. https://doi.org/10.1109/ICEEI47359.2019.8988804

Faramondi, L., Guarino, S., Oliva, G., & Setola, R. (2024). A Recovery Model for Faulty Networked System. *IEEE Systems Journal*, 18(1), 146-149. https://doi.org/10.1109/JSYST.2023.3315782

Ferdousi, S., Tornatore, M., Dikbiyik, F., Martel, C., Xu, S., Hirota, Y., & Awaji, Y. (2020). Joint Progressive Network and Datacenter Recovery After Large-Scale Disasters. *IEEE Transactions on Network and Service Management, 17*(3), 1501-1514. https://doi.org/10.1109/TNSM.2020.2983822

Gu, Y., Liu, L., Wu, C., Li, J., & Guo, M. (2024). Ada-WL: An Adaptive Wear-Leveling Aware Data Migration Approach for Flexible SSD Array Scaling in Clusters. *IEEE Transactions on Computers, Early Access Article*. https://doi.org/10.1109/TC.2024.3398493

HPE. (2024). HPE Serviceguard for Linux (SGLX). Hewlett Packard Enterprise Development LP. https://www.hpe.com/psnow/doc/c04154488

Jiménez, L. R., Solera, M., Toril, M., Gijón, C., & Casas, P. (2021). Content Matters: Clustering Web Pages for QoE Analysis With WebCLUST. *IEEE Access*, 9, 123873-123888. https://doi.org/10.1109/ACCESS.2021.3110370

Kauffmann, J., Esders, M., Ruff, L., Montavon, G., Samek, W., & Müller, K. R. (2024). From Clustering to Cluster Explanations via Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 35(2), 1926-1940. https://doi.org/10.1109/TNNLS.2022.3185901

Liu, Y., Zhou, F., Chen, C., Zhu, Z., Shang, T., & Torres-Moreno, J. M. (2021). Disaster Protection in Inter-DataCenter Networks Leveraging Cooperative Storage. *IEEE Transactions on Network and Service Management, 8*(3), 2598-2611. https://doi.org/10.1109/TNSM.2021.3089049

Mahmud, M. S., Huang, J. Z., Ruby, R., Ngueilbaye, A., & Wu, K. (2023). Approximate Clustering Ensemble Method for Big Data. *IEEE Transactions on Big Data*, 9(3), 1142-1155. https://doi.org/10.1109/TBDATA.2023.3255003

Mai, S. T., Jacobsen, J., Amer-Yahia, S., Spence, I., Tran, N.-P., Assent, I., & Viet Hung Nguyen, Q. (2022). Incremental Density-Based Clustering on Multicore Processors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *4*(3), 1338-1356. https://doi.org/10.1109/TPAMI.2020.3023125

Malhotra, A., Elsayed, A., Torres, R., & Venkatraman, S. (2023). Evaluate Solutions for Achieving High Availability or Near Zero Downtime for Cloud Native Enterprise Applications. *IEEE Access*, 11, 85384-85394. https://doi.org/10.1109/ACCESS.2023.3303430

Mudassir, G., Howard, E. E., Pasquini, L., Arbib, C., Clementini, E., Di Marco, A., & Stilo, G. (2021). Toward Effective Response to Natural Disasters: A Data Science Approach. IEEE Access, 9, 167827-167844. https://doi.org/10.1109/ACCESS.2021.3135054

NIST. (2024). Special Publication 800-34 Rev. 1: Contingency Planning Guide for Federal Information Systems. CreateSpace Independent Publis.https://csrc.nist.gov/pubs/sp/800/34/r1/upd1/final

Oracle. (2020). Protect your VMware SDDC in the cloud against disasters. Oracle Corporation. https://docs.oracle.com/en/solutions/implement-dr-for-ocvs/#GUID-2C632A62-3492-41EF-8778-0D548FF580CC

Petrenko, S. (2021). Developing an Enterprise Continuity Program. River Publishers.

Purohit, L., Rathore, S. S., & Kumar, S. (2023). A QoS-Aware Clustering Based Multi-Layer Model for Web Service Selection. *IEEE Transactions on Services Computing*, *16*(5), 3141-3154. https://doi.org/10.1109/TSC.2023.3264627

Qu, P., Lin, H., Pang, M., Liu, X., Zheng, W., & Zhang, Y. (2023). ENLARGE: An Efficient SNN Simulation Framework on GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems*, *34*(9), 2529-2540. https://doi.org/10.1109/TPDS.2023.3291825

Red Hat. (2024). High Availability Add-On. Red Hat, Inc. https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/7/html-single/high_availability_add-on_overview/index

Saxena, D., Gupta, I., Singh, A. K., & Lee, C. N. (2022). A Fault Tolerant Elastic Resource Management Framework Toward High Availability of Cloud Services. *IEEE Transactions on Network and Service Management*, *19*(3), 3048-3061. https://doi.org/10.1109/TNSM.2022.3170379

Schwartz, K. D., & Goodwin, P. (2022). Practices to Improve Disaster Recovery Testing. IDC PeerScape,.

Serek, A., Orynbekova, K., Talasbek, A., Kariboz, D., Saimassay, G., & Bogdanchikov, A. (2023). Recommendation System for Human Resource Management by the Use of Apache Spark Cluster. 17th International Conference on Electronics Computer and Computation, 1-4. https://doi.org/10.1109/ICECCO58239.2023.10147129

Smith, A. (2020). Cloud-Based Disaster Recovery Services Help Organizations Achieve Business Resilience. IDC Research, Inc.

Strawser, B. (2019). ISO 27031: Looking at ISO's Disaster Recovery Standard. BRYGHTPATH LLC. https://bryghtpath.com/iso-27031-looking-at-isos-disaster-recovery-standard/

Tian, J., Ma, P., Wang, C., & Wang, Z. (2023). Research on Development of Data Disaster Recovery System. *22nd International Conference on Trust, Security and Privacy in Computing and Communications*, 2210-2215. https://doi.org/10.1109/TrustCom60117.2023.00310

Tomás, L., Kokkinos, P., Anagnostopoulos, V., Feder, O., Kyriazis, D., Meth, K., . . . Varvarigou, T. (2020). Disaster Recovery Layer for Distributed OpenStack Deployments. *IEEE Transactions on Cloud Computing*, *8*(1), 112-123. https://doi.org/10.1109/TCC.2017.2745560

VMWare. (2024). VMware Site Recovery Manager. Broadcom Inc. https://docs.vmware.com/en/Site-Recovery-Manager/index.html

Wan, Y., & Zhu, Q. (2020). The IT Challenges in Disaster Relief: What We Learned From Hurricane Harvey. *IT Professional*, *2*(6), 52-58. https://doi.org/10.1109/MITP.2020.3005675

Zhang, Q., Yang, L. T., Chen, Z., & Li, P. (2022). PPHOPCM: Privacy-Preserving High-Order Possibilistic c-Means Algorithm for Big Data Clustering with Cloud Computing. *IEEE Transactions on Big Data*, *8*(1), 25-34. https://doi.org/10.1109/TBDATA.2017.2701816