The Journal of Supercomputing (2019) 75:7723–7745
https://doi.org/10.1007/s11227-019-02973-9

Locality-aware process placement for parallel and distributed simulation in cloud data centers

Saad Zaheer[1] · Asad Waqar Malik[1,2] · Anis Ur Rahman[1,2] · Safdar Abbas Khan[1]

Published online: 28 August 2019

Abstract
Cloud is a multi-tenant paradigm providing resources as a service. With its easily available computing infrastructure, researchers are adopting cloud for experimental purposes. However, using the platform efficiently for parallel and distributed simulations comes with new challenges. One such challenge is that the simulations comprise logical processes executing on distributed nodes, traditionally, organized in a sequential pattern. This placement strategy leads to delays as frequently communicating processes might get placed farther from one another. In this paper, we proposed a framework to facilitate implementation and evaluation of process placement algorithms inside a three-tier cloud data center. Furthermore, we used the framework to test different process placement strategies based on classical clustering techniques, as well as, our proposed efficient locality-aware placement algorithm. Our evaluation results show a performance gain of 14.5% for the algorithm in comparison with sequential process placement used in practice.

Keywords Parallel and distributed simulations · Cloud computing · Clustering · Process migration

1 Introduction

The domain of parallel and distributed simulations (PDS) has evolved over the last few decades to support large-scale complex simulations. In general, traditional simulation techniques are developed for closed environments such as an HPC cluster.

✉ Asad Waqar Malik
asad.malik@seecs.edu.pk

Anis Ur Rahman
anis.rahman@seecs.edu.pk

[1] School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan

[2] Department of Information Systems, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia

With the inception of the cloud paradigm, new directions have opened up for the PDS community. Researchers can run complex simulations on a pay-as-you-go model, as available cloud resources (computing, storage, and network) are accessible as a service to any IP-enabled device. To efficiently utilize these resources under the multi-tenant environment, cloud providers use different techniques including virtual machine (VM) migration, VM consolidation, physical host sharing, dynamic bandwidth allocation, and resource scheduling schemes. However, cloud resource utilization remains an open area of research [10]. For PDS community, a cloud provides a different execution platform compared to an HPC cluster environment. Since it provides a shared environment with different applications running alongside PDS processes. The distributed simulation often comprises a large number of participating logical processes (LPs) laid out across the data centers, and communicating with one another using time-stamped messages. However, in a cloud environment, this naive process placement affects the simulation performance as it may be running alongside compute-intensive or data-intensive applications [23]. Moreover, there is a higher probability that frequently communicating processes are placed at distinct physical nodes connected multiple hops apart. This is due to the fact that the process is placed randomly without considering their communication patterns.

The execution of traditional PDS frameworks fails to perform well over the cloud environment, due to the fact that PDS processes generate a large number of time-stamped messages for destination processes connected through multi-hop links inside the data center [17]. This inherent nature of such simulations can clog the underlying network incurring huge delays, affecting the performance of PDS. Over the years, the PDS community has been focused on the cluster environment. Recently, the community has started exploring PDS over the cloud, but no well-known works exist that enhance the distributed simulation performance based on data center network characteristics. In fact, many of the existing frameworks are designed to improve performance using message aggregation, scheduling techniques and/or adoption of a conservative approach. On the contrary, our focus is to reduce the multi-hop traffic using process migration and mitigate the communication delays incurred due to long-haul communications.

In this paper, we proposed an efficient process placement technique to reduce communication delay between frequently communicating processes in PDS. The main contributions of the proposed work are as follows:

- Provide a framework to simulate PDS over the cloud where processes are executed at different physical systems connected to one another through a multi-hop link.
- Propose a placement algorithm termed as find-rack-mate (FRM). The algorithm intelligently placed processes to reduce the overhead in terms of hop count, as PDS processes frequently communicate with one another sharing their state information. In this work, we use clustering-based placement to reduce this overhead, in contrast to traditional random placement techniques that place processes without considering their communication pattern.
- Evaluate PDS over cloud paradigm, we built a data center simulation framework in OMNeT++ to support PDS and its process migration.

The framework serves as a foundation for researchers to test new approaches improving PDS process placement within cloud data center environment.

The remainder of this paper is organized as follows. Related work is presented in Sect. 2. Problem formulation is described in Sect. 3. Section 4 details different clustering-based and proposed process placement technique. The results are discussed in Sect. 6. Section 7 covers the discussion and future directions. Section 8 concludes the research contribution.

2 Related work

Most of the research work in the domain of PDS is based on energy efficiency, load balancing, and simulation performance in a cluster environment. However, communications over the underlying network also play an important role to determine the overall efficiency and robustness of a PDS. In this section, we summarize and review the relevant works on PDS. The existing contributions are organized into cloud-based and cluster-based simulation frameworks.

2.1 PDS frameworks for cloud environment

There exist few works based on PDS execution in cloud paradigm. The execution of PDS over cloud introduces new challenges due to differences from classical cluster environment, such as process synchronization, workload varies at different nodes and significant network traffic. However, with no requirement for an initial investment to use the underlying execution environment, its use to execute simulations is getting popular among academics and industry.

Traditional PDS over cloud environment results in many deadlock scenarios result in increased execution times. To avoid such scenarios, in [28], the authors propose a deadlock-free scheduling algorithm for the execution of PDS over a virtual environment. Additionally, some techniques tend to exploit data locality by scheduling processes accordingly to reduce data transfers, thus, improving the PDS performance. Such data-aware techniques are proposed in [24] mainly focused on work stealing to improve the performance of PDS. However, these techniques are suitable where the simulation relays on large data chunks. In [17], the authors propose a barrier-based protocol to execute an optimistic simulation over the cloud environment. The objective of the protocol is to dynamically handle the compute resource optimization in parallel and distributed simulation.

Over the years, different techniques to improve its performance over the cloud have been proposed, for instance, priority-based work consolidation [16], resource sharing among parallel jobs using gang scheduling algorithm [25], process synchronization based on simulation instances [27] and use of shared event queue among cloud multi-core systems [23]. But the techniques lead to bottlenecks when a large number of threads are involved in the simulation. More recent approach in [14] proposes a federation-based approach to improve the performance of large-scale simulation in a data center. Here, the proposed design

is based on a federation comprising federates grouped into multi-core systems. Each federate contains a number of VMs managed by a federate-level coordinator using hierarchical resource management. Most of the aforementioned attempts are made to improve PDS performance over the shared multi-tenant environment.

Even though multi-threaded design is useful but poses new challenges due to the multi-tenant nature of the cloud environment, for instance, scheduling overhead due to multiple processes sharing a single node can affect PDS performance. Moreover, fault tolerance is important for performance, inadequate fault handling mechanisms often leads to re-execution of the entire simulation [6]. A more comprehensive study in [3] analyzes existing PDS frameworks in terms of usability and adaptability and concludes that the existing techniques have many limitations for use in the cloud paradigm. Thus, there is a requirement for new frameworks that can efficiently run PDS on the cloud.

2.2 PDS frameworks for cluster computing environment

In PDS domain, most of the work exist covers the execution in the cluster system environment. Here, we have briefly covered a few contributions in PDS. In [19], authors presented a master–worker paradigm to support PDS execution of simulations in a distributed environment termed as Aurora. The proposed framework provides computation as web services where PDS processes communicate with one another using timestamped messages. A master process distributes tasks among the workers. After job completion, the workers report back to the master process. The framework avoids local causality constraint by using a conservative synchronization algorithm to achieve high performance and interoperability using the gSOAP toolkit. Moreover, to further improve the performance of PDS, a multi-threaded framework is proposed, i.e., ROSS-MT [13]. It is designed to reduce delays incurred during synchronization. In contrast to standard PDS frameworks, the ROSS-MT used threads for communication instead of processes. The advantage of using threads is the access to shared memory space to improve performance. However, both the contributions Aurora and ROSS-MT have no support cloud architecture.

Similarly, to improve the performance of PDS over cluster environment, in [8], the authors analyze the overhead of global virtual time (GVT) computations. They use a separate thread for communication and computation and only consider GVT calculation to enhance the PDS performance over the cluster. Similarly, in [26], a framework is proposed that appropriately assigns virtual machines to reduce the execution time of PDS. The framework is evaluated in a cluster environment against first-in-first-out and max–min allocation algorithms. The results demonstrate improved PDS efficiency by reducing the execution time for large-scale system simulation.

The performance of PDS over cluster systems is still an open area of research that demands a new process placement, GVT computation, and task distribution algorithms that can support large-scale real-time simulations. With the

multi-disciplinary research, the role of distributed simulation in various fields is such as high-speed experimentation, and big data analytics are the emerging areas of research [21].

Discussion Generally, a huge amount of traffic is generated within the data center. The sources of this traffic include maintenance, VM migration, and process replication. In this study, our proposed framework is designed to support PDS over the cloud with the focus to reduce network cost among frequently communicating processes, thus, improving the overall performance of PDS. Table 1 lists some contributions targeting the PDS performance. It is worthwhile to mention that no previous work exists that provides a cloud data center environment for PDS simulation and supports process migration based on network communication. Thus, reducing network communication means improving overall PDS performance. Other work covers the performance of PDS either using load balancing approach or reducing lock times for shared memory space. The former is difficult to manage in a cloud environment, while the latter is limited by the number of threads supported.

3 System model

In PDS, a significantly large number of processes take part in the simulation, which is randomly placed on different physical nodes across the network. These processes communicate with one another by exchanging messages. Each message has an associated time-stamp value. The physical nodes inside the data center are placed in the form of racks. All nodes residing within the same rack are connected to the top of the rack switch (ToR). Further up the network, multiple racks are connected through an aggregate switch while core switches are used to connect the aggregate switches. This corresponds to a typical three-tiered data center as illustrated in Fig. 1. Note that the separate layer of aggregate switches is added to two-tiered data center architecture to provide scalability in terms of computing servers. To benchmark the proposed model, we used similar three-tiered data center architecture with η number of physical servers placed in every rack, whereas ν represents the number of racks in the data center.

The number of hops during an exchange of messages between two processes depends upon the number of switches the message passes through. Thus, a one-hop communication means that the two processes reside on different nodes on the same rack. Similarly, a three-hop communication means that the two processes reside on different racks with the same aggregate switch. Finally, a five hop message exchange means that communication between processes involves a core switch.

The cost of a n hop communication is n times the cost of single hop communication. Though, negligible it may be, there is a cost associated with the communication between processes on the same node. Therefore, it is pertinent to note that irrespective of the placement of processes, the cost of a message exchange is invariant to their role as a sender or receiver. Thus, a matrix $M = [m(i,j)]_{|P|\times|P|}$ representing number of messages exchange between processes i and j is strictly triangular and for no loss of generality we shall take it to be upper triangular. The order of the square matrix M is the total

Table 1 Summary of recent works in parallel and distributed simulation

Authors	Energy effic.	Netw. effic.	Simul. perform.	Power effic.	Load balancing	Explanation
Alfred et al. [19]	×	×	✓	×	✓	Master worker paradigm
Cai et al. [14]	×	×	✓	×	✓	Hierarchical resource management to augment the execution of large simulations on data centers
Lin et al. [15]	×	×	✓	×	✓	Bandwidth-aware divisible task scheduling
Wang et al. [24]	×	×	✓	×	✓	Data aware scheduling using data aware technique for work stealing
D'Angelo and Marzolla [7]	×	✓	✓	×	✓	Multi-agent system to facilitates adaptive migration
Wang et al. [23]	×	✓	✓	×	×	Proposed multi-threaded PDS simulator
Dong et al. [4]	×	✓	✓	×	×	VMP method to improve network efficiency
Jagtap et al. [13]	×	×	✓	×	×	Proposed ROSS multi-threaded system
Srikanth et al. [28]	×	×	✓	×	×	Deadlock free scheduling
Hassan et al. [12]	×	×	✓	×	×	Machine Learning's classification techniques
Gharibi et al. [11]	×	×	✓	×	×	Linear Integer program
Malik and Mahmood [18]	×	×	✓	×	×	Fault resilient framework
Chen et al. [1]	×	×	✓	×	×	Correlation-aware virtual machine placement system
Duong-Ba et al. [5]	✓	×	×	×	×	Multi-level Joint VM placement and migration algorithms (MJPMs)
Fu et al. [9]	✓	×	×	×	×	Energy aware initial VM placement
Ranjbari and Torkestani [20]	✓	×	×	×	×	Efficient resource allocation in the cloud
Tian et al. [22]	✓	×	×	×	×	Energy-efficient scheduling method for virtual machine reservations
Dai et al. [2]	×	×	×	✓	×	MinPow and MinCom algorithms

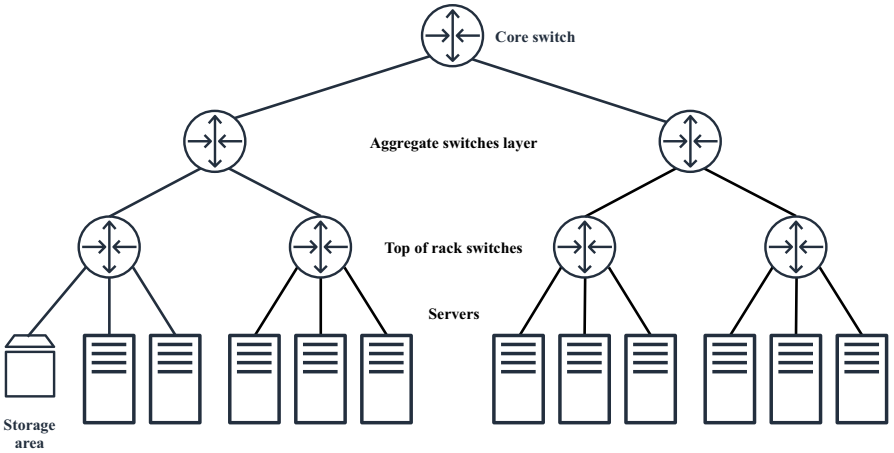

Fig. 1 Traditional three-tier data center architecture

number $|P|$ of processes. Let N denote the set of nodes, then the information regarding a certain node scheduled with specific processes is the operation:

$$f : N \to 2^P$$

such that $f[N]$ is the partition of P. Similar to f, the assignments of nodes to racks R, and racks to aggregate switches A are, respectively, described as

$$g : R \to 2^N \quad \text{and} \quad h : A \to 2^R$$

Note that processes, nodes and racks are referred by their index. For instance, $i \in P$ means the process P_i and $k \in N$ would be the kth node, etc.

To cater for the cost of communication between processes, the number of hops a message encounters is modeled with the help of various 2-arity predicates on P as mentioned below:

$$F(i,j) = 1 \Leftrightarrow \{P_i, P_j\} \subseteq f(k) \quad \text{for some } k \in N$$

The predicate for the (one-hop) communication of process running on different nodes on the same rack is:

$$\begin{aligned} G(i,j) = 1 \Leftrightarrow & F(i,j) = 0 \quad \text{and} \quad \exists\, k \in R, a, b \in N \\ & \text{with } a \neq b \quad \text{s.t. } \{N_a, N_b\} \subseteq g(k) \quad \text{and} \\ & P_i \in f(a) \quad \text{and} \quad P_j \in f(b) \end{aligned}$$

when the processes are located on different racks under the same aggregate switch (three-hop communication), then the predicate is designed as:

$$
\begin{aligned}
H(i,j) = 1 \Leftrightarrow F(i,j) = 0 \quad \text{and} \quad G(i,j) = 0 \quad \text{and} \\
\exists\, k \in A, a_1, a_2 \in R, b_1, b_2 \in N \\
\text{with} \quad a_1 \neq a_2, b_1 \neq b_2 \quad \text{s.t.} \{R_{a1}, R_{a2}\} \subseteq h(k) \\
\text{and} \quad N_{b1} \in g(a_1) \quad \text{and} \quad N_{b2} \in g(a_2) \\
\text{and} \quad P_i \in f(b_1) \quad \text{and} \quad P_j \in f(b_2)
\end{aligned}
$$

Lastly, the (five hop) communication involving the core switch happens to be

$$K(i,j) = 1 \Leftrightarrow F(i,j) = G(i,j) = H(i,j) = 0$$

Here, only one core switch is assumed in the data center, otherwise extension to multiple cores follows the same above mentioned pattern.

If ϵ is the cost of communication between processes residing on the same node and c is the cost of a single hop communication, then the total cost, T, of communication eventually depends upon assignment operations $f, g,$ and h

$$c \sum_{i=1}^{|P|-1} \sum_{j=i+1}^{|P|} m_{i,j} \left(\frac{\epsilon}{c} F_{i,j} + G_{i,j} + 3H_{i,j} + 5K_{i,j} \right)$$

The optimization problem is to find the argument

$$(f, g, h) \in \left(\mathcal{P}(P)\right)^N \times \left(\mathcal{P}(N)\right)^R \times \left(\mathcal{P}(R)\right)^A$$

that minimizes $T(f, g, h)$. The constraints due to hardware in the proposed setup are

$$
\begin{aligned}
g(1) = \{1, 2, ..., 5\} \\
g(2) = \{6, 7, ..., 10\} \\
g(3) = \{11, 12, .., 15\} \\
g(4) = \{16, 17, ..., 20\}
\end{aligned}
$$

Moreover, some additional constraints are the assumption of only two aggregate switches, only one core switch and that only one process is run per node.

4 PDS process placement framework

Execution of PDS without knowledge of process locations can lead to performance issues, mainly due to traffic generated by other applications hosted in the same data center. Moreover, the performance of optimistic simulations in a cloud environment may suffer due to network status with messages stuck on their way to destinations, which may lead to a large number of rollbacks. Interestingly, the literature lacks any works that propose efficient PDS process placement inside the data center to reduce the aforementioned communication delay. Although

research articles exist on VM migration techniques with a focus to reduce overall data center energy consumption; thus, focused on the benefits of cloud providers. The existing energy-aware migration techniques are not well-suited for PDS where LPs are lightweight and frequently generate timestamped messages. Therefore, in this study, we proposed a framework that provides PDS-based three-tier data center environment for process placement. Further, the framework provides PDS process states migration by swapping process states variable with other PDS processes instead of migrating an entire VM. In general, intra-rack node communication is more favorable compared with inter-rack node communication—the latter incurring more hops. Based on this, we propose a locality-aware criterion for process placement, so that to automatically restructure the underlying network for efficient placement of processes in the simulation. Traditionally, in PDS, processes are placed sequentially without knowing their communication patterns, referred to as the random placement method. This is considered the baseline placement strategy. As an initial attempt, we used different clustering techniques to group processes based on their communication patterns. Later in the study, to overcome shortcomings in the mentioned classical clustering techniques, we proposed our locality-aware process placement algorithm.

5 Implementation details

PDS process placement framework for data center was developed atop OMNeT++ and INET framework. The framework comprises two main modules: PDS controller and PDS processes, as illustrated in the framework architecture (Fig. 2). The framework execution steps are listed in Algorithm 1, while the detailed functionality of the two modules is presented below.

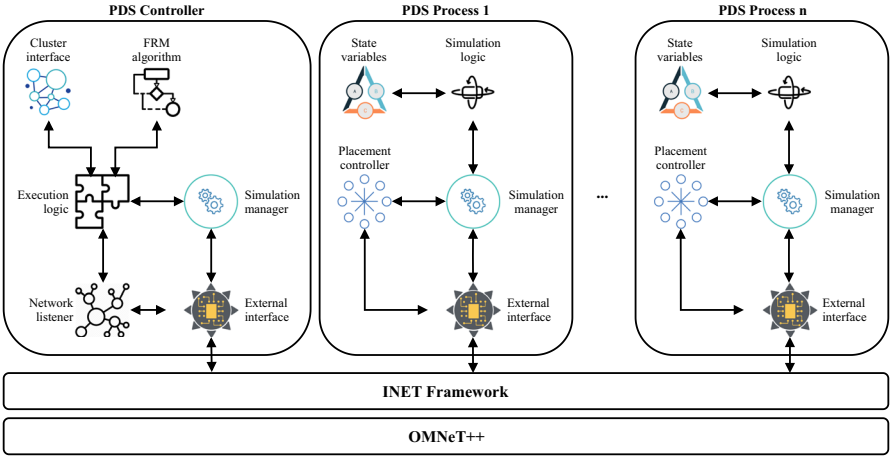

Fig. 2 Proposed framework architecture design

Algorithm 1 PDS cloud framework

```
1: Input : list of processes P, controller C and simulation time t0
2: Output : completion status
3: η ← t0 + Δt                          ▷ Simulation interval for invoking placement
4: initialize(P,C)
5: while true do
6:     if t0 = η then                   ▷ t represents current simulation time
7:         pause(P)
8:         ρ ← invoke(C)                ▷ Returns placement pairs
9:         placement(P, ρ)
10:        reset(η)
11:        resume(P)
12: terminate(P,C)
```

5.1 PDS controller

The PDS controller acts as a manager for the entire simulation. It knows the PDS process placement and underlying communication among the PDS processes. The network listener module is designed to sense all outgoing and incoming traffic at the PDS processes. It uses this information to build a communication matrix for the processes. The simulation manager module keeps track of the PDS process placement in the data center. During the simulation initialization phase, the module develops a process table showing process distances in terms of hop count from every other process. The simulation manager then activates an execution logic module after regular time intervals. The execution module takes the compiled communication matrix and implements a policy to generate PDS process placements. The new placements are handover to the simulation manager, further disseminating them to PDS processes through an external interface.

5.2 Find-rack-mate (FRM) algorithm

The proposed find-rack-mate (FRM) algorithm is designed to find an efficient PDS process placement based on frequently communicating processes. The proposed algorithm communicates with the execution logic module to acquire all the required data for placement recommendations. It takes in the communication pattern from the network listener module and generates rack mates for every participating rack inside the data center. Initially, the processes are placed at random physical systems inside the data center. Based on the initial deployment, the simulation runs for a predefined time interval. During this period, the network listener module generates a communication matrix based on inter-process communication. After the simulation interval ends, the matrix is forwarded to the FRM algorithm to update process placements based on underlying communication pattern.

Algorithm 2 describes the overall functionality of the proposed FRM algorithm. The algorithm starts off by determining candidates for populating the first rack. First, $MCP(P)$ routine finds the highest message-exchanging pair of processes (P_a, P_b) in the communication matrix. This pair is assigned to the current rack; that is, processes (P_a, P_b) are assigned to the first and second positioned physical nodes in the rack. Once done the pair is added to a comparison window W and the current rack pointer is updated. Now starting from the updated position to the maximum number of processes that can reside inside a rack. The routine $MFCVP(W(1))$ returns a process from the communication matrix that communicates most with the first process in the highest communicating process pair (P_a, P_b); that is, the first value of the comparison window W hence $W(1)$. The returned process is termed as P_x. Similarly, a process that communicates most with the second process in the comparison window is represented as P_y. Thus, $MCP()$ finds the highest communicating pair in the entire matrix, whereas $MFCVP()$ finds a process that communicates mostly with a particular process; therefore, it searches only a single row in the communication matrix. Since process P_a and P_b are already a part of the rack; therefore, we compared P_a communication with P_x and P_b communication with P_y, whichever is higher is added to the current position in the rack and that process is removed from the comparison window. The next empty slot in the comparison window is assigned to the process that is just added to the rack for further comparisons. Next step is to

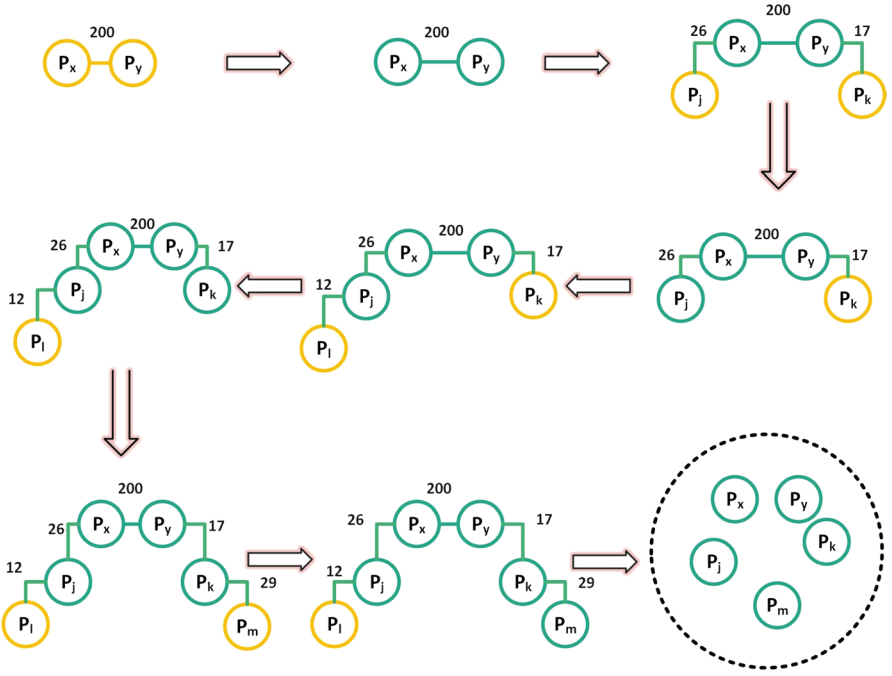

Fig. 3 Find-rack-mate execution model—initially, a pair with the most communications is selected. Thereafter, processes are selected based on their communication with already selected processes. At every stage, the process becomes the part of cluster based on maximum communication with already selected processes. Note that the process id is used to break any ties

remove all the processes from the unassigned process list that are already added to the rack. The sample execution of the algorithm is illustrated in Fig. 3.

Algorithm 2 Find-Rack-Mate()

for $i = 1$ *to* $|R|$ **do**
  $(a, b) \leftarrow mcp(P)$
  $g(i, [1, 2]) \leftarrow (a, b)$
  $W \leftarrow (a, b)$
  for $k = 3$ *to* $|g(i)|$ **do**
    $x \leftarrow mfcvp(W(1))$
    $y \leftarrow mfcvp(W(2))$
    if $M(W(1), x) > M(W(2), y)$ **then**
      $c \leftarrow W(1)$
    else
      $c \leftarrow W(2)$
    $W \leftarrow W \backslash \{c\}$
    $W(2) \leftarrow mfcvp(c)$
    $g(i, k) \leftarrow mfcvp(c)$
    $P \leftarrow P \backslash g(i, k)$
  $P \leftarrow P \backslash (a, b)$
procedure MCP(P)
  Input : *list of processes and* P_{id}
  Output : (a, b) – *most communicating process pairs*
  Find a and b such that
  $M(a, b) = max(M(:))$ ▷ Tie break using P_{id}
procedure MFCVP(W)
  Input : *process index*
  Output : *b such that*
  $M(a, b) = max(M(a, :))$ ▷ Tie break using P_{id}
  $M(a, b) = max(M(:))$ ▷ Tie break using P_{id}

5.3 PDS process

The PDS process is the actual process that executes on the same or different physical system while communicating with other processes and the controller. Here, the simulation logic module contains the actual simulation logic used to modify state variables. The placement controller module receives all the new placements from the controller. In the case of the PDS process migration call, the controller communicates with the simulation manager and performs state swapping. An external interface module is used for communication between the processes and the controller.

5.4 Clustering module

The proposed framework supports PDS process placement based on different clustering techniques. Once new placements are computed, the framework external interface reads cluster information to place the PDS processes accordingly. In this study, we used different cluster techniques to determine similarity between LPs. Here, we used the

sensed communication patterns as the criterion to determine similarity; that is, to identify frequently communicating LPs. That is, initial PDS processes communication patterns are recorded for a number of iterations, later used as input for cluster formation. The clustering techniques used for our initial study are density-based spatial clustering of applications with noise (DBSCAN), hierarchical agglomerative clustering (HAC) and *k*-means clustering. The techniques are briefly discussed as follows:

1. *DBSCAN* The DBSCAN algorithm starts by picking an initial unvisited data point. A neighborhood of data points is defined for the point based on a distance function (ϵ). A criterion referred to as minPoints, the minimum number of points allowed in a neighborhood, is used to either reject the neighborhood as noise or accept it by creating a new cluster. This process is repeated for the data points included in the newly created cluster. For the next cluster, a new unvisited data point is selected and the same process is repeated.
2. *Hierarchical agglomerative clustering* The hierarchical agglomerative clustering (HAC) is a bottom-up clustering approach. It starts off by making every data point a cluster. At each iteration, two clusters with the smallest distance are merged into a single cluster. Note that the merging is controlled by a predefined distance threshold. The merger process is repeated until all the data points fall into one cluster or specified number of clusters is reached.
3. *k-means clustering k*-means is the most well-known clustering technique. It starts off by initializing the number of required clusters. For each cluster, a central point is selected at random. The algorithm starts off by assigning all data points to clusters based on a distance function. The mean of all data points assigned to a cluster is set as the centroid of that cluster. This process is repeated many times until there is no or very small alteration in the resultant clusters.

The proposed framework reads the clustering outcome through its cluster interface module. Note that the technique can either be implemented inside the framework or used as an external tool for cluster formation. In this work, we used an external tool for cluster formation, i.e., Weka. The tool provides a flexible environment to create clusters, which in turn are used for PDS process placement. The next sections cover the performance of the PDS processes using different clustering techniques and the proposed FRM algorithm.

6 Performance evaluation

This section contains a detailed summary of our findings. We implement the proposed process placement algorithm presented in the previous sections, and we compare its communication cost to different clustering techniques used to solve the placement problem in PDS.

6.1 Environment and parameters

The proposed PDS process placement framework is used to evaluate the FRM and clustering-based techniques. We used different clustering techniques including our algorithm to determine clusters within the simulated process communication data. These clusters are used for efficient process placement. Initially, the processes are placed randomly in the simulated data center where the simulation comprises 1.5 million events with a total of 1133 messages exchanged between twenty processes. As a baseline model for comparison, we ran the entire simulation with random process placement and recorded the communication-related statistics. The statistics include the number of messages sent by a process, the number of messages received by a process, the total number of hops incurred by a process, the total number of messages (sent/received), and the mean hop count of a process. On the other hand, for the remaining process placement techniques, we ran the simulation for four hundred thousand events using random process placement and recorded the relevant statistics. These statistics were used to determine process clusters using different clustering techniques. In this study, we kept the number of clusters—groups of processes—fixed to four corresponding to the total number of racks in the simulated three-tier data center. Table 2 contains the list of parameters used for evaluation.

In the simulated data center, a message is transmitted from one node to another took either one, three or five hops. Note that a message consumes less network bandwidth when it requires less number of hops to reach its destination. In this work, the goal was to minimize the hop count (the total number of hops), as this can improve the performance of PDS over cloud environment. Since the cloud charges its users based on usage of computing, storage, and bandwidth. Thus, it seems more suitable to place frequently communicating processes on the same rack, reducing the total network load inside the data center.

6.2 Efficiency of communication costs

For each technique, we measure the efficiency in terms of total hop count after altered process placement. Table 3 summarizes the communication statistics per rack. The results show a break down of message communication costs in terms of one, three, and five hops, as we consider four racks in the simulated three-tier data center. The last column shows the total number of messages, the sum of all one, three, and five hop messages. Furthermore, the results for the proposed FRM algorithm are compared against baseline random placement and placement using different clustering techniques. Similarly, Fig. 4 illustrates the per rack performance with the x-axis representing the four racks and the y-axis representing the number of messages sent with one hop, three hops, and five hops, respectively.

Generally, in PDS, communicating processes are placed randomly across the simulated network. This strategy results in poor simulation performance because the technique lacks any locality awareness and hence ends up generating more network traffic. Using this strategy, out of the total 1133 messages sent during the simulation, 21.5%

Table 2 Simulation configuration and system specification

Parameter	Value
Simulator	OMNeT++/INET
Data center topology	Three-tier
Total number of nodes	20
Number of racks	4
Nodes per rack	5
Maximum hops	5
Communication link delay	
Node ↔ ToR switch	1.78814 ms
ToR ↔ aggregate switch	1.78814 ms
Aggregate ↔ core switch	0.178814 ms
Link capacity	
Node ↔ ToR switch	1 GE
ToR ↔ aggregate switch	1 GE
Aggregate ↔ core switch	10 GE
Total events	1.5 M
Events for training	400,000
Events for testing	1.1 M
Total messages exchanged	1133
Placement techniques	Random, *k*-means, DBSCAN, HAC, proposed FRM
Instances	20
Attributes	5 (all numeric)
Initial starting points	Random
Number of clusters	4 (maximum)
Maximum number of iterations	6
k-means	
Type	Partitional
(k, init, metric, max-iter)	(4, random, Manhattan distance, 6)
DBSCAN	
Type	Density-based
(ϵ, min-samples, metric)	(10, 2, Manhattan distance)
HAC	
Type	Linkage-based
(k, linkage, metric)	(4, single linkage, Manhattan distance)
CPU	3.1 GHz Intel Core i5
RAM	8 GB
OS	Windows 10

were one hop messages, 26.8% were three hop messages and 51.6% were five hop messages. It is evident that each rack has more five hop messages compared to messages with one hop and three hops. This is a clear case of more traffic across the network.

Table 3 Quantitative comparison of hop counts from five different clustering algorithms including our proposed algorithm

Rack	Hops per message			Total messages
	1 hop	3 hops	5 hops	
Random				
#1	61	89	175	325
#2	43	45	101	189
#3	62	71	150	283
#4	78	99	159	336
Total	244	304	585	1133
%	21.54	26.83	51.63	–
DBSCAN				
#1	33	51	117	201
#2	117	137	234	488
#3	56	54	99	209
#4	59	72	104	235
Total	264	314	554	1133
%	23.39	27.71	48.90	–
HAC				
#1	46	64	101	211
#2	37	50	107	194
#3	94	116	147	357
#4	81	113	177	371
Total	258	343	532	1133
%	22.77	30.27	46.95	–
k-means				
#1	42	55	104	201
#2	161	167	161	489
#3	65	51	92	208
#4	70	68	97	235
Total	338	341	454	1133
%	29.83	30.10	40.07	–
Proposed				
#1	41	43	108	192
#2	106	73	90	269
#3	67	56	94	217
#4	182	124	149	455
Total	396	296	441	1133
%	34.95	26.12	38.92	–

The very first clustering technique used to cluster processes was the DBSCAN algorithm. Our results show that each rack sent a higher number of five hop messages compared to messages sent with one hop and three hops. Using this technique, out of the total 1133 messages sent through the simulation, 23.3% messages were one hop, 27.7% messages were three hops and 48.8% messages were five hops. This

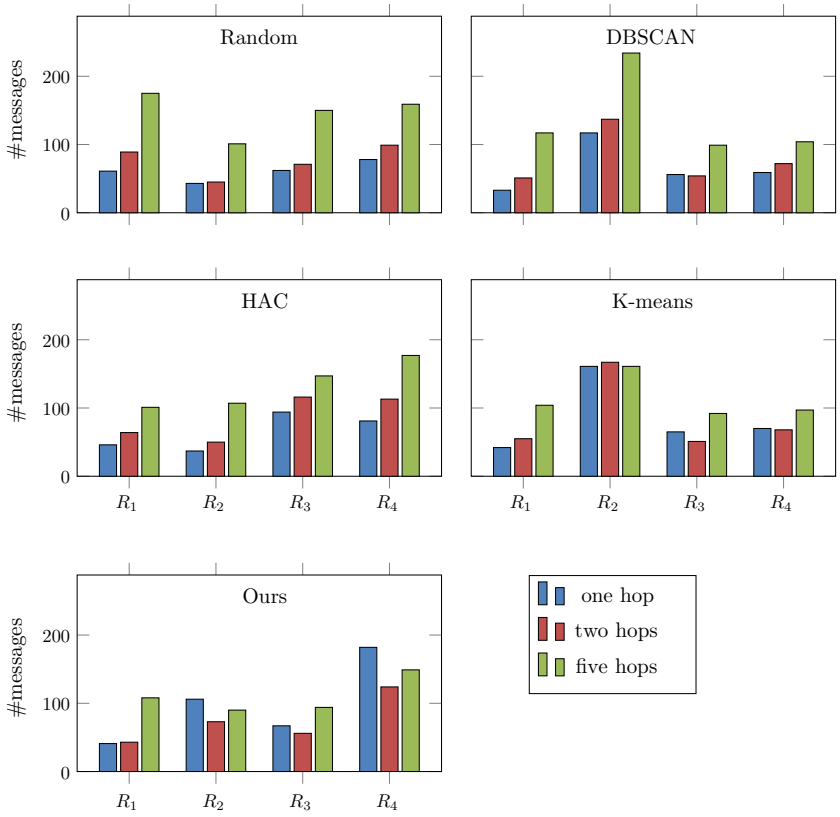

Fig. 4 Per rack performance of the proposed algorithm against five different clustering techniques

represents a minimal improvement with one hop messages increased by only 1.85% while five hops messages decreased by just 2.7%.

The next clustering algorithm used to cluster processes into different racks was based on a hierarchical clustering algorithm. Using this technique, the increase in the number of one hop messages was 1.2% only; however, a decrease of 4.67% was observed in messages sent using five hops. This was a significant improvement compared to when using random process placement.

Out of all the traditional clustering algorithms, the best performing algorithm was *k*-means. The algorithm surpassed random process placement method, DBSCAN and HAC in terms of performance. The number of messages taking one hop to reach their destinations increased by 8.29%, messages taking three hops increased by 3.26% and ones taking five hops decreased by 11.56%. These results demonstrate an improvement using *k*-means compared to clustering techniques presented earlier.

The proposed algorithm outperformed the baseline random process placement method, and all other techniques discussed including the best performing *k*-means-based process placement. Out of the total messages sent during the simulation, 34.95% of the messages took just one hop, 26.12% messages took three hops and 38.92% messages

took five hops to reach their destination. In comparison with the random method, one hop messages increased by a significant 13.4%, the messages with three hops increased slightly by 0.7% and messages with five hops decreased by a significant 12.71%.

Figure 5 depicts the performance of all the aforementioned techniques in terms of hops incurred after newly placed processes at the data center level for the entire run of the simulation. Here, the x-axis lists the techniques, whereas the y-axis presents the total number of messages sent.

Figure 6 shows the communication delay-based comparison between proposed and clustering techniques. The data value shows hop-wise total delay computed on a three-tier data center topology with link and communication delay mentioned in Table 2. In the proposed work, one hop communication is maximum compared to all other techniques; however, the max hop communication is reduced significantly and based on that delay is also reduced. The total delay shows a minimum 3.32% reduction in overall communication delay.

In summary, a total of 1133 messages were sent during the entire run of the simulation. Each message took either one, three or five hops to reach its destination. It is evident that the simulation performance drops if the majority of messages took five hops. On the contrary, the performance is much better if the majority of messages took just one hop to reach their destinations. Note that a better scenario is if the majority of five hop messages took a lesser number of hops, for instance, three hops. Table 4 shows a comparison of hop counts for different process placement techniques including our proposed approach. The results show a hop count reduction of 2.5%, 3.28%, 11.02%, and 14.50% compared to the baseline random process placement method using DBSCAN, hierarchical clustering, k-means, and our proposed method, respectively.

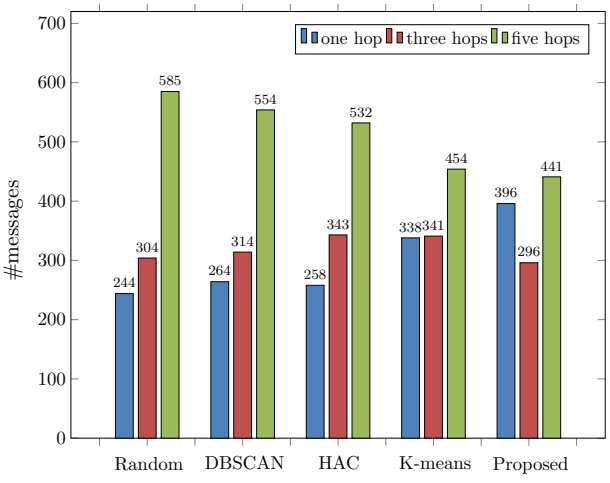

Fig. 5 Comparison of the proposed algorithm against five different clustering techniques

Table 4 Number of hops comparison for five clustering-based process placement techniques including our proposed approach

Technique	Hops	Gain (%)
Random	4081	–
DBSCAN	3977	2.54
HAC	3947	3.28
k-means	3631	11.02
Proposed	3489	14.50

The gain is computed as a ratio of clustering technique with the baseline random placement technique

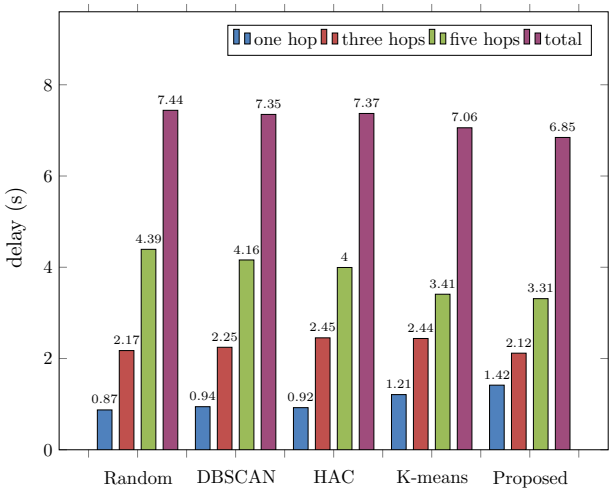

Fig. 6 Comparison of the proposed algorithm against five different clustering techniques in terms of delay

7 Discussion

In PDS, random process placement in a data center can affect the performance of a distributed simulation. The messages can get delayed inside the network due to high network traffic generated by other applications. For instance, in optimistic simulations, network delays can deteriorate the performance due to straggler messages. These messages initiate the rollback mechanism. Often this delay is caused due to a shared multi-tenant environment where other compute-intensive or communication intensive applications are executing. Therefore, placing such communicating processes close to one another, not only reduce the rollbacks but also the overall cloud usage cost. In other words, long-distance communication between processes is considered an infeasible approach for PDS. This is evident from our experimental evaluations that process placement, in order to reduce total communication hops, can improve the overall performance.

- *Locality-aware clustering* Organizing processes based on locality can improve performance and reduce network usage. In most of the cases, a process only communicates with some predefined processes. This is somewhat based on the simulation topology, for instance, in the case of tree-based topology, a root process directly communicates with its children. Here, the process can be placed—based on its locality—along with the directly connected nodes. In our proposed approach, we cluster processes based on their communication patterns to reduce communication costs.
- *Online extension of the algorithm* The proposed model can be extended to work in an online scenario where a master process can track all involved processes to identify communication patterns. Similar to the proposed approach, these patterns can be used to migrate the processes. It is worthwhile to mention that this can result in frequent migrations, an overhead for the system, which will require a mechanism. In the future, we are also interested to use deep learning techniques for PDS process placement. As all these techniques required a significant amount of data therefore, our proposed framework can facilitate in generating large-scale data sets.
- *Implementation on a real-world data center* We are interested to extend our work to support any distributed framework designed for data centers. In the present study, cluster size is user-defined—the number of processes residing in a rack. In future, an extended controller can automatically balance processed based on the underlying computing capacity. Moreover, processes consolidation techniques can be adapted to reduce the number of racks used for PDS.
- *Management implications* The proposed work introduces a new research paradigm to the PDS community, that is PDS process placement based on communication pattern. Generally, in a data center, VMs are placed to reduce the overall energy consumption; however, such placements are managed by the vendors to their benefit, with users having no control over the placements. It is worthwhile to mention that existing PDS frameworks are not designed to work on such a multi-tenant environment, with uneven workloads at physical nodes and network switches due to the execution of different applications. The proposed PDS process placement provides a dynamic environment where PDS processes can be moved to other physical nodes to improve the efficiency of the simulation model. This dynamic movement of the PDS processes can be based on data centers internal factors such as network congestion at various switches, available link capacity, and execution overhead on shared physical nodes. Furthermore, adopting the cloud paradigm for complex simulation models gives scalability and fault resilience when running large simulations. That is, the data center can dynamically manage failures through resource provisioning. However, for successful execution of PDS after node failures would require middleware to maintain causality among the PDS processes to produce correct results.
- *Time complexity* The time complexity of the procedure MCP is $\mathcal{O}(n^2 \log n)$ and that of the procedure MFCVP is $\mathcal{O}(n \log n)$. The time complexity of the inner for loop is $\mathcal{O}(n^2 \log n)$ and consequently the time complexity of the outer for loop and hence that of the FRM algorithm $\mathcal{O}(n^3 \log n)$. The dynamic placement of processes is a search problem, i.e., an NP problem. Under the already discussed

constraints, the heuristically proposed process placement technique not only presents a polynomial time solution, but it also surpasses the existing methods in the literature in terms of communication cost minimization as is shown in Fig. 7.

- *Convergence analysis* For the analysis, we used sigma (σ) convergence inspired from economics. In the context of process placement, it refers to the reduction in dispersion between hop counts at different racks. For describing this convergence tendency, we use a discrete time interval based on the messages communicated till t with random placement and thereafter $t + T$ using the proposed FRM algorithm for process placements. The convergence state at a certain time is measured using an indicator of variation. In contrast to the concept of convergence, this indicator represents one calculated using coefficient of variation σ at time t.

$$\sigma_t = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\log(y_{it}) - \log(\bar{y}_t))^2}$$

where y_{it} is the hop count at time t and $\bar{y}_t$ is the mean hop count at time t. In the present context, the degree of convergence increases when σ is increasing. The higher is the σ, the higher is the degree of convergence (i.e., $\sigma_t < \sigma_{t+T}$). We observe that σ stood at 0.017 by the end of the period t, suggesting a period of divergence (i.e., $\sigma_t > \sigma_{t+T}$). This increased to 0.111 by the end of the period $t + T$, suggesting an evidence for σ-convergence. The trend in the σ over the full time period analyzed (Fig. 7) was negative and statistically significant ($\alpha_0 = 0.05$, $F = 13.276$, $p < 0.001$).

Though the domain of process placement to improve the efficiency of PDS is relatively unexplored. Many previous works mainly used load balancing, scheduling, and energy to improve the performance of PDS. However, in this study, we

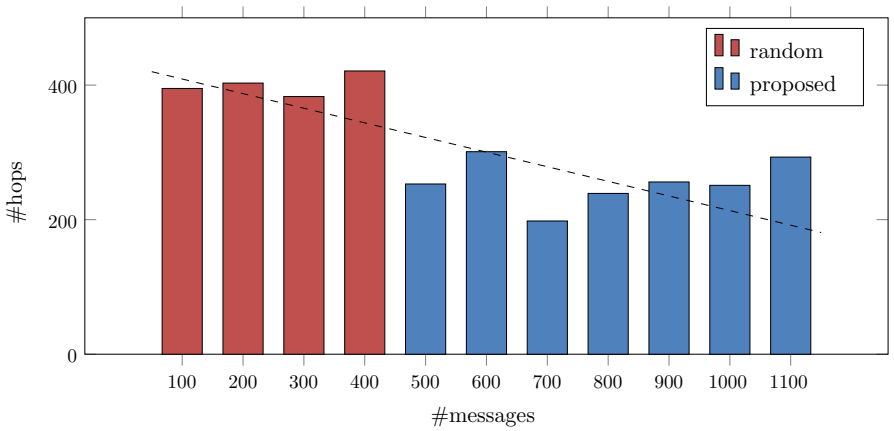

Fig. 7 Trend in number of hop counts with discrete time interval till t using random placement and thereafter till $t + T$ using the proposed FRM algorithm

introduced process placement to improve this performance by reducing the total number of communication hops. In future, we intend to extend the work by addition of dynamic load balancing, process consolidation, and adaptive learning techniques. This will further enhance the performance of PDS under real data center environment.

8 Conclusion

Execution of PDS over the cloud, a multi-tenant computing environment, affects its performance. That is, frequently communicating PDS processes get stuck inside the network due to the network traffic generated by other applications hosted on the cloud, consequently, slowing down the entire simulation. In this work, we demonstrate the generation of communication data from the framework for a three-tier cloud data center, which is used to perform PDS process placement using different unsupervised clustering-based techniques. We also proposed FRM algorithm for process placement alongside the aforesaid techniques, showing a 14.5% improvement. The proposed PDS process placement in the cloud paradigm provides a dynamic environment for running complex simulation models with support for scalability and fault resilience.

References

1. Chen T, Zhu Y, Gao X, Kong L, Chen G, Wang Y (2018) Improving resource utilization via virtual machine placement in data center networks. Mob Netw Appl 23(2):227–238
2. Dai X, Wang JM, Bensaou B (2014) Energy-efficient virtual machine placement in data centers with heterogeneous requirements. In: 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet). IEEE, pp 161–166
3. D'Angelo G (2011) Parallel and distributed simulation from many cores to the public cloud. In: 2011 International Conference on High Performance Computing and Simulation. IEEE, pp 14–23
4. Dong JK, Wang HB, Li YY, Cheng SD (2014) Virtual machine placement optimizing to improve network performance in cloud data centers. J China Univ Posts Telecommun 21(3):62–70
5. Duong-Ba TH, Nguyen T, Bose B, Tran TT (2018) A dynamic virtual machine placement and migration scheme for data centers. IEEE Trans Serv Comput. https://doi.org/10.1109/TSC.2018.2817208
6. D'Angelo G, Ferretti S, Marzolla M (2019) Fault tolerant adaptive parallel and distributed simulation through functional replication. Simul Model Pract Theory 93:192–207
7. D'Angelo G, Marzolla M (2014) New trends in parallel and distributed simulation: from many-cores to cloud computing. Simul Model Pract Theory 49:320–335
8. Eker A, Williams B, Chiu K, Ponomarev D (2019) Controlled asynchronous GVT: accelerating parallel discrete event simulation on many-core clusters. In: 48th International Conference on Parallel Processing (ICPP 2019), pp 5–8
9. Fu X, Zhao Q, Wang J, Zhang L, Qiao L (2018) Energy-aware vm initial placement strategy based on bpso in cloud computing. Sci Program. https://doi.org/10.1155/2018/9471356
10. Fujimoto RM (2016) Research challenges in parallel and distributed simulation. ACM Trans Model Comput Simul (TOMACS) 26(4):22

11. Ghribi C, Hadji M, Zeghlache D (2013) Energy efficient VM scheduling for cloud data centers: exact allocation and migration algorithms. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE, pp 671–678
12. Hassan M, Babiker A, Amien M, Hamad M (2018) SLA management for virtual machine live migration using machine learning with modified kernel and statistical approach. Eng Technol Appl Sci Res 8(1):2459–2463
13. Jagtap D, Abu-Ghazaleh N, Ponomarev D (2012) Optimization of parallel discrete event simulator for multi-core systems. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium. IEEE, pp 520–531
14. Li Z, Li X, Wang L, Cai W (2014) Hierarchical resource management for enhancing performance of large-scale simulations on data centers. In: Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. ACM, pp 187–196
15. Lin W, Liang C, Wang JZ, Buyya R (2014) Bandwidth-aware divisible task scheduling for cloud computing. Softw Pract Exp 44(2):163–174
16. Liu X, Wang C, Zhou BB, Chen J, Yang T, Zomaya AY (2012) Priority-based consolidation of parallel workloads in the cloud. IEEE Trans Parallel Distrib Syst 24(9):1874–1883
17. Malik A, Park A, Fujimoto R (2009) Optimistic synchronization of parallel simulations in cloud computing environments. In: 2009 IEEE International Conference on Cloud Computing. IEEE, pp 49–56
18. Malik AW, Mahmood I (2017) Crash me inside the cloud: a fault resilient framework for parallel and discrete event simulation. In: Proceedings of the Summer Simulation Multi-Conference. Society for Computer Simulation International, p 1
19. Park A, Fujimoto RM (2006) Aurora: an approach to high throughput parallel simulation. In: 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06). IEEE, pp 3–10
20. Ranjbari M, Torkestani JA (2018) A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers. J Parallel Distrib Comput 113:55–62
21. Taylor SJ (2019) Distributed simulation: state-of-the-art and potential for operational research. Eur J Oper Res 273(1):1–19
22. Tian W, He M, Guo W, Huang W, Shi X, Shang M, Toosi AN, Buyya R (2018) On minimizing total energy consumption in the scheduling of virtual machine reservations. J Netw Comput Appl 113:64–74
23. Wang J, Jagtap D, Abu-Ghazaleh N, Ponomarev D (2013) Parallel discrete event simulation for multi-core systems: analysis and optimization. IEEE Trans Parallel Distrib Syst 25(6):1574–1584
24. Wang K, Zhou X, Li T, Zhao D, Lang M, Raicu I (2014) Optimizing load balancing and data-locality with data-aware scheduling. In: 2014 IEEE International Conference on Big Data (Big Data). IEEE, pp 119–128
25. Wiseman Y, Feitelson DG (2003) Paired gang scheduling. IEEE Trans Parallel Distrib Syst 14(6):581–592
26. Yao F, Yao Y, Chen H, Li T, Lin M, Zhang X (2019) An efficient virtual machine allocation algorithm for parallel and distributed simulation applications. Concurrency Comput Pract Experience. https://doi.org/10.1002/cpe.5237
27. Yao F, Yao Y, Chen H, Li T, Lin M, Zhang X (2019) An intelligent scheduling algorithm for complex manufacturing system simulation with frequent synchronizations in a cloud environment. Memet Comput. https://doi.org/10.1007/s12293-019-00284-3
28. Yoginath SB, Perumalla KS (2015) Efficient parallel discrete event simulation on cloud/virtual machine platforms. ACM Trans Model Comput Simul (TOMACS) 26(1):5