RESEARCH ARTICLE

An Optimal Checkpointing with Message Logging Protocol for Fault Tolerance of Distributed Applications in the Cloud Data Center

Priti Kumari[1, *], Vandana Dubey[1], Vinita[2]

1 Department of Computer Science & Engineering, Ashoka Institute of Technology and Management, Varanasi, India

2 Department of Computer Science & Engineering, ABES Engineering College, Ghaziabad, India

Priti19sep@gmail.com, vandanashuklaec05@gmail.com, vinita89.cse@gmail.com

Abstract: ***Background:*** Services running on low-cost hardware generally enabled by cloud data centers (CDC). Scaling the hardware horizontally is made simple by the addition of more resources. The CDC uses commodity hardware, which results in a high failure rate for physical servers (PSs). As a result of this failure, virtual servers (VSs) were provisioned on the failed PS. Hence the fault tolerance is a major challenge for cloud service providers.

Objective: To handle failures of commodity hardware in CDC, there is need of fault tolerance method. The motive of this work is to develop an optimal and efficient failure recovery methodology based on checkpointing for fault tolerance of cloud-based services.

Methods: the proposed approach is implemented in two steps. In the first step, we build a virtual backbone across the CDC network architecture; the research suggests a novel Connected Dominating Set (CDS) based method. The PSs serve as the vertex of the graph used in the CDS creation approach to represent the network topology. In order to obtain a CDS or optimal number of PS for the topology graph, it then suggests a set of criteria based on the rate of CPU heating, storage capacity, and vertex degree of PSs. Moreover, This base is then used to create a fault tolerant system based on checkpointing and rollback recovery (CRR) in order to increase reliability.
In the second step, A CDSCKP method is proposed. In which we have implemented uncoordinated checkpointing with message logging while taking into account distributed applications. Checkpoint snapshots of tasks or VSs are placed on the CDS vertices.

Results: The suggested scheme's effectiveness is assessed using parameters such as recoverability, bandwidth and power consumption and rollback and recovery time. The CDSCKP is compared with a Random checkpointing placement protocol (RCKP), Wu-Li checkpointing placement protocol (WLCKP). The simulation results show that the CDSCKP offers greater recoverability, uses less bandwidth and power, and has little rollback and recovery overhead.

Conclusion: In order to increase the dependability of VS-based services, this study suggests a CDS-based scheme for building a virtual backbone over the DCN topology. This scheme is then used to produce a CRR-based fault tolerance scheme called CDSCKP.

Keywords: Cloud computing; cloud data center; physical server; virtual server; reliability; fault tolerance; connected dominating set; distributed application; checkpointing and rollback recovery; Uncoordinated checkpointing

1. INTRODUCTION

Cloud computing (CC) is increasingly being used as a platform for executing large-scale workloads such as scientific workflow execution, big data processing, analytics applications, and so on. This form of computing provides so many advantages such as on-demand scalability, resource provisioning elasticity, and massively parallel processing design. CC provides different kinds of services to the end users i.e., Software as a Service (SaaS) provides web-hosted applications to users such as email, security, social media applications, and so on; Platform as a Service (PaaS) provides computing platforms to end-users such as operating systems, compilers, and so on; and Infrastructure as a Service (IaaS) delivers computing infrastructure to users over the internet such as virtual servers, storage, and networking. [1][2].

The proposed work primarily focuses on the IaaS, which provides cloud customers with access to a data centre's computational capacity. Virtualization technology is used in IaaS to provide resources to users from faraway IaaS datacentres [3]. Thousands of real machines are housed in a data centre, which can host VSs for users dependent on their needs. The widespread use of the CDC for hosting corporate or enterprise applications, on the other hand, may result in service reliability hitches. Because of their distributed structure, vast scale of operation and variety of commodity technology, a variety of problems emerge in cloud systems, resulting in failures, outages, and decreased performance. Some of the cloud outages are discussed in Table 1.
Fault tolerance (FT) strategies can be used to handle failures in the cloud environment. Even if some system components aren't working properly, FT allows the system to serve user requests [4] [5]. CRR and replication are the most often utilized FT techniques. Replication based on the presence of redundant resources and makes multiple copies of a task or resource [6]. If a task or resource fails, its copy can be employed to keep operations running. CRR, on the other hand, is a technique for regularly saving the state of systems during normal operation. The system is resumed from a previously saved error-free state in the event of a failure [7][8].

Table 1. Cloud Providers Outages in 2022

Cloud Service Provider (CSP)	**Outage**	**Explanation**	**Year**
Google	cloud	A modification was made to the Traffic Director code that manages the configuration.	March-2022 (2 hours)
Microsoft	Mail Related	Afterward a day, in which Microsoft added new security capabilities to secure office 365 users, the failure happens.	2022
Cloudflare	internet	Created significant downtime on a significant portion of the Internet, purportedly affecting well-known websites including Discord, Shopify, Grindr, Fitbit, and Peloton.	June - 2022
Microsoft Azure	power	A sudden power fluctuation in one of our data centres, inside one of our Availability Zones, in the East US 2 regions.	June-2022
Facebook	Workplace collaboration tool	A server problem led to the outage that hit Facebook and Instagram.	2022
Apple	Cloud	Some customers are unable to access their iCloud accounts.	January - 2022
IBM	Cloud	Users of IBM Cloud services in the Dallas region were impacted.	January – 2022 (Less than an hour)
Salesforce	Slack	Some users are having issues loading Slack.	2022
Mimecast	power	Due to a "severe power outage," its North American grid experienced issues, which resulted in delays and "degraded service" for consumers.	2022

The current study proposes a CRR FT mechanism for the enhancing the reliability of cloud-based services. The suggested mechanism creates a CDS over the graph that represents the topology of a CDC. The PSs are the graph's vertex, while the links between them are the graph's edges. The CDS building standards were created with factors specific to data center PSs in mind. The CDS is used to build a virtual backbone across the data center network topology and place checkpoint snapshots of tasks or VSs. In case of failure, the last saved checkpoint is used to start the application. Simulation experiments have found keeping checkpoint images of task using CDS is more reliable, costeffective, and energy-efficient as compared to a RCKP and WLCKP.

This paper's main contribution is as follows:

- A CDS-based backbone for DCN topology is proposed.
- To acquire the CDS and to achieve optimal number of PS for the topology graph, developed some rules based on CPU heating rate, storage capacity, and vertex degree of a PSs. PSs with a lower CPU heating rate are thought to be more reliable vertices. The CDS is formed up of vertices with better storage capacity and vertex degrees.
- Moreover, a CRR based FT protocol such as CDSCKP was implemented using the CDS backbone.
- Finally, we have measured the performance of the suggested approach in terms of recoverability, bandwidth usage, power consumptions, and rollback and recovery times.

The remainder of the paper is structured as follows: In Section II, the associated strategies for CRR-based FT in CC are described. The CRR process is explained in Segment III. The anticipated CDS over CDN topology for storing checkpoint snapshot with messages is covered in Section IV. In Section V, it is addressed how well the suggested method works and the outcomes of the simulation. Section VI is where we finally put an end to our presentation.

2. RELATED WORK

In cloud-based services, FT is a major concern. In the presence of some defective system components, FT approaches allow the system to deliver important services with greater performance [1]. System errors may be caused by either software or hardware (H/W) issues, FT techniques are necessary to detect and handle them. Checkpointing, replication, and task migration are the most utilized strategies for fault tolerance of VSs in IaaS cloud computing, according to the literature [4][5].

In [9] a checkpointing-based FT mechanism was used to increase the resilience of server-based cloud applications. A CRR strategy for a CDC was devised by J. Zhao et al. [8]. A peer-to-peer CRR approach is used in their scheme, in which a VS's CP snapshot is kept on a nearby or peer host PS. The suggested solution eliminates the need for dominant memory in situations where network congestion affects whole host PSs and VSs. The authors also showed that using a distributed way to produce speedier checkpoints and recovery is a good idea. A. Zhou et al. [7] have suggested using the EDCKP checkpoint FT system in data centres' fat-tree topologies to tolerate host, PS, and edge switch failure. The checkpoint snapshots of the VS are installed or located on the PS in the same pod in this method. This strategy seeks to ensure CC-based service reliability while also reducing network resource consumption. Chinnathambi, et al. [10] offered a scheduling technique for keeping track of the performance of the virtual node and a useful checkpointing method for accommodating task and job migration with the least amount of overhead. The proposed technique, according to the authors, effectively recognizes VS flaws and frequently migrates the job to a functional VS. Amoon et al. [11] proposed a fault tolerance strategy based on checkpointing with a variable or changeable checkpoint interval. On the other hand, the percentage of servers that cannot accommodate the VS that is used to finish the application determines how long the interval for that application will be. Cheraghlou et al. [12] developed a framework for fault tolerance in a cloud-based environment based on fuzzy logic. A fuzzy system's design is built on three features: policies, fault detection techniques, and fault recovery techniques. An adaptive method was put out by Rezaeipanah, et al. [13] for quickly identifying cloud system faults. In order to achieve this, the authors used a predictive technique to keep an eye on the system before using a fuzzy logic-based algorithm to find the weaknesses. They also employed the checkpointing approach to cut down on processing costs and migration time. In [14], the authors have used fat tree topology based checkpointing approach to execute both bag of task application and distributed application.

Further, many works have emphasized that any FT scheme (or any other algorithm) does not add a significant overhead to the energy requirements of the cloud computing infrastructure [15][16]. Considering the need to lower the energy demands of a CDC, it is desired that any algorithm execution consumes low bandwidth and thus, low energy.

It is assumed that an efficient topology of a data center can contribute to this requirement. Therefore, many topologies, such as a fat tree, RUFT (reduced unidirectional fat-tree) and Z-fat tree have been suggested in the literature [17] [18]. Our work proposes a CDS based topology for a data center that has been designed for the fault-tolerant service execution. CDS have been often used for virtual backbone construction in networks. R. Kanniga Devi et al. [19] presented a theoretical model based on the dominating set. This model illustrated that the graph with the DS may tolerate faults as well as deliver improved load balancing by decreasing network overhead. In [20] [21], the authors have suggested a distributed marking system to find CDS in a completely connected graph.

3. PREFACES

By recovering the computation to the most recent checkpoint, CRR is a well-liked FT technique for minimizing the amount of computation lost when a process fails. The stable storage device, a logical device that can endure the most system faults, where the programs record their checkpoints during the failure-free operation. The CRR and stable storage processes, which combine checkpointing and application execution, can enhance application performance. [7][8][9]. The CRR method is illustrated in Fig. 1.

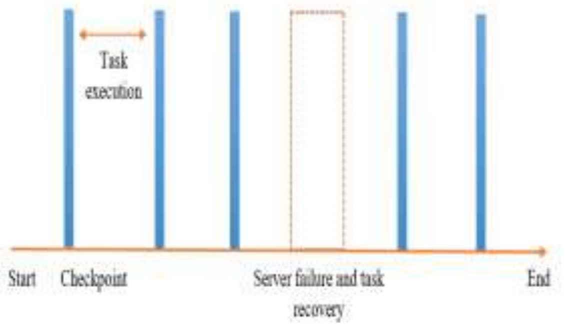


Fig. (1). Task/VS checkpoint and rollback recovery model

The states of each process that makes up the system are combined to form the global system state. The status of a process can be routinely recorded using the CRR technique as a checkpoint that can be restored in the event of a successful recovery failure. However, because message flow between processes may have developed inter-process dependencies, rolling back one process could result in an inconsistent system state. To maintain a consistent global state in this scenario, reverting one process may result in reverting other, fault-free processes as well. Consistent global states are those that represent a state that may have been attained throughout execution [8][22].

The global state G_i is said to exist in a system if there are n processes.

$$\left[\left\{1 \le i \le N, \exists i : G_i = \bigcup_{p=1}^{k} LS_p\right\} \wedge \left\{RM(m) \in LS_p \Rightarrow SM(m) \in LS_q, p \neq q\right\}\right]$$

The send message event is SM, and the local state logged at process p is LSp.
RM represents the receive message event.
A compilation of the local states from each process results in the global state, or Gi.

Orphan messages are those whose sending was not recorded by the sender but whose receipt has been logged by the recipient. There are no orphan messages that are in a stable state.

This work focuses on distributed applications, which are composed of multiple processes or tasks that communicate via messages in order to finish the distributed program [23]. A system must get back to a consistent state after a process failure. The communication between processes must therefore be recorded using the CRR approach. Coordinated checkpointing, where processes synchronize or coordinate their checkpoints in order to save a globally consistent system state, is one technique for achieving global consistency. Because each process continues where it left off from its most recent checkpoint, this strategy facilitates recovery [23][29][30]. As a result, the scope of rollback is constrained and a domino effect, or rollback to the prior system state, is avoided. Conversely, because all processes must take part in each checkpoint action, coordinated protocols are inappropriate for usage in large-scale systems because they significantly increase synchronization costs and delay.

Since each process can choose when to take a checkpoint, asynchronous or uncoordinated checkpointing is preferable to coordinated checkpointing. As a result, there is no need for synchronization, and each process can choose the location and timing of its checkpoints [23] [24]. Uncoordinated checkpointing, on the other hand, could start a chain reaction that spreads throughout the recovery process and costs a large amount of productive labor. Each process must maintain various checkpoints, and if one fails, multiple processes must rollback.

To enable independent checkpointing and process recovery, this study uses CDSCKP, a log-based uncoordinated checkpointing technique. Each process's current state is periodically captured as a checkpoint, and each message it receives afterward (before to the next checkpoint) is recorded in a message log. The checkpoints and the messages that have been recorded are kept in a safe place. When a system process fails, it may still be running and replay the message log from the most recent checkpoint. This makes it possible for the process to reach its pre-failure state on its own, without the aid of other processes. [14][22][23]. The log-based recovery technique is depicted in Fig. 2.

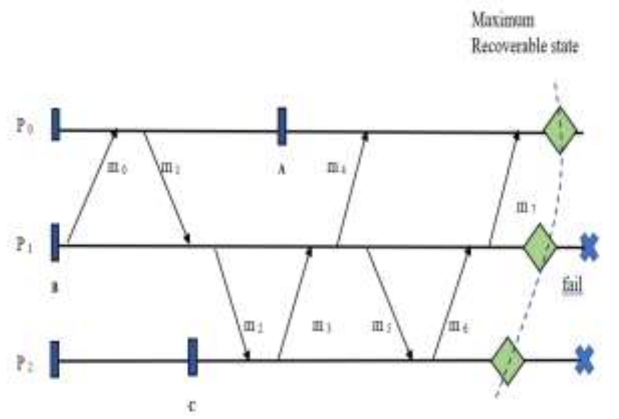


Fig. (2). Log-based Recovery process

The determinants needed to repeat messages m0, m5, m1, m3, m6, and m2, m4 during the failure-free activity are recorded by processes P0, P1, and P2. According to Figure 2, if either process P1 or P2 fails, it starts over from checkpoint B or C and sends the messages using the determinants that were previously registered in the same order. Therefore, once the recovery is complete, the states of both processes will be consistent with the state of process P0, which includes the receipt of message m5 from process P1. By using a pessimistic logging approach, every process' observable state is always recoverable [14].

4. PPROPOSED METHODOLOGY

The study suggests a CDS-based technique for building a virtual backbone over the architecture of the data centre network. The most reliable vertices in the data centre, or those vertices that are least likely to fail, are those that were chosen to be a part of the CDS. The selected vertices are optimal as well. The development of a CRR scheme for large-scale applications such distributed apps running in a CDC is then done using the CDS backbone. A number of jobs are supposed to make up an application, each of which is operating on a VS hosted by a PS in the data centre.

The projected approach consists of two steps. In the first step, we have proposed some CDS rules based on heat rate of CPU, energy consumption and vertex degree to get reliable vertex, which is named as a dominator PS. A vertex is said to be reliable if its CPU heating rate is less and it has a high vertex degree. In the second step, we have proposed a checkpointing algorithm for keeping the checkpoint images of tasks executing on VSs and message log onto the dominator PS in order to enhance the reliability of cloud services. The extended rule for CDS and algorithms for proposed protocol are defined next.

4.1. Building of connected dominating set

If all of the system's vertices are either contained within the set or are neighbours of vertices within the set, then the subset of vertices selected from the system's graph representation is dominant. Additionally, the dominating set, also known as CDS, is linked if the sub graph created by the vertices in this set is. A virtual layer may be overlaid on top of a network's topology using CDS. In order to communicate with their neighbours who are CDS members, the vertices that do not belong to this group send messages to them.

Furthermore, it has been established that it is NP-hard to extract a minimum CDS from a graph [25]. A distributed marking technique was created by Wu and Li [24] to compute CDS in a linked network. Our CDS calculation procedure is different even if we use their marking method. A fully connected unit-disk graph is not necessary for the method to work. The vertex is indicated if at least two of a node's neighbours are not directly connected. We also employ a terminology that is similar to that of the Wu-Li method. It is presented here in a condensed form for the reader's convenience. If a vertex b is marked (m(b)) then it is set to T (True), and F (False) if it is not marked.

Each vertex has a unique id and it is marked *T* or *F*, i.e., CDS member or not, depending on its links with other nodes. When a node b becomes the member of CDS, then *m (b)* is set to 1 (T), otherwise 0 (F). We use the notations Vertex (*b*) and *Vertex[b]* to represent the open and closed neighbour sets of *y* respectively. Thus,

$Vertex(b) = \{b \mid \{a,b\} \in E\}$ denotes the open neighbour set of vertices *b* and
$Vertex[b] = Vertex(a) \cup \{b\}$ denotes the closed neighbour set of vertices *b*

The following localized rules can now be used to minimize the size of CDS [18].

Rule 1: If *m(b) = 0* for a pair of marked nodes *a* and *b* in CDS, then

$$[Vertex[b] \subseteq Vertex[a] \ \ \&\& \ V\_id(b) < V\_id(a)] \quad (1)$$

Rule 2: If a marked vertex b in the CDS has marked neighbours' nodes *a* and *c*, set *m(b) = 0* if

$$\begin{bmatrix} Vertex(b) \subseteq Vertex(a) \cup Vertex(c) \ \&\& V\_id(b) = \\ \min(V\_id(a), V\_id(b), V\_id(c)) \end{bmatrix} \quad (2)$$

where, $V\_id(b)$ denotes id of vertex *b*.

In the CDC, to get the most reliable vertices or PSs having good storage along with the great coverage, we have extended the CDS rules in our proposed work.

4.2. Extending rules for CDS creation

The id-based static selection of dominating nodes may result in poor network performance as nodes in the dominating set use more energy processing messages than nodes outside of it. As a result, we assign two different weights to each PS, which is calculated using the PS's properties, such as CPU heating rate and storage capacity. We are also considering the coverage of PS by computing its neighbours. PSs with a higher weight, i.e., lower CPU temperature, good storage capacity, and a higher vertex degree, are preferred by the CDS vertex selection technique.

Let *h* stand for CPU heating rate, *s* for storage capacity, *d* for a PS's vertex degree, and *nv* for the number of VSs hosted by PSs. Moreover, we have computed two different weights such as *W1* (to get the reliable PS) and *W2* (for saving the CP image along with message) and degree of node is used for defining the coverage of PSs.

- $w_1 = h \times nv$ (3)
- $w_2 = s \times nv$ (4)

We have extended the existing CDS construction rules [13] with a three objective. Firstly, these extended rules aid in reducing the size of CDS. Secondly, these rules identify the more reliable vertices in the network topology of the CDC. Third, is to find the suitable PS machine which is having good storage along with great coverage. For rule formation, CPU heating rate, storage capacity and degree of vertices have been considered as parameters. The proposed rules are defined as follows:

Rule 3: Set m(b) = 0 for any pair of marked vertices a and b if one of the following circumstances applies:

- $V[a] \subseteq V[b]$ && $(W_1(b) > W_1(a))$ (5)
- $V[a] \subseteq V[b]$ && $(W_1(b) = W_1(a))$ && $(V\_id(b) < V\_id(a))$ (6)

Rule 4: If the nodes a, c is both marked neighbours of the vertex b in CDS, then the vertex
b is marked to false, if it satisfies any of the subsequent conditions:

- $V(b) \subseteq \{V(a) \cup V(c)\}$ *and* $V(a) \not\subset \{V(b) \cup V(c)\}$ *and* $V(c) \not\subset \{V(a) \cup V(b)\}$ (7)
- $V(b) \subseteq \{V(a) \cup V(c)\}$ *and* $V(a) \subseteq \{V(b) \cup V(c)\}$ *and* $V(c) \not\subset \{V(a) \cup V(b)\}$ *and* $\left[\left(W_1(b) > W_1(a) \text{ or } \left(W_1(b) = W_1(a) \text{ and } V\_id(b) < V\_id(a)\right)\right)\right]$ (8)
- $V(b) \subseteq \{V(a) \cup V(c)\}$ && $V(a) \subseteq \{V(b) \cup V(c)\}$ && $V(c) \subseteq \{V(a) \cup V(b)\}$ && $\begin{pmatrix} (W_1(b) > W_1(a)) \ \&\& \ (W_1(b) > W_1(c)) \| (W_1(b) = W_1(a) > W_1(c)) \&\& \\ V\_id(b) < V\_id(a) \| \\ (W_1(b) = W_1(a) = W_1(c)) \ \&\& \\ (V\_id(b) = \min(V\_id(a), V\_id(b), V\_id(c))) \end{pmatrix}$ (9)

where, *v* denotes vertex,

Rule 5: For a pair of vertices *a* and *b* in CDS, the node *b* is marked to 0 if

- $V[b] \subseteq V[a] \text{and} \left((W_2(b) < W_2(a)) \text{ or } \left(W_2(b) = W_2(a) \text{ and } \begin{pmatrix} (W_1(b) > W_1(a)) \text{ or} \\ (V\_id(b) < V\_id(a)) \end{pmatrix} \right) \right)$

  (10)

Rule 6: The vertices *a, c* both are marked neighbours of the vetrex *b* in CDS, vertex *b* is marked to 0 if it satisfies any of the subsequent condition:

$$V(b) \subseteq \{V(a) \cup V(c)\} \text{ and } V(a) \not\subset \{V(b) \cup V(c)\} \text{ and } V(c) \not\subset \{V(a) \cup V(b)\} \quad (11)$$

$$V(b) \subseteq \{V(a) \cup V(c)\} \text{ and } V(a) \subseteq \{V(b) \cup V(c)\} \text{ and } V(c) \not\subset \{V(a) \cup V(b)\} \quad (12)$$

and if it holds any of the subsequent conditions:

- $(W_2(b) < W_2(a))$ *or* $(W_2(b) = W_2(a))$ *and* $(V\_id(b) < V\_id(a))$
- $(W_1(b) < W_1(a))$ *or* $((W_1(b) = W_1(a))$ *and* $(V\_id(b) < V\_id(a)))$

$$V(b) \subseteq \{V(a) \cup V(c)\} \text{ and } V(a) \subseteq \{V(b) \cup V(c)\} \text{and } V(c) \subseteq \{V(a) \cup V(b)\} \quad (13)$$

and satisfies any of the subsequent conditions:

- $(W2(b) < W2(a))$ *and* $(W2(b) < W2(c))$
- $(W_2(b) = W_2(a) < W_2(c))$ *and* $(V\_id(b) < V\_id(a))$
- $(W_2(b) = W_2(a) = W_2(c))$ *and* $(V\_id(b) = \min(V\_id(a), V\_id(b), V\_id(c)))$
- $(W_1(b) = W_1(a) > W_1(c))$ && $(W_1(b) > W_1(a)$ *or* $(V\_id(a) < V\_id(b)))$

- $(W_2(b) = W_2(a) = W_2(c))\ \&\&$ $\left[\begin{array}{l}(W_1(b) > W_1(a) \text{ and } W_1(b) > W_1(c)) \text{ or} \\ \left(\begin{array}{l} W_1(b) = W_1(a) > W_1(c) \text{ and} \\ (V\_id(b) < V\_id(a)) \end{array}\right) \\ \text{or} \\ \left(\begin{array}{l} (W_1(b) = W_1(a) = W_1(c)) \text{ and} \\ (V\_id(b) = \min(V\_id(a), V\_id(b), V\_id(c))) \end{array}\right)\end{array}\right]$

Rule 7: PS Node degree-based CDS rule: -Consists of two cases:

Case 1: - For a pair of vertices a and b in CDS, the vertex b is marked to 0, if

$$VD[b] \subseteq VD[a] \qquad (14)$$

and satisfies any of the subsequent conditions:

- $(VD(b) < VD(a))$ or $(W_1(b) = W_1(a))$ and $(VD(b) < VD(a))$
- $(W_2(b) = W_2(a))\ \&\&\ (VD(b) < VD(a))$
- $(VD(b) = VD(a))$ and $(V\_id(b) < V\_id(a))$
- $(VD(a) = VD(b))$ and $(W1(b) > W1(a))$ and $(V\_id(b) < V\_id(a))$
- $(VD(b) = VD(a))$ and $(W_2(b) < W_2(a))$ and $(V\_id(b) < V\_id(a))$
- $(VD(b) = VD(a))$ and $(W_1(b) > W_1(a))$ and $(W_2(b) < W_2(x))$ and $(V\_id(y) < V\_id(x))$

where VD(b) represents the degree of a vertex b in CDC.

Case 2: - The vertices a, c both are marked neighbours of the vertex b in CDS, vertex b is marked to false if it satisfies any of the subsequent conditions:

$$V(b) \subseteq \{V(a) \cup V(c)\} \text{ and } V(a) \not\subset \{V(b) \cup V(c)\} \text{ and } V(c) \not\subset \{V(a) \cup V(b)\} \qquad (15)$$

$$V(b) \subseteq \{V(a) \cup V(c)\} \text{ and } V(a) \subseteq \{V(b) \cup V(c)\} \text{ and } V(c) \not\subset \{V(a) \cup V(b)\} \qquad (16)$$

and if it holds any of the subsequent conditions:

- $(VD(b) < VD(a))$ or $(VD(b) = VD(a))$ and $(V\_id(b) < V\_id(a))$
- $(W_1(b) > W_1(a))$ or $(W_2(b) < W_2(a))$
- $(W1(b) = W1(a))$ and $\left[(VD(b) < VD(a)) \text{or} \left(\begin{array}{l} VD(b) = VD(a) \text{ and} \\ V\_id(b) < V\_id(a) \end{array}\right)\right]$
- $(W_2(b) = W_2(a))$ and $\left[(VD(b) < VD(a)) \text{or} \left(\begin{array}{l} VD(b) = VD(a) \text{ and} \\ V\_id(b) < V\_id(a) \end{array}\right)\right]$

$$V(b) \subseteq \{V(a) \cup V(c)\} \text{and } V(a) \subseteq \{V(b) \cup V(c)\} \text{and } V(c) \subseteq \{V(a) \cup V(b)\} \qquad (17)$$

and satisfies any of the subsequent conditions:

- $(VD(b) < VD(a))$ and $(VD(b) < VD(c))$
- $(VD(b) = VD(a) < VD(c))$ and $(VD(b) < VD(a))$
- $(VD(b) = VD(a) = VD(c))$ and $\left(\begin{array}{l} V\_id(b) = \\ \min(V\_id(a), V\_id(b), V\_id(c)) \end{array}\right)$
- $(W1(b) = W1(a) > W1(c))\ \&\&$ $\left(\begin{array}{l} VD(b) < VD(a) \text{ or} \\ \left(\begin{array}{l} (VD(b) = VD(a)) \text{ and} \\ (V\_id(b) < V\_id(a)) \end{array}\right) \end{array}\right)$
- $(W1(b) = W1(a) = W1(c))$ and $\left[\begin{array}{l} (VD(b) < VD(a) \text{ and } (VD(b) < VD(c))) \text{ or} \\ (VD(b) = VD(a) < VD(c) \text{and} (V\_id(b) < V\_id(a))) \text{ or} \\ \text{or} \left((VD(b) = VD(a) = VD(c)) \text{ and} \left(\begin{array}{l} V\_id(b) = \\ \min(V\_id(a), V\_id(b), V\_id(c)) \end{array}\right)\right) \end{array}\right]$
- $(W2(b) = W2(a) > W2(c))$ and $\left(VD(b) < VD(a) \text{ or} \left(\begin{array}{l} (VD(b) = VD(a)) \text{ and} \\ (V\_id(b) < V\_id(a)) \end{array}\right)\right)$
- $(W2(b) = W2(a) = W2(c))$ and $\left[\begin{array}{l} (VD(b) < VD(a) \text{ and } (VD(b) < VD(c))) \text{ or} \\ (VD(b) = VD(a) < VD(c) \text{and} (V\_id(b) < V\_id(a))) \text{ or} \\ \text{or} \left((VD(b) = VD(a) = VD(c)) \text{ and} \left(\begin{array}{l} V\_id(b) = \\ \min(V\_id(a), V\_id(b), V\_id(c)) \end{array}\right)\right) \end{array}\right]$

The proposed rules find a near-minimal CDS, which contains more reliable nodes, i.e., these nodes are less prone to failure.

For example, we have considered an undirected connected graph (see Fig. 3), which has 20 vertex or PSs. The performance of the CPU, specific MIPS (millions of instructions per second), network bandwidth, RAM size, and disc storage are used to categorize each PS in this graph. A cloud-based data center often accommodates several concurrent users at once. The host PS may contain n heterogeneous VSs. The leasing of n heterogeneous VSs provisioned on the host PSs is typically requested by users. RAM, disc storage, network bandwidth, and CPU performance needs can all be used to pinpoint VSs.

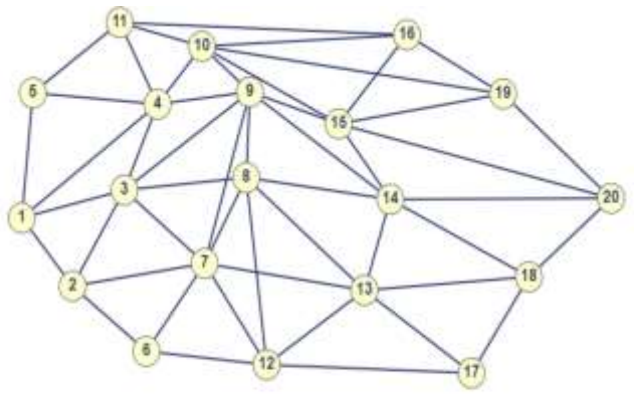


Fig. (3). Undirected Linked Graph

On the graph above, we used both the existing CDS rule and the newly suggested rule to reduce the graph's size and obtain a reliable node.

After applying existing CDS rule, the reduced CDS consists of the following vertices: 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, while the remaining nodes are marked as false.

After applying newly proposed rules, the reduced CDS consists of the following vertices: 3, 4, 7, 9, 10, 11, 13, 14, 15, 16, and the other nodes are marked as false.

From the given result, it can be seen that the proposed rules provided smaller CDS having the more reliable vertices with greater coverage. More dependable PSs will be found in the minimum CDS. Furthermore, this minimized CDS is used to implement CRR FT. An approach to save checkpoint snapshot along with the message log is described below where checkpoint snapshot of task or VS are saved from a host PS to the dominator PS.

4.3. CDSCP Algorithm

We have taken into consideration distributed applications in this study, where the individual tasks or processes interact with one another through messages while being executed. It is crucial that the system does not become inconsistent as a result of recovering from one unsuccessful process. As a result, during regular operation, information regarding both external and internal communications with other processes must be saved. To improve the reliability of the VS-based services of distributed application, a checkpointing-based FT algorithm is presented. The proposed protocol is a Connected Dominating Set based Checkpointing Protocol (CDSCKP). In this protocol, first, the appropriate dominator node of each host PS is located in the data center. A vertex p is said to be a dominator PS or vertex of vertex q if it is a member of CDS and can be accessed from q. The dominator PS is expected to be less prone to failure.

Additionally, each task or process can select its own checkpointing frequency and independently save its checkpoint on the stable storage available at its dominator PS thanks to the condensed CDSCKP protocol. After being logged at the dominator PS, any communication intended for the work is also provided to the process through the nearby PS. The message and information about the sending process are both included in this record. With the use of this technique, any unsuccessful tasks or processes can be recovered separately without affecting or interfering with the states of any successful tasks. The procedure is described in algorithm 1:

Algorithm 1. CDSCP Algorithm

Data Structures Used:

Set of vertices Host_PS[1…n] ,
Set flag for vertex vf[1…m]: Set of flag value for Vertices in the given graph
(Initially set to 1)
Neighbours of vertex Vertex_Nbrs[1…k],
Number of Virtual Servers VSs [1…m],
Marked value of each Vertex Marked_Vertex [i…n]
Dominator_PS[1…n]: List of obtained CDS vertex after applying the proposed rules.
c_i, n: nth checkpoint of PS i;
Recovered_task: Boolean variable to indicate the success of recovery

Procedure:

Finding the Dominator
```
1.  Initialize Dominator_PS [Host_Vertex] = NULL
2.  Initialize Marked_Vertex [ i] = 1 (for the node which are
     the members of CDS)
3.  For each $i \in Host\_PM$ in CDS do
4.    If (Marked_ Vertex [i] == 1)
5.       Then
6.          Dominator_PS [i] = i
7.          Exit
8.
9.     Else
10.      Find neighbors of host vertex i, store in the
```

```
        Vertex_Nbrs
11.     For each $j \in Vertex\_Nbr$ do
12.     Do
13.         If (Marked_ Vertex [j] == 1)
14.         Then
15.             Dominator_PS[i] = j
```

When it is time to take n^th checkpoint c_i, n for PS i

```
16.             For each task k ε Tasks[i] do
17.                 Save checkpoint images of k from the host
                    node to the Dominator_PS j
18.                 Save Message log of k from the host node
                    to the Dominator_PS  j
19.                 End for
20.                 Delete, if exists, $c_i$, n-1
21.             End if
22.         End for
23.     End if
24. End for
```

When a VS k fails

```
Set i= PS of Task k
Set s= SCDS[i]
Set flag = es_flag[s]
If (flag ==1) then
Retrieve last saved checkpoint of task k from the CDS Vertex s
Rollback to the preceding checkpoint of k
Recommence calculation from the last save checkpoint
Set Recovered_task = true
Exit
   End if
Return Recovered_task
```

4.4. Failures of VSs or PSs

In a CDC, node failures happen independently of one another. A host node may malfunction for a number of reasons, including hardware failure, software bugs, and network issues. This paper emphasized hardware malfunctions. We have taken into account each node's CPU's temperature when modeling these failures. If a node's CPU temperature exceeds the temperature threshold value, the CPU temperature may increase exponentially and the node may fail [26] [27].

The heating rate model for the PS is taken from [28] and defined as:

For heating rate h greater than 0 and less than h_i

$$f\left(heating\_rate(h) \mid A, \omega, h_i, h_{i+1}\right) = e^{h} \qquad (18)$$

This mathematical function reflects the variation of processor heating rate when computer boots

For h greater than h_i and less than h_{i+1}

$$f\left(heating\_rate(h) \mid A, \omega, h_i, h_{i+1}\right) = e^{h_i} \qquad (19)$$

For t is greater than t_{i+1} and less than t_{i+2}

$$f\left(heating\_rate(h) \mid A, \omega, h_i, h_{i+1}\right) = A\sin\left(\omega t - \omega t_{i+1}\right) + e^{h_i} \qquad (20)$$

Where i denotes a set of positive integers and h_i is a static value, calculated by

$e^{h_i} = 35$

e^{h_i} designates the processor heating rate without any load, (where e^{h_i} =35°C);

h_{i+1} contains an arbitrary value, and h_{i+2} is calculated by $h_{i+2} = \pi / \omega + h_{i+1}$;

A= amplitude, represents the peak value of processor heating rate (< 68°C);

ω specifies the period for which the processor accomplishes the load.

π / ω used to calculate the first half-round of sinusoidal function;

It is also feasible to arbitrarily alter A's value and to indicate different types of consumption of the CPU in various time frames. This heating rate model is used to predict the deterioration of a PS.

4.5. Power Consumption Model for PSs

Most of the power consumption in the CDC is from disk storage, network, cooling system, and computation processing. The power model used in the literature [25][28] is defined as follows:

$$PM(u) = PS_{idle} + \left(PS_{busy} - PS_{idle}\right) * U^{\epsilon} \qquad (21)$$

Where PSidle denotes power consumed by idle PSs, PSbusy denotes power consumed by a fully loaded PS and U denotes CPU utilization.

5. PERFORMANCE ASSESSMENT

In this section, we've used simulated tests to assess the CDSCKP scheme's efficacy. Additionally, we compared the RCKP (Random Checkpointing Protocol) and WLCKP (Wu

Li based Checkpointing protocol) and proposed methods in terms of recovering from unsuccessful tasks or virtual machines (VSs), total bandwidth usage, total power consumption, and task or VS rollback and recovery times.

5.1. Simulation Step

The performance of the proposed approach implemented and evaluated on the platform having Intel(R) Core (TM) CPU i5-7200 processor having a CPU speediness of 2.70 GHz and storage of 8 GB. The system configuration for CDSCKP is defined in Table 1.

Each job or assignment is evenly dispersed in size between [10 hours, 20 hours] with a 20-minute checkpoint intermission. The system checkpoint image and the VS setup image each have sizes of 1.78 GB and 1.28 GB, respectively [3].

Table 2. System configuration for CDSCKP

5.2. Metrics

The following performance measures are used to gauge the CDSCKP protocol's effectiveness:

Recovery Capability (%): This metric is determined as follows to determine the percentage of tasks or VSs that are successfully recovered:

$$Recoverability = \frac{Total\ Number\ of\ tasks\ or\ VSs\ Recovered\ Sucessfully}{Total\ tasks\ or\ VSs\ Failure} \times 100 \qquad (22)$$

Total Bandwidth Usage (TBWU): - Used to compute the total amount of bandwidth used during the CRR process. The mathematical formula to calculate total bandwidth consumption is defined below:

$$TBWC = \sum_{i=1}^{n}\sum_{j=1}^{m} link\_count\left(Host\_PS_i,\ Dominator\_PS_j\right) \times link\_capacity \qquad (23)$$

Total Power Consumption (TPC): - This metric is used to find the power consumed by all the dominator PSs in the CDC. The values for the power consumption model are described in Table 2. The mathematical formula to compute TEC is defined below:

$$TEC = \sum_{i=1}^{n} E\left(Dominator\_PS_i\right) \qquad (24)$$

Table 3. Values for energy consumptions for different CPU types

Energy Consumption Parameters		
CPU-Type	dominator PS idle	dominator PS busy
8-core	160w	270w
12 to 15-core	210w	360w

Rollback and Recovery Time (RRT): - This metric is used to calculate the rollback and recovery time taken by a failed task. The mathematical formula is defined as follows: -

$$RRT = \sum_{i=1}^{n}\sum_{j=1}^{m} time\left(Host\_PS_i,\ Dominator\_PS_j\right) \qquad (25)$$

5.3. Results

To measure the performance of the newly proposed method we compare it by a RCKP (Random Checkpointing Placement Method) and WLCKP (Wu-Li based Checkpointing Placement method). In RCKP, random dominator PSs are selected to place the checkpoint image and message log. The WLCKP is existing method for reducing the size of CDS based on id of PS.

Platform Configuration	
Type	**Specification**
Minimum CPU heating rate	35oC
Maximum CPU heating rate	68oC
VSs	0 to 4

5.3.1. Number of Reliable Dominator:

In this section, we calculated the number of dominator PS obtained after applying proposed rules of CDS constriction. From Fig. 4, it can be said that the CDSCKP is providing a minimized number of dominator PS, which are most appropriate and reliable for placing the checkpoint images of tasks or VSs and message logs.

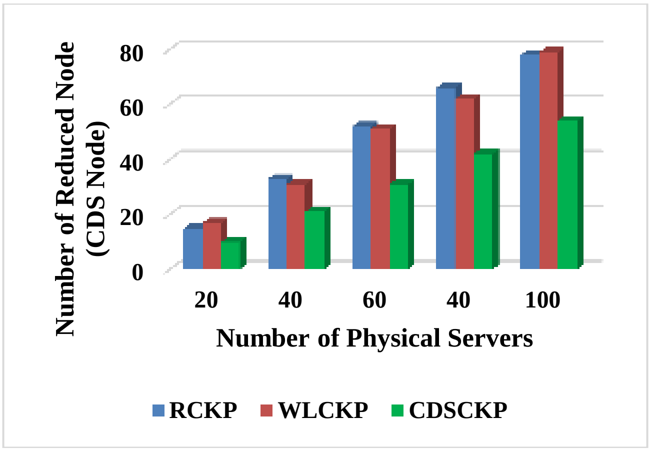


Fig. (4). Minimized Number of dominator PS

5.3.2. Measuring Successful Recoveries

The CDSCKP approach is used in this experiment to calculate the percentage of successfully recovered tasks or VSs during a predetermined time frame. Figure 5 shows how the number of jobs or VSs affects the recoverability of various VSs. The results show that the CDSCKP method achieves a higher recoverability than the RCKP and WLCKP protocols by an average of about 26% and 24%, respectively. Additionally, the recoverability does not decrease as the number of tasks rises, making the PESUCR approach performance scalable.

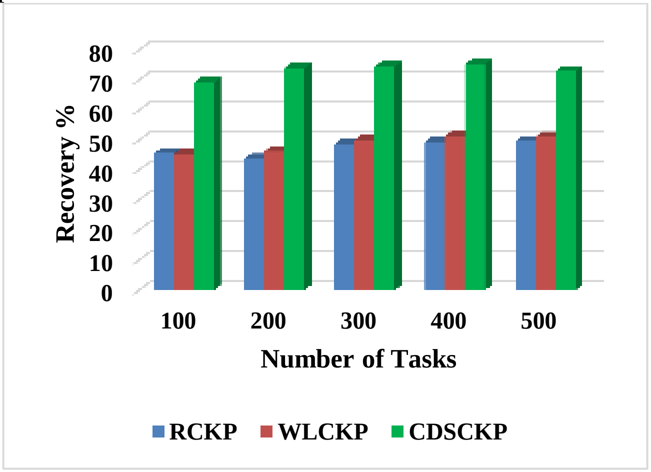
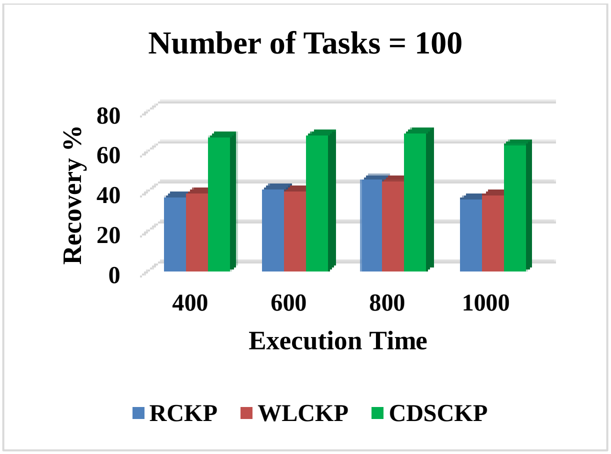
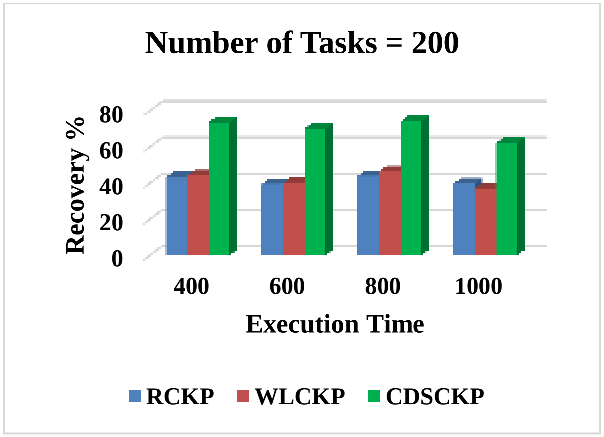
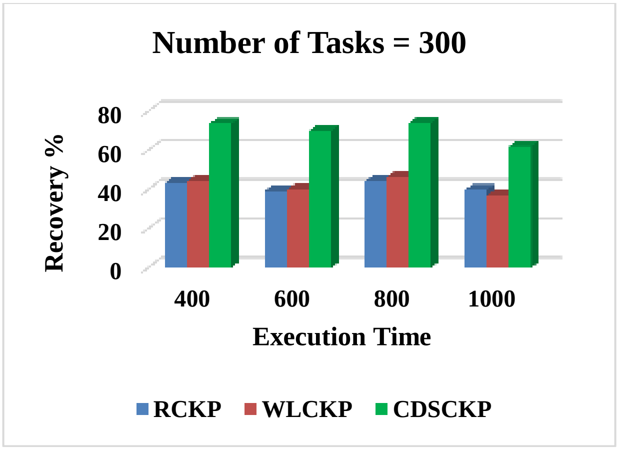


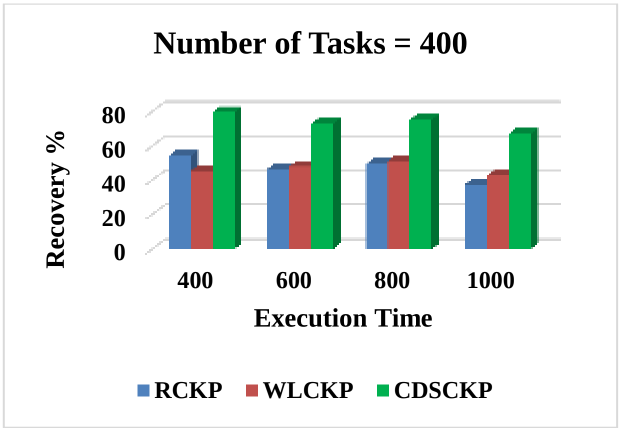
Fig. (5). Recoverability of Varied number of Tasks

Fig. 6 illustrates how the recoverability of RCKP, WLCKP, and CDSCKP changes as time goes on. For all numbers of jobs, CDSCKP consistently yields superior recoverability than the RCKP and WLCKP approaches. The effectiveness of the approaches does not significantly deteriorate with time.









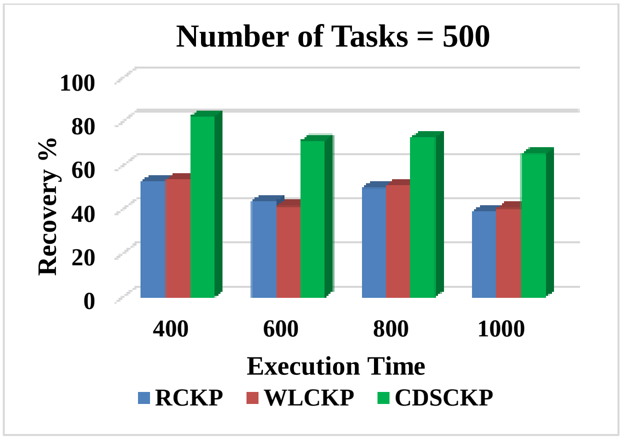
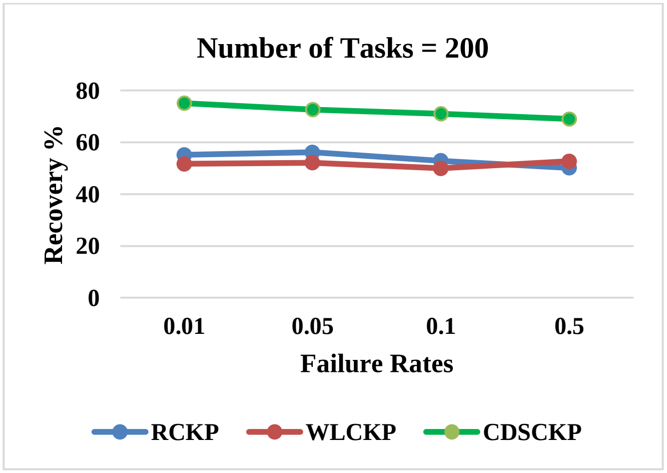


Fig. (6). Recoverability vs. Execution Times

The succeeding simulations assess recoverability when task failure rates change. Failure rate is a measure of how frequently a host or virtual server (VS) fails over a specific time frame [25]. For 200 and 500 tasks, respectively the recoverability of the system for failure rates of 0.01, 0.05, 0.1, and 0.5 is depicted in Fig. 7 (denoting low to high failure rates).



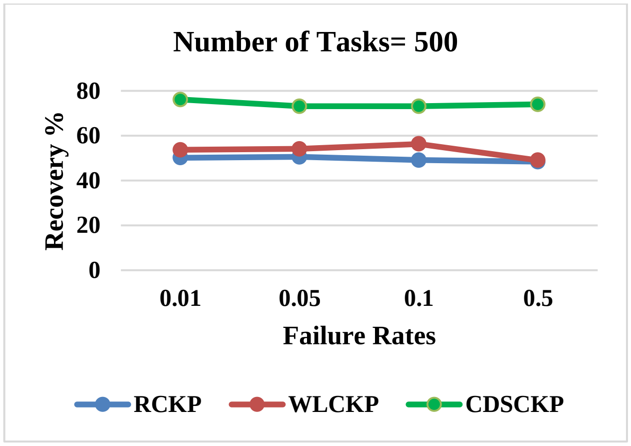


Fig. (7). Recoverability Vs. Failure Rates

5.3.3. Measuring Bandwidth Usage

In this experiment, the bandwidth utilized to deploy checkpoint images for tasks or VSs is measured. The CDSCKP consume an average of approximately 79% less bandwidth than the RCKP. Fig. 8 illustrates the total amount of bandwidth used by different number of PSs.

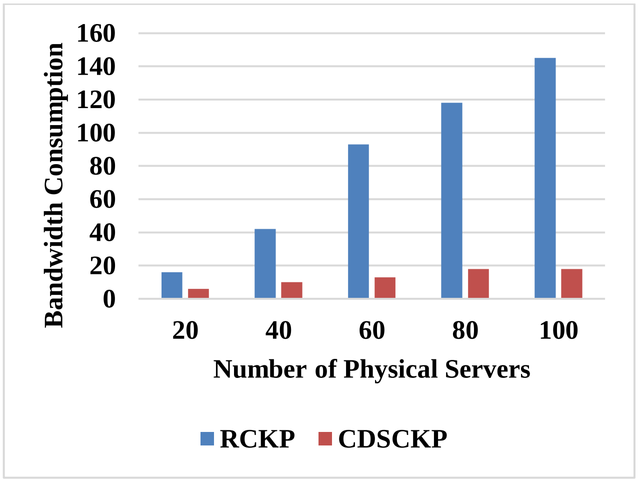


Fig. (8). Total bandwidths used CRR Process

5.3.4. Measuring Power Consumption

This series of tests examines the CPU's power usage when taking a placement checkpoint snapshot and sending it to the dominator PS. Fig. 9 illustrates the total amount of power used by different CPU types.

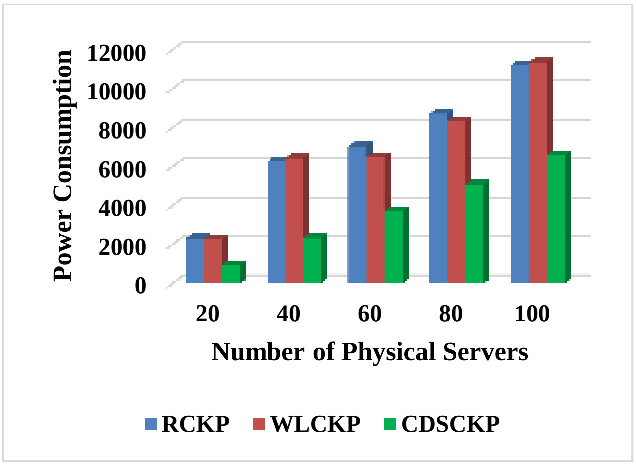


Fig. (9). Total energy consumed by different CPU types

5.3.5. Measuring Rollback and Recovery Time:

The time required for tasks or VSs to rollback and recover is measured in this final set of trials (see Fig. 10). From the result it can be illustrates that the CDSCKP is rollback and recover the failed task in less time as compared to RCKP method.

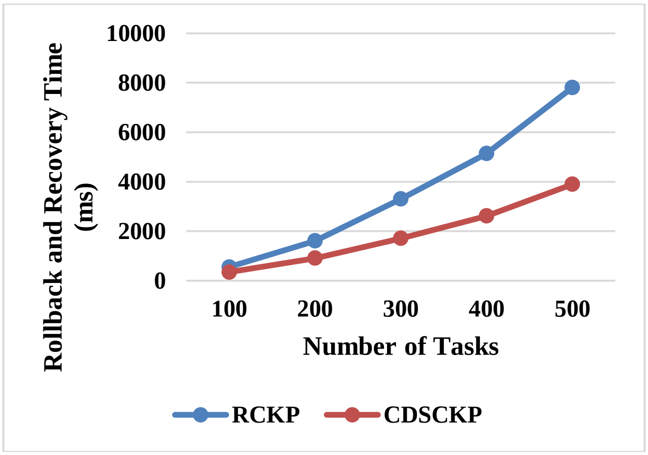


Fig. (10). Rollback and Recovery time of different Tasks

Thus, the simulation's outcomes demonstrate that the CDSCKP protocol is providing the higher recoverability, consume less bandwidth than RCKP and less power than the RCKP and WLCKP protocol. Additionally, CDSCKP takes less time than RCKP to resume execution after rolling back to the last saved checkpoint.

6. CONCLUSION

In order to increase the dependability of VS-based services, this study suggests a CDS-based scheme for building a virtual backbone over the DCN topology. This scheme is then used to produce a CRR-based fault tolerance scheme called CDSCKP. In order to obtain a CDS for the data center topology graph, the work suggests various guidelines based on the rate of CPU heating, storage capacity, and vertex coverage. The PSs with the lowest CPU heating rate, the greatest storage, and the highest vertex degree are chosen to build the CDS, after which the checkpoint image of the task or VS as well as the message log are put on these CDS nodes. Utilizing measures like recoverability, total bandwidth, power consumption, and rollback and recovery times, the CDSCKP's performance is assessed. After that, it is compared with an RCKP and WLCKP. The simulation results show that compared to previous approaches, the CDSCKP offers superior dependability and requires less time and resources for rollback and recovery.

REFERENCES

[1] Kumari P, Kaur P. A survey of fault tolerance in cloud computing. Journal of King Saud University-Computer and Information Sciences. 2021 Dec 1;33(10):1159-76.

[2] Wang T, Su Z, Xia Y, Hamdi M. Rethinking the data center networking: Architecture, network protocols, and resource sharing. IEEE access. 2014 Dec 18; 2:1481-96.

[3] J. Guo, F. Liu, J.C.S. Lui, H. Jin, "Fair network bandwidth allocation in IaaS datacentres via a cooperative game approach," Computer Communications, Vol. 24, No. 2, pp. 873-886, April 2016.

[4] Z. Amin, H. Singh, N. Sethi, "Review on fault tolerance techniques in cloud computing," International Journal of Computer Applications, Vol. 116, No. 18, April 2015.

[5] L.P. Saikia, Y.L. Devi "Fault tolerance techniques and algorithms in cloud computing," International Journal of Computer Science & Communication Networks, Vol. 4, No. 1, pp. 01-08, 2014.

[6] A.Zhou, S.Wang , B.Cheng , Z. Zheng, F. Yang, R. N. Chang, M.R. Lyu, R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," IEEE Transactions on Services Computing Vol. 10, No.6,pp. 902-913, January 2016

[7] A. Zhou, Q. Sun, and J. Li, "Enhancing Reliability via Checkpointing in Cloud Computing Systems," pp. 108–117, 2017.

[8] Zhao, Juzi, et al. "Elastic reliability optimization through peer-to-peer checkpointing in cloud computing." IEEE Transactions on Parallel and Distributed Systems 28.2 (2017): 491-502.

[9] Han, Li, et al. "Checkpointing workflows for fail-stop errors." IEEE Transactions on Computers 67.8 (2018): 1105-1120.

[10] Chinnathambi, Sathya, Agilan Santhanam, Jeyarani Rajarathinam, and M. Senthilkumar. "Scheduling and checkpointing optimization algorithm for Byzantine fault tolerance in cloud clusters." Cluster Computing 22, no. 6 (2019): 14637-14650.

[11] Amoon, Mohammed, Nirmeen El-Bahnasawy, Samy Sadi, and Manar Wagdi. "On the design of reactive approach with flexible checkpoint interval to tolerate faults in cloud computing systems." Journal of Ambient Intelligence and Humanized Computing 10, no. 11 (2019): 4567-4577.

[12] Cheraghlou, Mehdi Nazari, Ahmad Khademzadeh, and Majid Haghparast. "New fuzzy-based fault tolerance evaluation framework for cloud computing." Journal of Network and Systems Management 27, no. 4 (2019): 930-948.

[13] Rezaeipanah, Amin, Musa Mojarad, and Ahad Fakhari. "Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic." International Journal of Computers and Applications (2020): 1-9.

[14] Kumari P, Kaur P. Checkpointing Algorithms for Fault-Tolerant Execution of Large-Scale Distributed Applications in Cloud. Wireless Personal Communications. 2021 Apr;117(3):1853-77.

[15] S.K. Mishra, D. Puthal, B. Sahoo, P. P. Jayaraman, S. Jund, A.Y. Zomaya, R. Ranjan, "Energy-efficient VS-placement in CDC," Sustainable computing: informatics and systems, Vol. 20, pp. 48-55, December 2018.

[16] N.K. Sharma, P. Sharma, R. M. R. Guddeti, "Energy efficient quality of service aware virtual machine migration in cloud computing," in 4th International Conference on Recent Advances in Information Technology (RAIT), IEEE, March 2018.

[17] M. Adda, A. Peratikou, "Routing and fault tolerance in Z-fat tree," IEEE Transactions on Parallel and Distributed Systems, Vol. 28, No .8, pp. 2373-2386, February 2017.

[18] R.K. Devi, G. Murugaboopathi, P. Vijayakumar. "A Graph-based Mathematical Model for an Efficient Load Balancing and Fault tolerance in Cloud Computing," in Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM), IEEE, February 2017.

[19] P.K. Jaggi, A.K. Singh. "Movement-Based Checkpointing and Message Logging for Recovery in MANETs," Wireless Personal Communications, Vol. 83, No.3, pp. 1971-1993, March 2015.

[20] J. Yu, N. Wang, G. Wang, D. Yu, “Connected dominating sets in wireless ad hoc and sensor networks–A comprehensive survey,” International Journal of Computer Applications, Vol. 36, No. 2, pp. 121-134, January 2013.

[21] Singh, Awadhesh Kumar, and Parmeet Kaur Jaggi. "Asynchronous rollback recovery in cluster based multi hop mobile ad hoc networks." International Journal of Enhanced Research in Management &Computer Applications, ISSN (2013): 2319-7471.

[22] Kshemkalyani, Ajay D., and Mukesh Singhal. Distributed computing: principles, algorithms, and systems. Cambridge University Press, 2011.

[23] Mansouri H, Pathan AS. Checkpointing distributed computing systems: An optimisation approach. International Journal of High Performance Computing and Networking. 2019;15(3-4):202-9.

[24] J. Wu, H. Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks," In Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications, ACM, pp. 7-14, August 1999.

[25] X. Dai, J. M. Wang,B. Bensaou, "Energy-efficient virtual machines scheduling in multi-tenant data centers," IEEE Transactions on Cloud Computing, Vol. 4, No. 2, pp. 210-221, September 2015.

[26] J. Liu, S. Wang, A. Zhou, S.A.P. Kumar, F. Yang, R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," IEEE Transactions on Cloud Computing, Vol. 6, No. 4, pp. 1191-1202, May 2016.

[27] Kumari P, Kaur P. Topology-aware virtual machine replication for fault tolerance in cloud computing systems. Multiagent and Grid Systems. 2020 Jan 1;16(2):193-206.

[28] Kumari P, Kaur P. Connected Data Set-based Virtual Machine Replication in Cloud Computing. International Journal of Performability Engineering. 2020 Sep 1;16(9).

[29] Kumari P, Dubey V. Implementation of Fuzzy Rule-Based Inference System for Adaptive Movie Recommendation System. Computer Integrated Manufacturing Systems. 2022 Jan 30;28(1):121-35.

[30] Dubey V, Kumari P, Ojha S, Singh OP, Mishra GR. Issues Confronted By Street Food Vendors During COVID-19 First Wave In India-Word Cloud Analysis. Webology. 2021;18(1):1990-6.