Journal of Parallel and Distributed Computing 138 (2020) 15–31

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

On demand clock synchronization for live VM migration in distributed cloud data centers

Yashwant Singh Patel [a,*], Aditi Page [b], Manvi Nagdev [c], Anurag Choubey [a], Rajiv Misra [a], Sajal K. Das [d]

[a] Department of Computer Science and Engineering, Indian Institute of Technology Patna, Bihar 801106, India
[b] Department of Computer and Information Science Engineering, University of Florida, FL, USA
[c] Department of Computer Science, North Carolina State University, NC, USA
[d] Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409, USA

ARTICLE INFO

Article history:
Received 5 May 2019
Received in revised form 1 November 2019
Accepted 29 November 2019
Available online 19 December 2019

Keywords:
Cloud data centers
Live VM migration
Data center Time Protocol (DTP)
Precision Time Protocol (PTP)
Data center networks

ABSTRACT

Live migration of virtual machines (VMs) has become an extremely powerful tool for cloud data center management and provides significant benefits of seamless VM mobility among physical hosts within a data center or across multiple data centers without interrupting the running service. However, with all the enhanced techniques that ensure a smooth and flexible migration, the down-time of any VM during a live migration could still be in a range of few milliseconds to seconds. But many time-sensitive applications and services cannot afford this extended down-time, and their clocks must be perfectly synchronized to ensure no loss of events or information. In such a virtualized environment, clock synchronization with minute precision and error boundedness are one of the most complex and tedious tasks for system performance. In this paper, we propose enhanced DTP and wireless PTP based clock synchronization algorithms to achieve high precision at intra and inter-cloud data center networks. We thoroughly analyze the performance of the proposed algorithms using different clock measurements. Through simulation and real-time experiments, we also show the effect of various performance parameters on the data center networking architectures.

1. Introduction

With the expeditious growth of cloud for computation, storage and networking, live migration of VMs [6] has given rise to the need for effectively managing data centers. Migration of VMs from one data center to another has made it possible to conveniently maintain data centers without affecting much of the performance of VMs. In live migration, data from a physical machine (PM) is copied to destination PM in another data center while the VM is continuously running on the former PM. Once the data is copied, the VM is continued on the new PM. It ensures negligible downtime and thus achieves high performance. During the migration of VMs, clock synchronization becomes paramount [2–4]. It is very crucial for the clocks to be synchronized and that too, at higher precision. Since a lot of systems and many time-sensitive applications and services such as distributed transactions (e.g., Google Spanner) [9], SSL-based, smart grid [15], disaster alert, safety-critical [13], stream gaming and mobile edge computing based [1] depends on it, and if not done, it can result in loss of data, unsuccessful scheduled operations, or problems in monitoring log files [11,16]. Various clock synchronization protocols, which were traditionally used on the internet for packet transmissions via ethernet, IEEE 802.11, or coaxial cables can be extended to provide a solution to the virtualized architecture. Protocols like NTP, PTP, GPS etc. [12,14] have been used in this regard. Network Time Protocol (NTP) is the most frequently used synchronization protocol. It achieves microsecond precision in a local area network (LAN) and millisecond precision in a wide area network (WAN). Due to its low precision and inaccurate time stamping nature, it is not feasible for cloud data centers, especially for live VM migrations. Recently a new clock synchronization protocol for synchronizing clocks globally in a data center, named as data center clock synchronization protocol (DTP), has been proposed [10]. This protocol uses the physical layer of the devices to synchronize clocks since it reduces the additional overhead from the layers above. This protocol gives bounded precision in sub-nanoseconds and hundreds of nanoseconds depending on the number of hops required for data transmission. Also, the protocol is highly scalable and can synchronize an entire data center. One major drawback associated with the use of DTP across data centers is it requires the modification at the physical layer of all the network devices. However,

* Correspondence to: 513, Advanced Lab, Block-3 Department of Computer Science and Engineering, Indian Institute of Technology Patna, 801106, India.
E-mail address: yashwant.pcs17@iitp.ac.in (Y.S. Patel).

https://doi.org/10.1016/j.jpdc.2019.11.012
0743-7315/

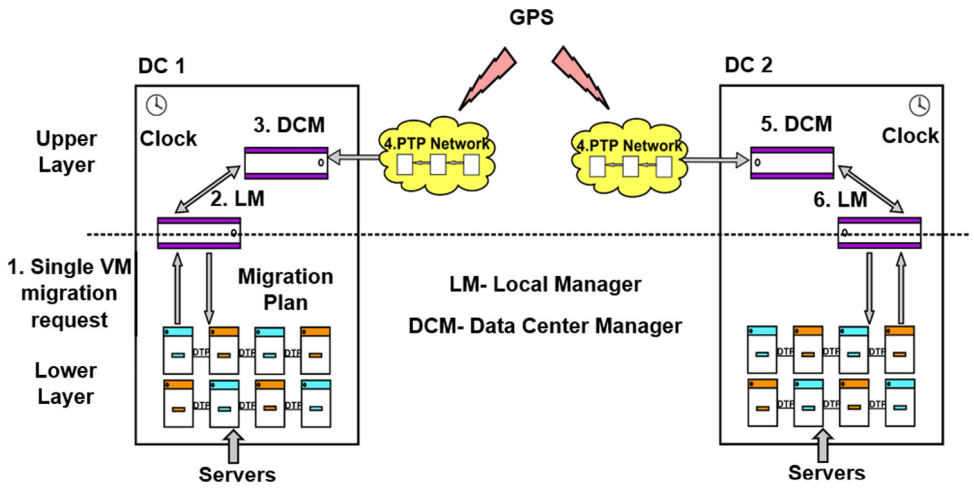

Fig. 1. Inter-data center scenario.

Precision Time Protocol (PTP) [8] is a generic approach of time stamping and used to synchronize clocks across the data center networks and achieve sub-microsecond precision. PTP uses the master–slave clock synchronization architecture where the root of the synchronization tree is considered as a master and the children of a tree become a slave to its parent [14]. The most prominent external time synchronization protocol is the Global Positioning System (GPS) [10]. It achieves about 100 nanosecond precision via the use of atomic clocks or satellites. In practice, for a large data center, GPS based synchronization solutions are not feasible and realistic because it requires a lot of extra cables and also depends on the GPS signal availability. Although GPS can be used in tandem with DTP, PTP, and NTP.

All these protocols work with millisecond, microsecond and sub-microsecond precision but during live VM migration, synchronization cannot be achieved with such precision. Hence, selection of the most feasible time synchronization protocol is a very important and challenging task [7]. Thus, to deal with clock synchronization during live VM migration, we propose enhanced DTP [10] and wireless PTP [5] based approaches at the intra-data center and inter-data center level.

The key contributions of the work are : (i) Firstly, we structured four well-known data center networking architectures such as Fat-tree, Bcube, Three-tier, VL2 for cloud systems and built the scenarios for both intra-data center and inter-data center architectures. (ii) Secondly, based on the live VM migration operations, we have formulated various measures to ensure efficient synchronization at different data center levels. (iii) Thirdly, we present enhanced DTP and wireless PTP based two different algorithms to return a fairly well synchronized live VM migration in practice (iv) Through simulation results; we finally demonstrate the optimal precision with different loss measurements such as error bounds, convergence time, clock variance, mean synchronization time and connectivity radius.

2. System model and architectural overview

In this section, we briefly discuss the overall view of the synchronization and migration process. The model consists of 2 layers: (i) The lower layer is for handling the migration process within a data center and, (ii) The upper layer is responsible for migration between the data centers. As shown in Figs. 1 and 2, the main components in the architecture are:

1. Data Center Manager (DCM): It is responsible for coordinating the migration activities by obtaining the information from the Local Manager (LM).

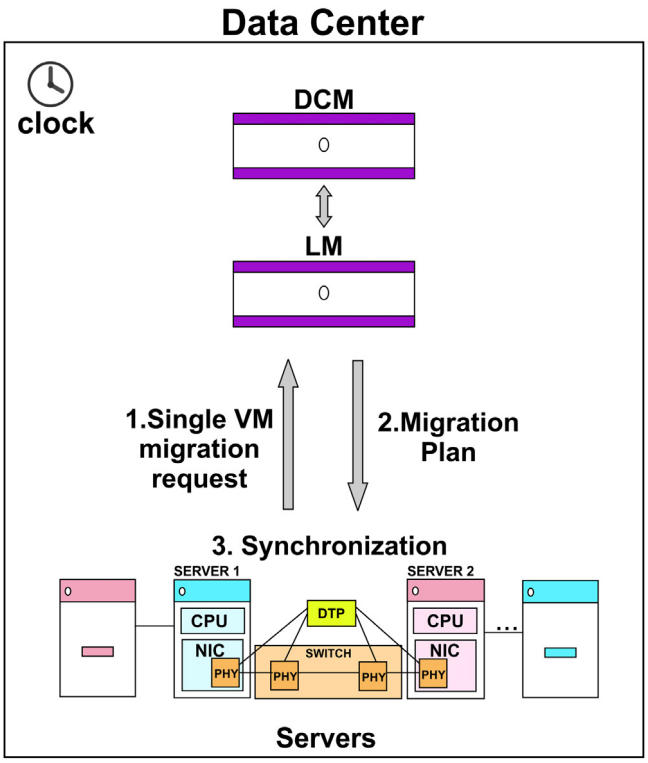

Fig. 2. Intra-data center scenario.

2. Local Manager (LM): It receives the VM migration request from the servers and then assigns the VM to the destination server.

To handle the increasing demand for low latency cloud services, it is vital to manage the dynamic allocation and migrate the VM workload either within the data center or across the data center. The decision-making for on-demand live migrations is dependent on several critical factors such as network availability, PM capacity, user mobility, rate of SLA violations and migration cost. Thus, the steps for synchronization and migration can be categorized in two scenarios:

1. Inter-data center scenario: As shown in Fig. 1, the VM migration request is sent by the server to LM and LM forwards this request to the DCM along with the state

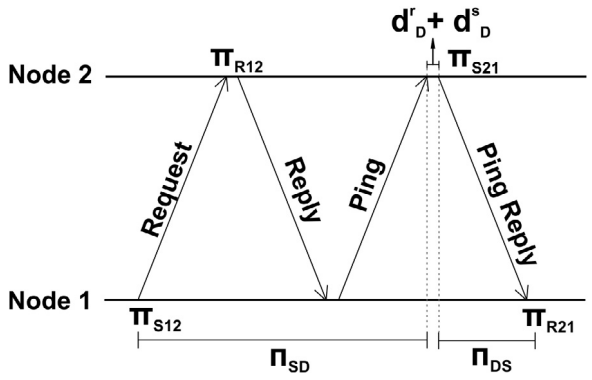

Fig. 3. Working between two levels.

information of the server. Then, a wireless PTP protocol is used for synchronization between Data Center 1 (DC1) and Data Center 2 (DC2). Finally, live VM migration takes place between two data centers.

2. Intra data center scenario: As represented in Fig. 2, the overloaded/ under-loaded utilization of PMs is detected by the hypervisor which continuously monitors resource uses of its assigned PMs. Then for mitigation of overloaded/ under-loaded PMs, migration request is sent to LM, which decides the server on which the VM must be migrated. Finally, synchronization between the servers takes place via enhanced DTP protocol followed by the live VM migration within the data center.

3. Synchronizing clocks at cloud data centers

In this section, we analyze and derive the generalized equations for clock synchronization among the data centers. To address the synchronization issues during live VM migration in inter and intra-data center scenarios, we assume all the network devices and PMs as nodes of the network.

3.1. Intra-data center synchronization

To compute round-trip time, clock offset error and convergence time for various levels in intra-data center synchronization scenario, we assume all the network devices and PMs as nodes for both asymmetric as well as symmetric networks.

Round Trip Time (RTT). The *round trip time* can be expressed as:

$$\kappa = Sending\_Time + Delay + Receiving\_Time \tag{1}$$

1. *Asymmetric Network:*
   A network in which the nodes transmit and receive packets at different rates is called an asymmetric network. For calculating the RTT in this case, we assume different packet transmission times and delays between a source node, routers, and destination node.
   First, we assume that there is a direct connection between the source node and the destination node. There will be two nodes at two levels as shown in Fig. 3. The packet will be sent from the node at level one, L_1, to the node at level two, L_2. We assume $Sending\_Time = \Pi_{SD}$, $Delay = \Lambda$ and $Receiving\_Time = \Pi_{DS}$. Here, Π_{SD} denotes the sending time between the source node at L_1 and the destination node at L_2. The delay can be written as:

   $$\Lambda = d^r_D + d^s_D \tag{2}$$

   Since we are calculating the delays for an asymmetric network, we express the total delay in terms of sending delay and receiving delay. In this case, d^s_D denotes the delay at the destination node when it receives it from the source node and d^r_D denotes the delay at the destination node when it sends it back to the source node. The total RTT can be expressed as:

   $$\kappa = \Pi_{SD} + \Lambda + \Pi_{DS}. \tag{3}$$

   $$\kappa = \Pi_{SD} + d^s_D + d^r_D + \Pi_{DS} \tag{4}$$

   Taking a more elaborated scenario where we consider a router between a source node and a destination node. There will be a total of three levels as shown in Fig. 4. The sending time between L_1 and L_2, and L_2 and L_3 can be expressed as:

   $$Sending\_Time = \Pi_{SR} + \Pi_{RD} \tag{5}$$

   The sending time, Π_{SR}, is elapsed between the source node at L_1 and the router at L_2. Π_{RD} is the sending time between L_2 and L_3. Similarly, we can write the receiving time as:

   $$Receiving\_Time = \Pi_{DR} + \Pi_{RS} \tag{6}$$

   Since it is an asymmetrical network, the delay during sending and receiving at the source node, router and destination node will be different. We can thus express the total delay as:

   $$\Lambda = Sending\_Delay + Receiving\_Delay \tag{7}$$

   $$Sending\_Delay = d^s_{R1} + d^s_D \tag{8}$$

   $$Receiving\_Delay = d^r_{R1} + d^r_D \tag{9}$$

   $$\Lambda = d^s_{R1} + d^s_D + d^r_{R1} + d^r_D \tag{10}$$

   We can generalize the delay in case of n levels which can be expressed as:

   $$\Lambda = d^s_{R1} + d^s_{R2} + \cdots + d^s_{R(n-2)} + d^s_D + d^r_{R1} + d^r_{R2} + \cdots + d^r_{R(n-2)} + d^r_D \tag{11}$$

   Similarly, we can generalize the equations for sending time and receiving time with,

   $$\Pi_{SD} = \Pi_{SR1} + \Pi_{R1R2} + \Pi_{R1R3} + \cdots + \Pi_{R(n\text{-}2)D} \tag{12}$$

   $$\Pi_{DS} = \Pi_{DR(n\text{-}2)} + \Pi_{R(n\text{-}2)R(n\text{-}3)} + \cdots + \Pi_{R1S} \tag{13}$$

   We can now express the generalized equation for RTT as:

   $$\kappa = \Pi_{SD} + \Lambda + \Pi_{DS} \tag{14}$$

2. *Symmetric Network:* In a symmetric network, the rates of transmission and receiving packets are equal. To calculate RTT in this case, we assume the same transmission time and delays while sending and receiving packets.
   In the case where the source node is directly connected to the destination node, we can write the RTT by denoting $Sending\_Time = \Pi_{SD}$, $Delay = \Lambda$ and $Receiving\_Time = \Pi_{DS}$. Since it is a symmetric network, $Sending\_Time = Receiving\_Time$ which means $\Pi_{SD} = \Pi_{DS}$ The delay can be obtained from Eq. (2), and being a symmetrical network, $d^r_D + d^s_D$. We can thus denote $\Pi_{SD} = \Pi_{DS} = \Pi$ and $d = 2d^s_D$ as both the delays are equal. We can thus write RTT between any two levels for symmetrical networks by:

   $$\kappa = 2\Pi + d \tag{15}$$

   When we take the case where there is a router present between the source node and the destination node. We can

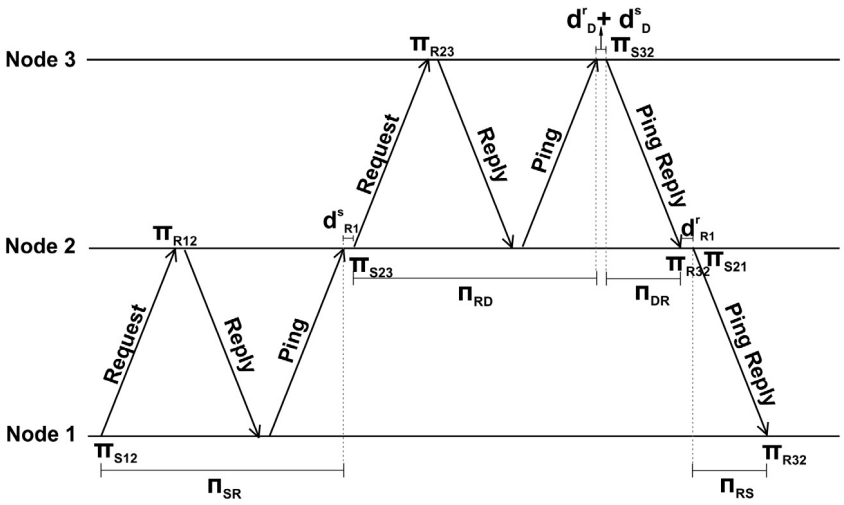

Fig. 4. Working between three levels.

express the sending time and the receiving time by using Eqs. (6) and (7). The delay can be written as:

$$Delay = 3\Lambda \tag{16}$$

This is the total delay in this case because the first delay will occur at L2, the second delay will occur at L3 and the third delay again at L2. We can thus write the RTT in this case as:

$$\kappa = \Pi_{\mathrm{SD}} + 3\Lambda + \Pi_{\mathrm{DS}} \tag{17}$$

Generalizing the equations for sending time, receiving time and the delay when there are n levels and hence $(n-2)$ routers:

$$Sending\_Time(\Pi_{\mathrm{SD}}) = \Pi_{\mathrm{SR}} + (n-3)\left\{\Pi_{\mathrm{RR}}\right\} + \Pi_{\mathrm{RD}} \tag{18}$$

$$Delay(\Lambda) = (2n-3)d \tag{19}$$

$$Receiving\_Time(\Pi_{\mathrm{DS}}) = \Pi_{\mathrm{DR}} + (n-3)\left\{\Pi_{\mathrm{RR}}\right\} + \Pi_{\mathrm{RS}} \tag{20}$$

Since it is a symmetrical network which means sending time will be equal to receiving time, we can write RTT as:

$$\kappa = 2\Pi + (2n-3)d \tag{21}$$

Offset. The following equation shows the *offset* for packet sent directly from source to destination:

$$\theta = \frac{\pi_{\mathrm{s}21} + \pi_{\mathrm{r}12}}{2} - \frac{\pi_{\mathrm{s}12} + \pi_{\mathrm{r}21}}{2} \tag{22}$$

Here, π_{s21} denotes the sending timestamp from the node at L_2 to the node at L_1, π_{r12} denotes the timestamp at which node at L_2 received packet from node at L_1. Similarly, π_{s12} denotes the sending timestamp of node at L_1 and π_{r21} denotes the receiving timestamp of node at L_1. For asymmetric networks, we take link delays into account. In that case the offset will become:

$$\theta = \frac{\pi_{\mathrm{s}21} + \pi_{\mathrm{r}12}}{2} - \frac{\pi_{\mathrm{s}12} + \pi_{\mathrm{r}21}}{2} - \frac{t_{12} - t_{21}}{2} \tag{23}$$

Here t_{12} is the time elapsed between sending of packet at time π_{s12} and receiving of packet at time π_{r12}. Similarly, t_{21} is the time elapsed between sending of packet at time π_{s21} and receiving of packet at time π_{r21}. The offset for n levels can be calculated as:

$$\begin{aligned}\theta = &\left[\frac{\pi_{\mathrm{s}21} + \pi_{\mathrm{r}12}}{2} - \frac{\pi_{\mathrm{s}12} + \pi_{\mathrm{r}21}}{2} - \frac{t_{12} - t_{21}}{2}\right] + \\ &\left[\frac{\pi_{\mathrm{s}32} + \pi_{\mathrm{r}23}}{2} - \frac{\pi_{\mathrm{s}23} + \pi_{\mathrm{r}32}}{2} - \frac{t_{23} - t_{32}}{2}\right] + \\ \ldots + &\left[\frac{\pi_{\mathrm{sn(n-1)}} + \pi_{\mathrm{r(n-1)n}}}{2} - \frac{\pi_{\mathrm{s(n-1)n}} + \pi_{\mathrm{rn(n-1)}}}{2} \right. \\ &\left. - \frac{t_{(n-1)n} - t_{n(n-1)}}{2}\right]\end{aligned} \tag{24}$$

Clock drift. Let us assume that the time at which the packet is sent from the slave node to the head node is π_s. The time at which this packet is received at the head node is π_r. The time taken between the subsequent sending and receiving of the packet is equal to the propagation delay which is represented by Π_p. Now, let Π_i be the value that is sent in the packet. This value is the timestamp of the sending time at the slave node. The calculation of clock drift between the clocks give rise to either of the three cases: (i) The two clocks are synchronized, i.e, there is no clock drift, (ii) The two clocks are not synchronized and the clock at the slave node lags behind the clock at the head node, (iii) The two clocks are not synchronized and the clock at the head node lags behind the clock at the slave node.

For the first case, since there is no clock drift:

$$\pi_{\mathrm{s}} = \pi_{\mathrm{r}} - \Pi_{\mathrm{p}} \tag{25}$$

For the second case, since the clock at the head node is faster:

$$\left\{\pi_{\mathrm{r}} - \Pi_{\mathrm{p}}\right\} - \pi_{\mathrm{r}} > 0 \tag{26}$$

For the third case, since the clock at the slave node is faster:

$$\left\{\pi_{\mathrm{r}} - \Pi_{\mathrm{p}}\right\} - \pi_{\mathrm{s}} < 0 \tag{27}$$

Convergence time. To achieve synchronization between any two nodes, special packets, called sync packets, are sent from one node to another. The time it takes for the sync packet to travel from one node to another can be expressed as:

$$ConvergenceTime = \frac{\rho\left\{\kappa\right\}}{2} \tag{28}$$

Here, ρ is the number of packets exchanged between two nodes to exchange messages. For calculating the convergence time between any two data centers with hops, we calculate the time taken for synchronization.

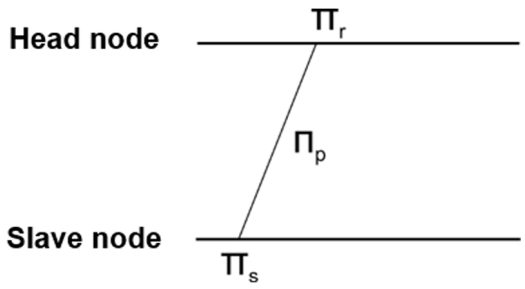

Fig. 5. Convergence Time.

As shown in Fig. 5, there are two data centers, D_1 and D_2, which can have different or the same network architectures. Let the total time taken to synchronize them be Π_{D1D2}, which can be broken down into-

$$(\Pi_{D1D2}) = (\Pi_{D1C}) + (\Pi_{D2C}) \tag{29}$$

Here, Π_{D1C} and Π_{D2C} is the time taken for synchronization between the data center and the cloud. The time between the initiation of any two requests is assumed to be the same for all cases and is represented as p. Thus, the total convergence time in this case is-

$$\begin{aligned}(\Pi_{D1D2}) = (\Pi_{Sync1}) + (\Pi_{DReq1}) + (\Pi_{DReply1}) + \\ (\Pi_{Sync2}) + (\Pi_{DReq2}) + (\Pi_{DReply2}) + 4p\end{aligned} \tag{30}$$

For heterogeneous data center network architectures, we can calculate the convergence time using the following formula-

$$(\Pi_{D1Dn}) = (\Pi_{D1C}) + (\Pi_{D2C}) + np \tag{31}$$

For homogeneous data center network architectures,

$$(\Pi_{D1}) = (\Pi_{D2}) \tag{32}$$

$$(\Pi_{D1D2}) = 2(\Pi_{D1}) \tag{33}$$

The generalized formula for calculating the convergence time in case of homogeneous network architectures-

$$(\Pi_{D1Dn}) = 2(\Pi_{D1C}) + np \tag{34}$$

Here, n is the number of hops between any two data centers.

3.2. Inter-data center synchronization

In this section, we propose a wireless PTP based method for inter-data center synchronization assuming that each data center is connected to another data center as shown in Fig. 6. Since there is a direct connection, messages can be sent between any two data centers directly instead of forwarding. In such a setup of data centers, PTP can be implemented such that first a data center sends a SYNC and Follow_up message to another data center. The second data center then sends a request and the first data center sends a reply. Along with sending a request message to the first data center, the second data center broadcasts the same message to all the next adjacent data centers as well. This way, by broadcasting the request messages, a SYNC message is sent to all the adjacently connected data centers.

By broadcasting the request message, the total time for synchronization of all data center reduces and the total number of message packets required is also decreased. This can be represented mathematically as follows:

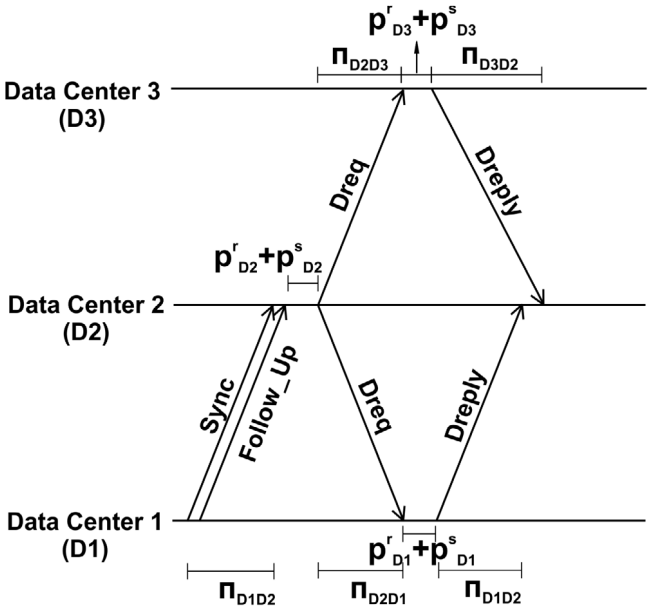

Fig. 6. Timing diagram for inter-data center synchronization.

Round trip time. For inter-data center scenarios, the *round trip time* can be expressed as follows:

1. *Asymmetric Network:*

$$\kappa = \begin{cases} 2\Pi_{D1D2} + \Pi_{D2D1} + p^s_{D2} + p^r_{D2} + \\ p^s_{D1} + p^r_{D1} & \text{if } n = 1 \\ 2\Pi_{D1D2} + \Pi_{D2D1} + \Pi_{D2D3} + \Pi_{D3D2} + \\ \Pi_{D3D4} + \Pi_{D4D3} + \cdots + \Pi_{Dn-1Dn} + \\ \Pi_{DnDn-1} + (p^s_{D1} + p^r_{D1} + p^s_{D2} + p^r_{D2} \\ + \cdots + p^s_{Dn} + p^r_{Dn}) & \text{if } n > 1 \end{cases} \tag{35}$$

   Here, Π_{D1D2} denotes the time required to send a packet from data center 1 to data center 2. Similarly, Π_{Dn-1Dn} represents the time it takes to send a packet from data center $n-1$ to data center n. p^s_{D1} and p^r_{D1} denote the sending delay and receiving delay respectively between two requests at data center 1. Similarly, the time between the initiation of two requests at other data centers is written.
2. *Symmetric Network:* In this case we consider that the time between the initiation of two requests is the same represented by p.

$$\kappa = \begin{cases} 2\Pi_{D1D2} + \Pi_{D2D1} + 2p & \text{if } n = 1 \\ 2\Pi_{D1D2} + \Pi_{D2D1} + 2p + \Pi_{D2D3} + \\ \Pi_{D3D2} + \Pi_{D3D4} + \Pi_{D4D3} + \cdots + \\ \Pi_{Dn-1Dn} + \Pi_{DnDn-1} + np & \text{if } n > 1 \end{cases} \tag{36}$$

Number of packets. The *number of packets*, ρ can be reduced in this scenario. When first data center sends the Sync and Follow_Up message to the second data center, we count it as two packets, namely $2\rho_0$. For the request, DReq, and reply, DReply, between the first two data centers, we can denote the packets as $2\rho_1$. Hence, the number of packets exchanged between the first two data centers are:

$$Packets = 2\rho_0 + 2\rho_1 \tag{37}$$

Similarly, the number of packets exchanged between the second and third data center will be the request and reply packets which can be represented as $2\rho_2$. Thus, the total packets

exchanged between any three data centers can be calculated as:

$$Packets = 2\rho_0 + 2\rho_1 + 2\rho_2 \tag{38}$$

Generalizing this equation for n data centers, the total number of packets that will be exchanged can be expressed as:

$$Packets = 2\rho_0 + 2\rho_1 + 2\rho_2 + \cdots + 2\rho_n \tag{39}$$

$$Packets = 2\{\rho_0 + \rho_1 + \cdots + \rho_n\} \tag{40}$$

$$Packets = 2\sum_{i=1}^{n} \rho_i \tag{41}$$

Convergence time. The time it takes to exchange synchronization packets among nodes is shown by convergence time. For achieving this, Sync, Follow_Up, Dreq, and Dreply messages are exchanged between the nodes. We consider two cases:

1. *Asymmetric Network:*
   In asymmetric network, the convergence time for two nodes is given by:

   $$ConvergenceTime = (\frac{3\kappa_{12}}{2} + p^s_{D1} + p^r_{D1} + p^s_{D2} + p^r_{D2}) \tag{42}$$

   Here, κ_{12} denotes the round trip time for the exchange of messages between 2 nodes and p^s_{D1} and p^r_{D2} represent the sending delay and receiving delay between two messages. Similarly, the delay can be calculated for other data centers. On generalizing the formula for n nodes we obtain:

   $$\begin{aligned} ConvergenceTime = (\frac{3\kappa_{12}}{2} + p^s_{D1} + p^r_{D1} + p^s_{D2} + \\ p^r_{D2}) + (\frac{2\kappa_{23}}{2} + p^s_{D3} + p^r_{D3}) + (\frac{2\kappa_{34}}{2} + p^s_{D4} + p^r_{D4}) \\ + \cdots + (\frac{2\kappa_{(n-1)n}}{2} + p^s_{Dn} + p^r_{Dn}) \end{aligned} \tag{43}$$

2. *Symmetric Network:*
   In the symmetric network, the time delay between the initiation of two messages is the same, i.e., p. Now the convergence time for two nodes is given by:

   $$ConvergenceTime = (\frac{3\kappa_{12}}{2} + 2p) \tag{44}$$

   Here, κ_{12} denotes the round trip time for the exchange of messages between 2 nodes. On generalizing the formula for n nodes we get,

   $$\begin{aligned} ConvergenceTime = (\frac{3\kappa_{12}}{2} + 2p) + (\frac{2\kappa_{23}}{2} + p) + \\ (\frac{2\kappa_{34}}{2} + p) + \cdots + (\frac{2\kappa_{(n-1)n}}{2} + p) \end{aligned} \tag{45}$$

   $$\begin{aligned} ConvergenceTime = (\frac{3\kappa_{12}}{2}) + (\kappa_{23} + \kappa_{34} + \cdots + \\ \kappa_{(n-1)n}) + (n+1)p \end{aligned} \tag{46}$$

   For homogeneous data centers, we assume round trip time between the data centers is same i.e. equal to κ. Hence we obtain:

   $$ConvergenceTime = (n + \frac{1}{2})\kappa + (n+1)p \tag{47}$$

4. Proposed algorithms

4.1. Algorithm for intra-data center synchronization

In Algorithm 1, we present an enhanced DTP algorithm [10] for network devices. For network ports, the inputs are the list of the local timestamps of all the ports, Ω and the number of the ports, η. We assume the number of ports to be η. For each of these ports, there are local timestamps, and we compare each of these local timestamps to select the maximum value. The maximum value is assigned as the global timestamp π_g. All the local timestamps are then updated according to the global timestamp. The clocks of the ports of a device are synchronized. Phase 3 of this algorithm deals with live VM migration. First, the number of pages that are to be migrated from one host to another is estimated. Next, parameters such as page size, P, and we assume the volume that is to be migrated from host A, H_A, to host B, H_B, as Q. The number of pages, n, to be transferred is then calculated as the volume to be transferred divided by the size of one page. We assume a variable, δ, which behaves like a Boolean variable. It denotes whether the page is dirty or not. Initially, this variable is initialized to zero for all pages. Now, α is taken as the time taken for a page to get transferred and the total time taken for all pages to is T_α. After time T_α, all the pages are checked whether they are dirty pages or not. For this, a log is maintained that contains the most recent update time stamp of the page and the migration time of the page. If most-recent update timestamp, π^u, is higher than the migration timestamp, π^m, then it means that the page is dirty. Thus, δ is set to value 1. Once all the pages are checked whether they are dirty or not, the dirty ones are migrated to host B again. This process is repeated until all the pages to be transferred are migrated, and no dirty page is left behind.

Algorithm 1 Enhanced DTP Algorithm for Intra Data Center Synchronization

Input: $D_A, D_B, \pi_s^1, \pi_r^2, \Omega, \eta, Q, P, \delta, \Gamma$

Phase 1: Enhanced DTP phase for network ports
Take the local timestamps of all η ports.
Select the maximum of these local timestamps and set it as Global Timestamp, π_g.
$\pi_g \leftarrow max(\Omega[1], \Omega[2], ..., \Omega[\eta])$
Update the local timestamps of all the ports and equate them to the global timestamp.
for all $1 \le \xi \le \eta$ **do** $\Omega[\xi] \leftarrow \pi_g$
Phase 2: Live VM migration phase
Estimate the number of pages to be transferred.
Let the size of a page be P.
Let the volume for transfer from host A, H_A, to host B, H_B, in a VM be Q.
Number of pages for transfer, $n = Q/P$
for all $1 \le \xi \le n$ **do** Set $\delta = 0$
Let α be the time taken for transfer of one page. Time taken for transfer of n pages, $T_\alpha = \sum_{n=1}^{n} \alpha$
After time T_α,
for all $1 \le \xi \le n$ **do**
    Check if page is dirty.
    if $\pi^u > \pi^m$ Set $\delta = 1$ **then** ▷ Comparing most recent update timestamp with migration timestamp
    else $\delta = 0$
Let the page dirtying rate be Γ.
if $\Gamma > \frac{n}{T_\alpha}$ **then** Stop and copy all pages to H_B
    ▷ Comparing page dirtying rate with page transfer rate
else Continue from step 12 until all pages are transferred.

4.2. Algorithm for inter-data center synchronization

In Algorithm 2, we present the inter-data center synchronization, where the nodes are separated by a large distance, and therefore, it is challenging to synchronize all the nodes at once.

Algorithm 2 Algorithm for Inter Data Center Synchronization

Input: $\pi_G, \omega^r, \omega^p, \upsilon, \omega^r, \omega^s, \iota, \Phi$

Phase 1: Synchronize the GPS node with the ν Cluster Heads
for all $1 \leq \zeta \leq \iota$ **do**
  if $\omega^r_\zeta - \omega^p_\zeta = \pi_G$ **then** ▷ Already synchronized to GPS
  else if $\omega^r_\zeta - \omega^p_\zeta - \pi_G > 0$ **then** ▷ Cluster Head node is slower
    Add $\omega^r_\zeta - \omega^p_\zeta - \pi_G$ *to the clock of Cluster Head node*
  else ▷ Cluster Head node is faster
    Subtract $\omega^r_\zeta - \omega^p_\zeta - \pi_G$ *from the clock of Cluster Head node*
Phase 2: Synchronize the Cluster Heads and Cluster Nodes
for all $1 \leq \zeta \leq \iota$ **do**
  for all $1 \leq \beta \leq \upsilon$ **do**
    Broadcast message from the cluster nodes to the Head Node
    Calculate RTT between the Cluster node and the Cluster Head
    $\kappa_{\zeta,\beta} = (\Pi_{SR} + (n-2)\{\Pi_{RR}\} + \Pi_{RD}) + ((2n-1)\lambda) + (\Pi_{DR} + (n-2)\{T'_{RR}\} + T_{RS})$
    Calculate one way delay as: $\vartheta_{\zeta,\beta} = \frac{\kappa_{\zeta,\beta}}{2}$
    Set minimum one-way delay, ϵ
  Synchronize the ζ *cluster node with the* β *cluster head with which it has minimum* ω
  if $\omega^r_\zeta - \epsilon = \omega^s_\beta$ **then** ▷ Already synchronized to Cluster Head node
  else if $\omega^r_\zeta - \epsilon - \omega^s_\beta > 0$ **then** ▷ Cluster node is slower
    Add $\omega^r_\zeta - \epsilon - \omega^s_\beta$ *to the clock of Cluster node*
  else ▷ Cluster node is faster
    Subtract $\omega^r_\zeta - \epsilon - \omega^s_\beta$ *from the clock of Cluster node*
Phase 3: Applying wireless PTP
for all $1 \leq \zeta \leq \iota$ **do**
  Initialize all nodes from $\zeta = 1 \cdots \iota$
  Establish link between them.
  For communication between any two nodes
  Assign source $=$ *node*$[\alpha]$*, destination* $=$ *node*$[\sigma]$*, where* $\alpha \neq \sigma$
  Set master node $\equiv$ *source node.*
  Let the slave node be node$[\alpha + 1]$*.*
  Master node sends π^1 *to slave node.*
  Master node sends follow-up message with π^1. At slave node, let Φ be a time variable and $\Phi = 0$ initially.
  When π^1 *reaches slave node, record time in* π^2 and set $\Phi = 1$.
  Slave node sends π^3 *(time at which it sends the message) back to master node.*
  Master node stores receiving time π^4 *and forwards to slave.*
  if Slave node $=$ destination **then** Stop
  else Make node$[\alpha + 1]$ the master node and repeat from step 31

Therefore, clusters are created, and some nodes are appointed as the head nodes of the clusters. These head nodes then synchronize their clocks with the GPS node. We denote the timestamp of the GPS node by π_G, list of receiving timestamps by ω^r, list of propagation time by ω^p, the number of cluster heads by υ, the list of the sending timestamps by ω^s, number of cluster head nodes by υ and the remaining nodes by ι. Once the cluster nodes synchronize their clocks with the cluster head node, wireless PTP algorithm is implemented. For the implementation of the wireless PTP algorithm, all ι nodes are initialized to be registered in the network, and a link is established between them. The source node $[\alpha]$ is assigned as the master node and the next node i.e. destination, with $\sigma = \alpha + 1$ which acts as the slave node.

4.3. Illustrative example

We assume a cloud data center with three heterogeneous physical machines (PMs) denoted as $PM = \{PM_1, PM_2, PM_3\}$. The capacity of each PM is characterized with two type of resources such as CPU and memory. Currently six VMs, $VM = \{VM_1, VM_2, VM_3, VM_4, VM_5, VM_6\}$ are allocated to PMs as shown in Fig. 7(a). The total capacity of all PMs is represented through a capacity vector for instance $Cap_{PM_1} = (16, 24)$, $Cap_{PM_2} = (8, 10)$ and $Cap_{PM_3} = (15, 14)$. Here $Cap_{PM_1} = (16, 24)$ means the CPU capacity of PM_1 is 16 GHz and memory capacity is 24 GB. We consider CPU threshold $Th = 0.9$ for this example. Let us say the total capacity of each VM is represented through capacity vectors $Cap_{VM_1} = (2, 2)$, $Cap_{VM_2} = (2, 8)$, $Cap_{VM_3} = (2, 2)$, $Cap_{VM_4} = (4, 4)$, $Cap_{VM_5} = (6, 8)$ and $Cap_{VM_6} = (9, 8)$. Here $Cap_{VM_1} = (2, 2)$ denotes that the CPU consumption of VM_1 is 2 GHz and memory consumption is 2 GB. Let us consider the current resource usage demand of VMs is denoted as, $RU_{VM_1} = (1, 1.4)$, $RU_{VM_2} = (1, 4)$, $RU_{VM_3} = (1, 1)$, $RU_{VM_4} = (1, 1)$, $RU_{VM_5} = (1, 1)$, and $RU_{VM_6} = (1, 1)$. Here $RU_{VM_1} = (1, 1.4)$ shows that VM_1 demands 1 CPU capacity of 2 GHz and 1.4 memory of 2 GB. Fig. 7(a) shows the initial stage of VM allocation. Based on the initial placement, the current utilization of PMs are calculated as: $CU_{PM_1} = (3, 6.4)$, $CU_{PM_2} = (1, 1)$, and $CU_{PM_3} = (2, 2)$. Here $CU_{PM_1} = (3, 6.4)$ means PM_1 accommodate 3 VMs namely VM_1, VM_2 and VM_3 with demanded capacity $(1, 1.4)$, $(1, 4)$ and $(1, 1)$ respectively. Hence the total utilization of PM_1 is calculated via sum of VMs demand vectors. If we estimate the load of each PM through below formula:

$$L_{PM_1} = \frac{CU^{CPU}_{PM1}}{Cap^{CPU}_{PM1}} + \frac{CU^{MEM}_{PM1}}{Cap^{MEM}_{PM1}} \tag{48}$$

$$= \frac{3}{16} + \frac{6.4}{24} = 0.4541 \tag{49}$$

In similar fashion, we find the load of other PMs as: $L_{PM_2} = 0.225$, $L_{PM_3} = 0.2761$. Now we consider the two different cases of on demand migration:

- Case I: On demand migration due to energy saving:
  As shown in Fig. 7(a), we do not find any of the PMs to be overloaded based on the current utilization. Here for energy saving, we can migrate VM_6 from PM_2 to any of the PMs. So that the PM_2 can be switched to the sleep mode. For migration of VM_6, we select the highly-loaded PM i.e., PM_1 and check the below condition for sufficient capacity:

  $$CU_{PM_1} + RU_{VM_6} \leq TH \times Cap_{PM_1} \tag{50}$$

  $$(3, 6.4) + (1, 1) \leq 0.9 \times (16, 24) \tag{51}$$

  Here, TH is an upper threshold value, which is assumed to be 0.9. As the condition is satisfied, we can migrate the VM_6 from PM_2 to PM_1. For this on demand migration, we first synchronize both the PMs (Fig. 7(b)) then migrate VM_6 from PM_2 to PM_1. Now PM_2 does not host any of the VMs so we can switch it to the sleep mode for energy saving as shown in Fig. 7(c).
- Case II: On demand migration due to threshold violation:
  In this case, we assume resource utilization demand of VMs as, $RU_{VM_1} = (2, 2)$, $RU_{VM_2} = (2, 8)$, $RU_{VM_3} = (2, 2)$, $RU_{VM_4} = (1, 1)$, $RU_{VM_5} = (1, 1)$, and $RU_{VM_6} = (9, 8)$ as shown in Fig. 8(a). We calculate the load level of active PMs $L_{PM_1} = 1.7705$, and $L_{PM_3} = 0.2761$. We observe that the PM_1 is

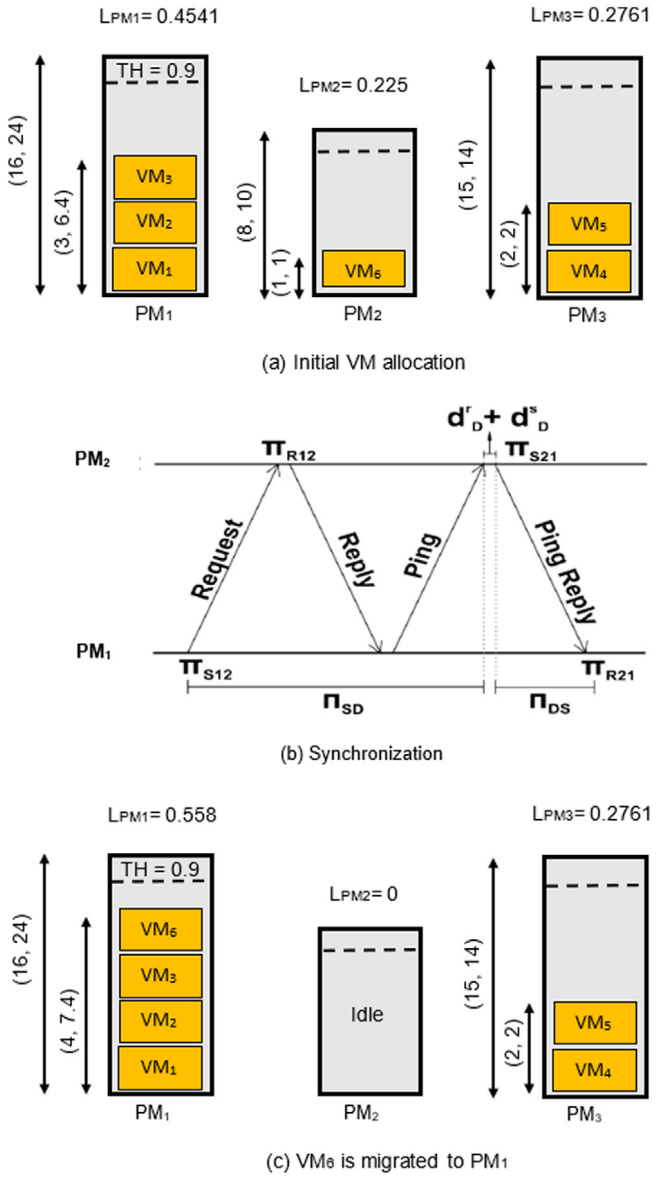

Fig. 7. Case-I: On demand migration due to energy saving.

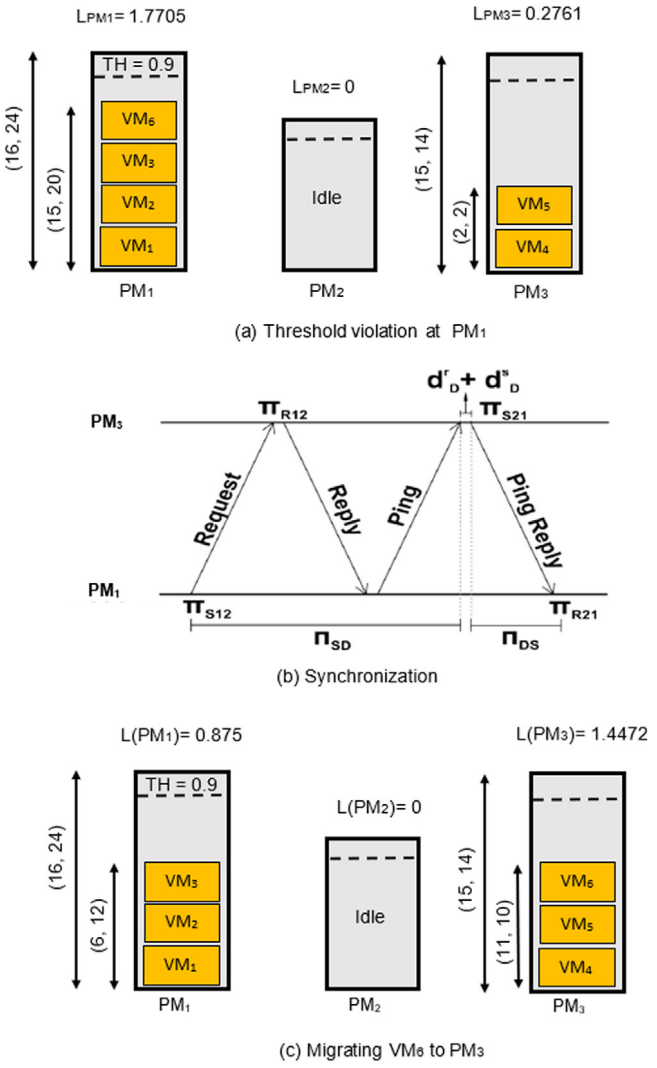

Fig. 8. Case-II: On demand migration due to threshold violation.

overloaded based on the current CPU utilization and does not hold the following equation.

$$CU_{PM_1} \leq TH \times Cap_{PM_1} \tag{52}$$

$$(15, 20) > 0.9 \times (16, 24), (15, 20) > (14.4, 21.6) \tag{53}$$

Therefore, we need to migrate any of the VMs from PM_1. So that the SLA (service level agreement) violation can be avoided at next time instant. For migration, we first select the highly-loaded VM by measuring the load level of all the VMs through following equation:

$$L_{VM_1} = \frac{RU^{CPU}_{VM_1}}{Cap^{CPU}_{VM_1}} + \frac{RU^{MEM}_{VM_1}}{Cap^{MEM}_{VM_1}} \tag{54}$$

$$= \frac{2}{2} + \frac{2}{2} = 2 \tag{55}$$

$$L_{VM_2} = \frac{RU^{CPU}_{VM_2}}{Cap^{CPU}_{VM_2}} + \frac{RU^{MEM}_{VM_2}}{Cap^{MEM}_{VM_2}} \tag{56}$$

$$= \frac{2}{2} + \frac{2}{8} = 1.25 \tag{57}$$

$$L_{VM_3} = \frac{RU^{CPU}_{VM_3}}{Cap^{CPU}_{VM_3}} + \frac{RU^{MEM}_{VM_3}}{Cap^{MEM}_{VM_3}} \tag{58}$$

$$= \frac{2}{2} + \frac{2}{2} = 2 \tag{59}$$

$$L_{VM_6} = \frac{RU^{CPU}_{VM_6}}{Cap^{CPU}_{VM_6}} + \frac{RU^{MEM}_{VM_6}}{Cap^{MEM}_{VM_6}} \tag{60}$$

$$= \frac{9}{9} + \frac{8}{8} = 2 \tag{61}$$

Based on the load level, we can migrate any of the higher loaded VMs among VM_1, VM_3 and VM_6 to PM_3 and mitigate the threshold violation. Randomly, we pick VM_6 and synchronize both PM_1 and PM_3 (Fig. 8(b)). After synchronization, we migrate VM_6 to PM_3 as shown in Fig. 8(c). Now again we check the following condition:

$$CU_{PM_1} \leq TH \times Cap_{PM_1} \tag{62}$$

$$(6, 12) < 0.9 \times (16, 24) \tag{63}$$

So, we find that after the migration of VM_6, the PM_1 is not overloaded. Hence the threshold violation at PM_1 is mitigated.

5. Algorithmic analysis

Theorem 5.1. *The worst-case time complexity of Algorithm 1 is* $O(2n^2 + 2n + 2\eta + 1)$.

Proof. Algorithm 1 implements DTP in intra-data center networks. We assume that there are η ports and we select the maximum local timestamps amongst all the timestamps of each of these ports. For $\eta = 1$, the time complexity would be 1. For $\eta = 2$, the time complexity would be 2 as one timestamp will have to be compared with another and in the worst-case, the second local timestamp may be of maximum value. Similarly, for comparing η timestamps, the worst-case time complexity would be η. We write this worst-case time complexity as $O(\eta)$. After choosing the maximum of all local timestamps, we set the maximum value as the global timestamp. Next, we update the value of local timestamps by equating them with the value of global timestamp. If there are $\eta = 1$ ports, then it will be updated at once and the time complexity would be 2 because the timestamp will be updated once and before ending the loop, the condition for the number of timestamps will be checked again that will increment the time complexity by 1 unit. If there are $\eta = 2$ ports, the time complexity would be 3 because after updating the values of 2 local timestamps in a loop, the loop will run again to check the condition for the number of local timestamps, hence incrementing the time complexity by 1 unit. Similarly, using mathematical induction, for η local timestamps, the time complexity will be $O(\eta + 1)$.

In Phase 2 of the algorithm, we migrate the VMs from one host to another. In this process, we transfer n pages calculated as per the formula in the algorithm. For calculating the time taken for the total transfer of all the pages, we use a loop. If $n = 1$, the worst case complexity will be $O(2)$. If $n = 2$, the worst-case time complexity will be $O(3)$. The worst-case time complexity will be 1 unit extra in each case because, after every loop, the condition for the number of pages will be checked each time, even in the case where the condition is first violated, and the loop ends. Thus, the worst-case time complexity of this loop will be $O(n+1)$. After computing the total transfer time for n pages, we check if these pages are dirty. We do this using another loop that runs n times in which a condition for whether the page is dirty is checked. The worst-case time complexity for this loop will be $O(n + 1)$ as each of these n pages will be checked for the condition, and the loop will run one extra time while checking the false condition that ends the loop. Cumulatively, the worst-case time complexity of the algorithm is $O(\eta + \eta + 1 + n + 1 + n + 1)$ which can be simplified as $O(2\eta + 2n + 3)$. Next, we compare the page dirtying rate with the transfer rate of the pages. If the former is greater, then the process is halted, and all the pages are copied to host B, else the process of page transfer continues until all pages are transferred. Since there are n pages, in the worst-case scenario, this process can continue a maximum of n times. This means that the process of calculating total transfer time of n pages and checking if each page is dirty will take place n times in the worst-case, hence making the worst-case time complexity n times that of the initial time complexity. Thus, the worst-case time complexity will be $O(2\eta + 1 + 2n(n + 1))$ which is $O(2n^2 + 2n + 2\eta + 1)$. ■

Theorem 5.2. *The worst-case time complexity of Algorithm 2 is* $O[2n^2 + \iota * \upsilon + 2n + 2\iota + \zeta + 4]$.

Proof. In Phase 1 of Algorithm 2, we synchronize the GPS node with the ν cluster heads. For doing so, we first calculate the offset and adjust the clock to synchronize with the GPS node. Calculating the offset of ζ nodes, the worst-case time complexity comes out to be $O(\zeta + 1)$ since the loop will work ζ times plus the one time when the condition fails, and the loop ends. Inside the loop, the different conditions are checked whose worst-case complexity would be $O(1)$. Since the loop goes on for ζ times, the total worst-case complexity of the loop will remain $O(\zeta + 1)$.

In Phase 2 of the algorithm, we synchronize the cluster heads with the cluster nodes. Since there are ι and υ cluster heads and cluster nodes respectively, we use a nested for-loop for calculating the RTT, one-way delay and the offset between the cluster node and the cluster head. The worst-case time complexity of this nested loop would be $O(\iota^*\upsilon + 1)$.

In phase 3, PTP is implemented. All the ι nodes are initialized with increasing numbers from 1 to ι. For doing so, the loop will be used whose complexity would be $O(\iota + 1)$. Then a link is established between all nodes. We then assign a node as Source and another as Destination. The Source sends out a message to a node which is to be received by a node called the Destination. For this, the algorithm proceeds sequentially through every node before it reaches the Destination node. For this, the worst-case time complexity would be $O(\iota + 1)$. So the total worst-case time complexity would be $O[2^*(\iota + 1)]$.

In phase 4, VM migration takes in case of inter-data center migration. For this, the worst-case time complexity will be $O[2n(n + 1)]$.

This means that the total worst-case complexity of Algorithm 2 is $O[\zeta + 1 + \iota^*\upsilon + 1 + 2^*(\iota + 1) + 2n(n + 1)]$, which is equivalent to $O[2n^2 + \iota^*\upsilon + 2n + 2\iota + \zeta + 4]$. ■

6. Simulation analysis

6.1. Simulation setup

In our implementation of data center (DC) architectures and synchronization protocols, we have used Omnet++ 5.1. We have implemented enhanced DTP and wireless PTP for Fat-tree, BCube, VL2 and Three-Tier DC architectures using the INET framework. In Figs. 9 and 10, we have shown the examples of Bcube and VL2 data center architectures respectively. We assume all the network devices and PMs as nodes of the network ranging from 80–100.

Using these DC architectures as a basis, we have built intra and inter-data center networks for homogeneous and heterogeneous data center scenarios respectively. For inter-data center scenario, we have built networks implementing a combination of:

1. Fat-tree and BCube: An example of an inter-data center network for Fat-tree with 21 routers and 16 servers and BCube with 9 routers and 16 servers is shown in Fig. 11.
2. Three-tier and VL2: An example of three-tier data center architecture with 27 routers and 36 servers and VL2 with 15 routers and 16 servers is shown in Fig. 12.

In the implementation of the DC architectures, we have estimated the clock variance and the synchronization times after running the simulations on these networks. For the implementation of wireless PTP, we have used an algorithm that sends a sync message to the client, the client then replies with a request and then a reply is sent to the client. The timestamps for receiving and sending these messages are recorded and used for the calculation of convergence time, offset calculation, and RTT, etc. In these simulations, we have also performed experiments to observe their effect on mean synchronization times while varying connectivity radius for the values 0.95 units, 1 unit, 1.25 units, and 1.5 units.

6.2. Simulation results

1. Intra-Data Center Synchronization Experiment 1: Fig. 13 shows the synchronization time for all nodes of the four DC architectures. From this plot, it can be inferred that when the number

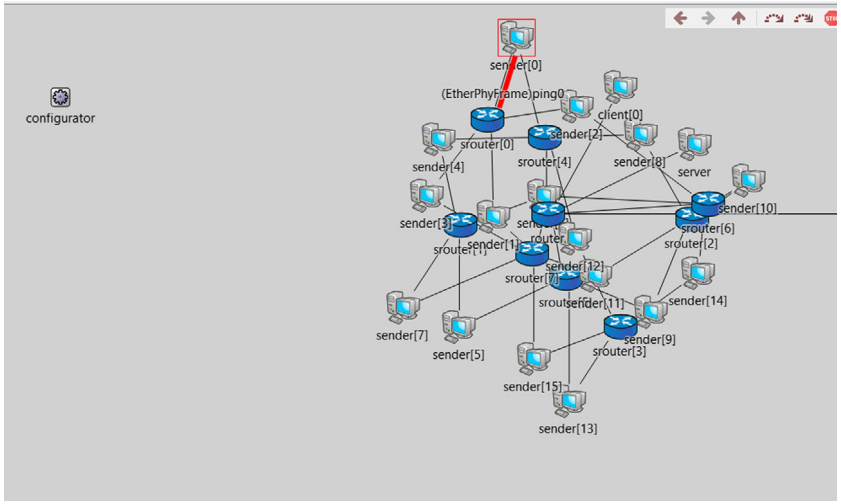

Fig. 9. Example of Bcube with 16 server and 9 routers.

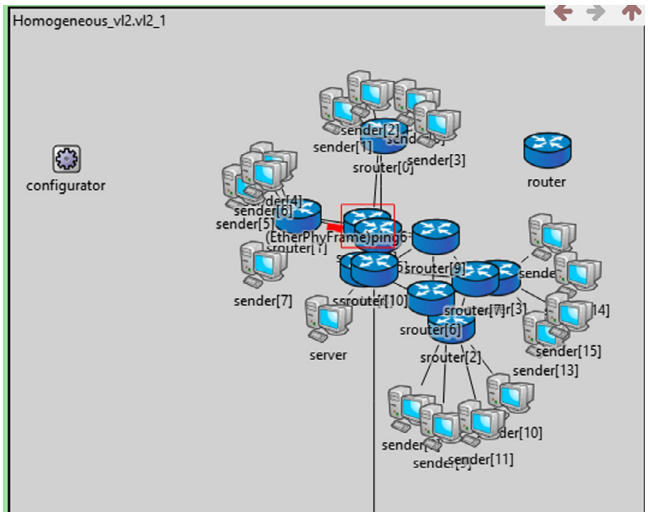

Fig. 10. Example of VL2 with 16 servers and 15 routers.

of nodes are **less (<41)**, **BCube** takes least time to synchronize and when the number of nodes are in the **range of 41–73**, **VL2** gives the best results. For nodes in **range 73–92, Three-tier** takes least time to synchronize. **Fat-tree** takes minimum time to synchronize if the number of nodes are **greater than 92**. The least time taken for synchronization by nodes in Fat-tree, VL2, and Three-tier is **3.8454% more** than the least time taken by the nodes in BCube. When the number of nodes are above 73, then the Fat-tree architecture gives the least time for synchronization of all nodes. The least time taken by the nodes of other DC architectures to synchronize is **12.4984% more** than the least time obtained for synchronization in Fat-tree architecture.

Table 1
Analysis of Variance in Clocks for Intra Data Center Architectures.

Simulation Time (in μs)	Data Center Architectures			
	Bcube	Fat-Tree	3 Tier	VL2
500000000	4.9939	6.3330	6.7245	5.7038
1000000000	4.1434	6.2220	6.0439	4.2360
1500000000	6.3040	5.0229	6.7225	4.9432
2000000000	4.2753	3.5812	5.5203	6.7504
2500000000	5.5590	5.5962	6.7399	6.1522
3000000000	5.4603	4.8749	5.6439	4.3909
3500000000	5.6687	6.1238	4.8377	5.7485
4000000000	6.5333	5.7808	6.0902	5.6907
4500000000	4.9753	5.8327	5.1636	5.8170
5000000000	6.5751	4.7849	5.1461	5.4394

Experiment 2: Fig. 14 and Table 1 show the clock variance for the four data center architectures. It is observed that for a total simulation time of 5000 s and the number of nodes in the range 80–100, Fat-tree has the maximum range of clock variance. Meanwhile, VL2 shows the least difference between its minimum and maximum synchronization time. The range of variance for VL2 is observed to be in between **0.37921 to 7.51227 s**.

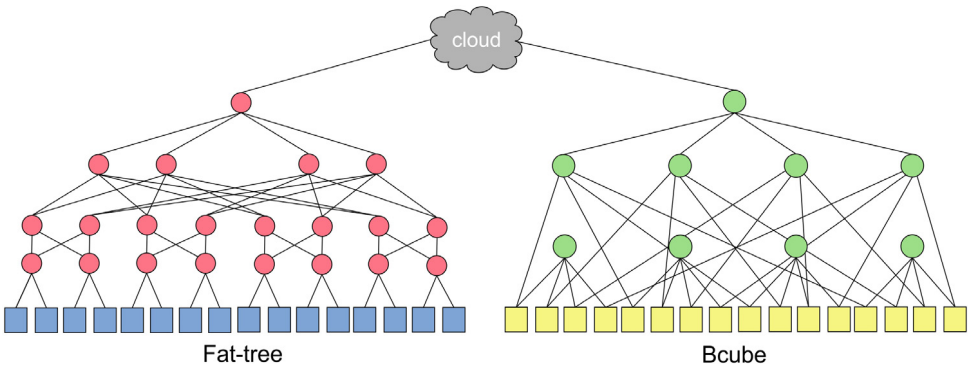

Fig. 11. Example of Inter-data center architecture for Fat-tree and Bcube.

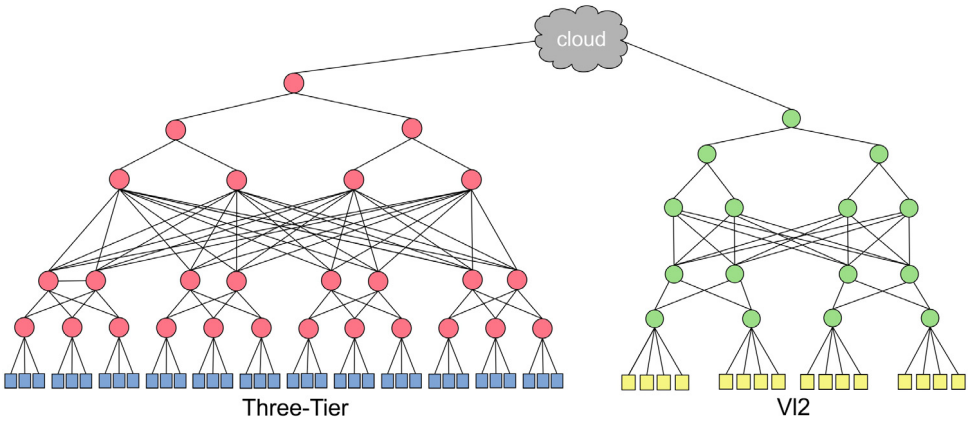

Fig. 12. Example of Inter-data center architecture for Three-tier and VL2.

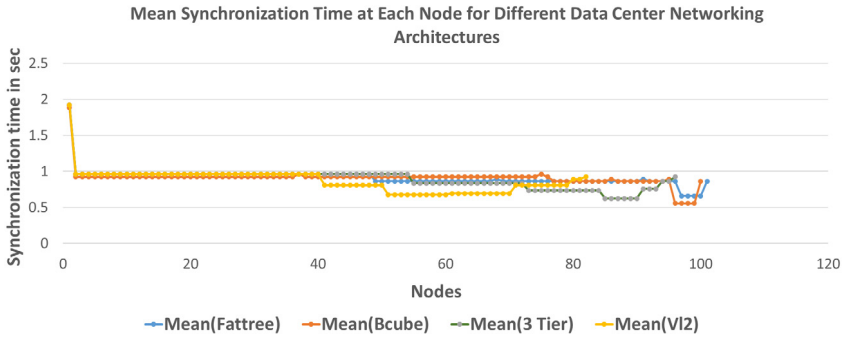

Fig. 13. Comparison of Synchronization Time for Intra Data Center architectures.

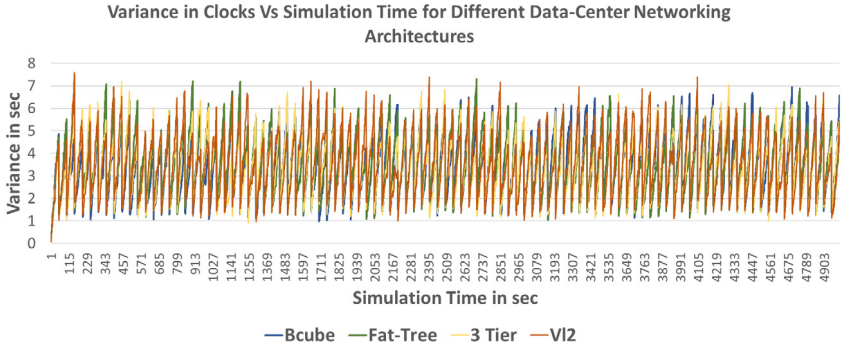

Fig. 14. Comparison for Variance in Clocks for Intra Data Center Architectures.

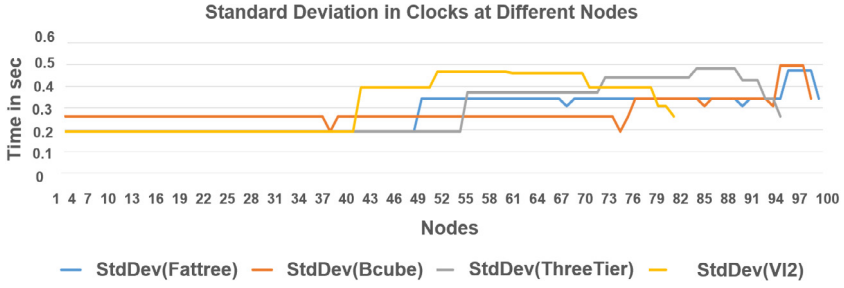

Fig. 15. Comparison for Standard Deviation in Clocks for Intra Data Center Architectures.

Experiment 3: Fig. 15 and Table 2 show the standard deviation in clocks for the four data center architectures. We can infer from this plot that, for the number of nodes **less (<40)**, BCube architecture has the maximum value of clock standard deviation and, all the other architectures, i.e., Fat-tree, VL2, and Three-Tier have the same value of clock standard deviation. When the

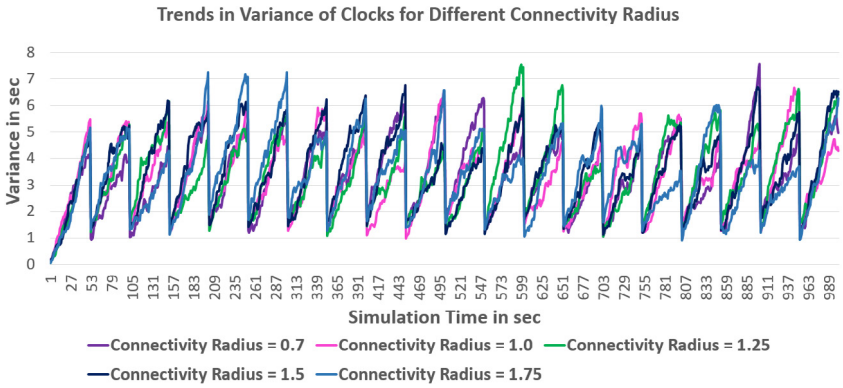

Fig. 16. Comparison for Standard Deviation in Clocks for Intra Data Center Architectures.

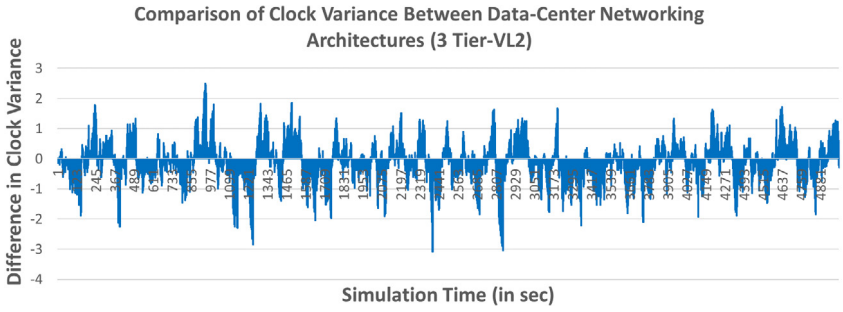

Fig. 17. Difference Between Clock Variance: 3-Tier and VL2.

Table 2
Analysis of Standard Deviation in Clocks for Intra Data Center Architectures.

Nodes	Data Center Architectures			
	Fat-Tree	Bcube	3 Tier	VL2
1	0.192347	0.278324	0.192347	0.192347
4	0.192347	0.278324	0.192347	0.192347
7	0.192347	0.278324	0.192347	0.192347
10	0.192347	0.278324	0.192347	0.192347
13	0.192347	0.278324	0.192347	0.192347
16	0.192347	0.278324	0.192347	0.192347
19	0.192347	0.278324	0.192347	0.192347
22	0.192347	0.278324	0.192347	0.192347
25	0.192347	0.278324	0.192347	0.192347
28	0.192347	0.278324	0.192347	0.192347
31	0.192347	0.278324	0.192347	0.192347
34	0.192347	0.278324	0.192347	0.192347
37	0.192347	0.214982	0.192347	0.192347
40	0.192347	0.278324	0.192347	0.395679
43	0.192347	0.278324	0.192347	0.395679
46	0.192347	0.278324	0.192347	0.395679
49	0.362147	0.278324	0.192347	0.395679
52	0.362147	0.278324	0.192347	0.485732
55	0.362147	0.278324	0.383245	0.485732
58	0.362147	0.278324	0.383245	0.485732
61	0.362147	0.278324	0.383245	0.472391
64	0.362147	0.278324	0.383245	0.472391
67	0.310522	0.278324	0.383245	0.472391
70	0.362147	0.278324	0.383245	0.413467
73	0.362147	0.278324	0.441792	0.413467
76	0.362147	0.362147	0.441792	0.413467
79	0.362147	0.362147	0.441792	0.413467
82	0.362147	0.362147	0.441792	0.265432
85	0.362147	0.313291	0.493241	0.265432
88	0.362147	0.362147	0.493241	0.265432
91	0.310522	0.362147	0.426982	0.265432
94	0.362147	0.331543	0.263240	0.265432
97	0.486432	0.498231	0.263240	0.265432
100	0.362147	0.362147	0.263240	0.265432

number of nodes are in the **range of 40–70** then **VL2** has the maximum value and **BCube** has the minimum value of standard deviation. For the number of nodes between **70 and 92**, **Three-Tier** gives the highest value of standard deviation.

Table 3
Analysis of Standard Deviation in Clocks for Intra Data Center Architectures.

Simulation Time (in μs)	Connectivity Radius				
	0.7	1.0	1.25	1.5	1.75
100000000	3.8674	5.0057	5.2848	5.2413	5.1259
200000000	6.1662	6.0271	4.8743	5.7964	7.2571
300000000	5.7329	4.8494	5.7141	5.8473	7.2613
400000000	5.5060	4.7216	5.3491	6.3966	6.2572
500000000	6.3211	6.3227	4.5523	4.2823	6.5794
600000000	4.9701	6.0913	7.4469	6.1081	3.9515
700000000	4.6305	4.6581	3.7037	4.6248	5.8756
800000000	5.5035	5.3535	5.3647	5.2254	3.5344
900000000	7.5662	4.7797	5.1081	6.5845	4.3026
1000000000	4.9650	4.2885	6.4149	6.5318	6.2649

Experiment 4: This experiment shows the variance in clocks for different connectivity radius as shown in Fig. 16 and Table 3. The simulation is run for 1000 s and comprises of 60 nodes, the variance in clocks for the nodes within the connectivity radius of 0.7, 1.0, 1.25, 1.5, and 1.75 units were recorded. It is observed that the range of values for variance in clocks is minimum for connectivity radius = 1.0 unit (**6.594269076**) and maximum for connectivity radius = 0.7 units (**7.478001273**). No particular trend in the values is observed, and hence, it can be concluded that the variance in clocks depends on other factors as well.

2. Inter-Data Center Synchronization:

Experiment 5: Case-I: Here, an inter-data center architecture is implemented using Three-Tier for one data center and VL2 DC architecture for another data center as shown in Fig. 17. On subtracting the clock variance values of VL2 from that of Three-Tier, it is observed that the average of the difference between the values of clock variance at each instant was −**0.14237 s**. Since the average value is negative, it means that the clock variance is higher in the data center with the VL2 DC architecture.

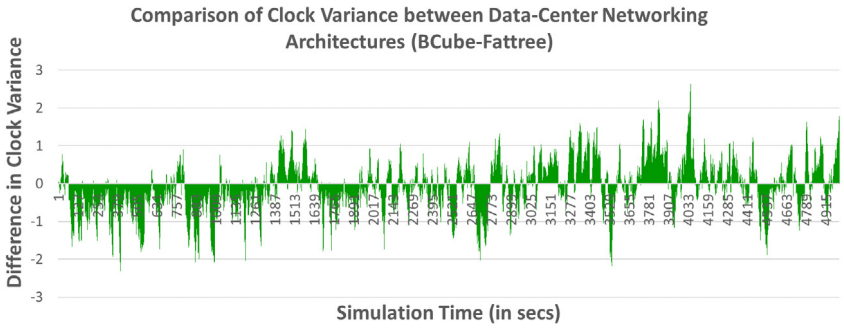

Fig. 18. Difference Between Clock Variance: BCube and Fat-tree.

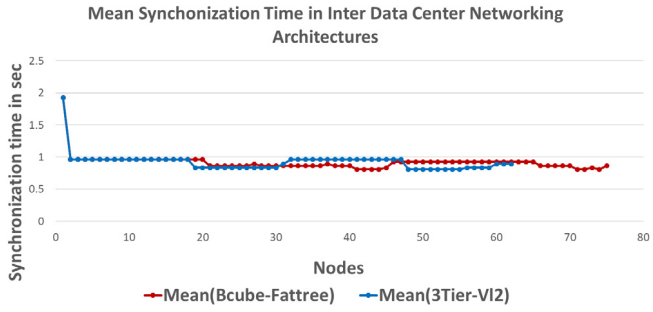

Fig. 19. Difference in Mean Synchronization Time: 3 Tier-VL2 and BCube-Fat-tree.

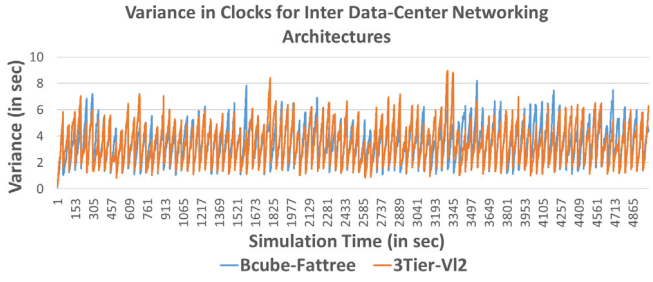

Fig. 20. Difference Between Clock Variance: 3 Tier-VL2 and BCube-Fat-tree.

Case-II: Here, an inter-data center architecture is implemented using BCube for one data center and Fat-tree DC architecture for another data center as shown in Fig. 18. On subtracting the clock variance values of Fat-tree from BCube, it is observed that the average of the difference between the values of clock variance at each instant is **−0.09011 s**. Since the average value is negative, it means that the clock variance is higher in the data center with the Fat-tree DC architecture.

Comparing the two scenarios, the **range** for the Three tier-VL2 inter data center communication is **5.58712 s** with an **average** of **−0.14237 s** whereas for BCube-Fat-tree inter data center communication, the **range** is **4.942972 s** with an **average** of **−0.09011 s**. It can be deduced from these observations that the inter data center communication with BCube-Fat-tree DC architecture is better.

Experiment 6: Through this experiment, the trends for mean synchronization for the two scenarios, one where Three-tier and VL2 DC architectures are implemented for inter-data center synchronization and the other where BCube and Fat-tree DC architectures are implemented as shown in Fig. 19. No particular trend for synchronization of nodes between these two scenarios was observed. From Fig. 20 and Table 4, it can be inferred that the range in values of clock variance is more in the case of

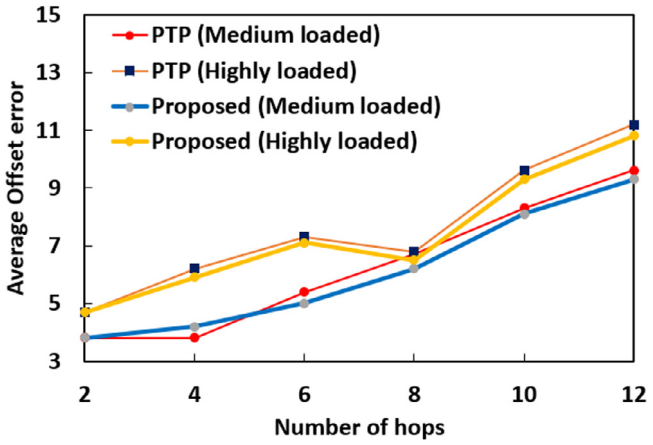

Fig. 21. Comparison of average offset error.

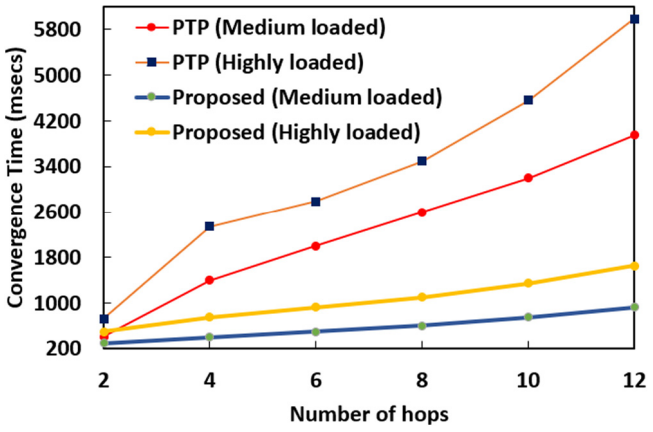

Fig. 22. Comparison of convergence time.

Three tier-VL2 inter-data center networking architectures and is equal to **8.87036** s as compared to the range **8.116116** s for the BCube-Fat-tree scenario.

Experiment 7: Fig. 21 represents the comparison of a offset error for cloud network with 12 levels of nodes. In this experiment, we analyze the average offset error and compare the proposed approach with standard PTP under medium and high network load. The average offset error for proposed and standard PTP is 3.845 clock units and 3.968 clock units respectively.

In Fig. 22, we have analyzed the convergence time of proposed and standard PTP for a network with 12 levels of nodes. Here, we find that during initial levels the overall convergence time gain in proposed approach is exponential in comparison with

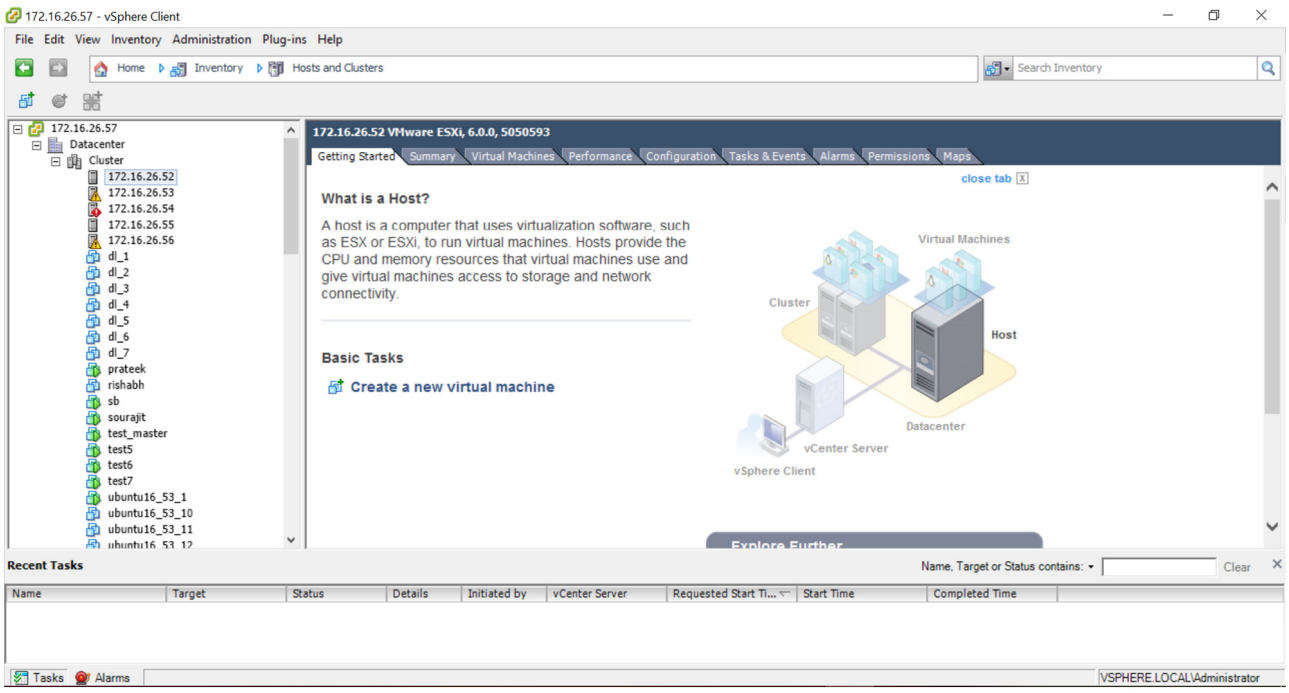

Fig. 23. vSphere Client interface.

Table 4
Difference Between Clock Variance: BCube-Fat-tree and 3 Tier-VL2.

Simulation Time (in μs)	Data Center Architectures	
	Bcube-Fat-Tree	3 Tier-VL2
500000000	4.2439	3.4815
1000000000	4.3072	5.1086
1500000000	6.2509	5.4121
2000000000	6.0959	6.5187
2500000000	5.2076	4.7298
3000000000	6.2250	6.3434
3500000000	6.1253	5.4031
4000000000	6.4468	4.0305
4500000000	5.6781	4.2662
5000000000	4.3546	6.3177

standard PTP and after the third level the convergence time gain is nearly constant around 59%. Finally, we find that the overall performance enhancement of proposed approach is around 47.82%.

6.3. Real-time experiments with VMware vSphere

To perform real-time experiments, we have setup a HPC cluster having 1-master node with configuration Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz with 2 CPU sockets, (2*8 = 16core), 256 GB RAM, 4 TB Hard disk and four compute servers each with configuration Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz with 2 CPU sockets (2*8 = 16core), 128 GB RAM, 1 TB Hard disk. Each compute node is connected using infiniBand (IB) network (56 Gbps) as well as Ethernet interconnection Ethernet to master node server. For virtualization, we have used VMware vSphere 5.5 with ESXi type 1 hypervisor, vSphere client and vCenter components in a four host machines with configuration 2 × Intel®Xeon®processor, CPU E5-2630V3 @ 2.4 GHz, (2*8 = 16 cores), 64 GB RAM, 1 TB Hard disk. Each host machine have 16 VMs with guest OS as Ubuntu 16. One master machine is having vcenter manager to manage all the VMs. We have used vSphere web client interface to access the cluster as shown in Fig. 23. Fig. 24 shows the cluster network map of all the host machines with active VMs.

We analyze the timekeeping architecture, in the context of ESXi hypervisor under live VM migration for different big data applications, which are critically dependent on reliable timekeeping. We have created a job-bank with 20 jobs including Wordcount, Naive Bayes Classifier, K-Means Clustering using Spark Graphx and Machine Learning Library (MLlib) to create variable loads at different nodes in the cluster. We have analyzed the clock errors under migration and compare the performance of ntpd with proposed method for intra-data center synchronization as shown in Fig. 25. As expected, ntpd produces extremely large errors. Whereas the proposed approach is narrowly affected by the dynamic variations in the system load and network traffic along with the migrations of the guest. As a result, ntpd is a clock synchronization paradigm, which is incompatible with VM migration.

7. Conclusions

In this work, we have proposed an enhanced DTP and wireless PTP based algorithms to achieve high precision clock synchronization for on-demand live VM migrations. We have demonstrated their performance on various data center networking architectures such as Fat-tree, Three-tier, VL2, and Bcube. Through simulations, we have observed that the optimal precision for clock synchronization is 0.55560 s at intra-data center architecture for 100 nodes using Bcube networking architecture. On the other hand, the synchronization time for Fat-tree, VL2, and Three-tier is 3.8454% more than the least time taken by the nodes in Bcube. Whereas in the inter-data center scenario, the achieved optimal precision is bounded by 0.80648 s for 62 connected nodes in Three-tier - VL2 network. We have also compared the convergence time of proposed approach with standard PTP for inter-data center scenario and observe that the convergence time gain is nearly constant around 59% after the third level. The overall performance enhancement of proposed approach for inter-data center is around 47.82%.

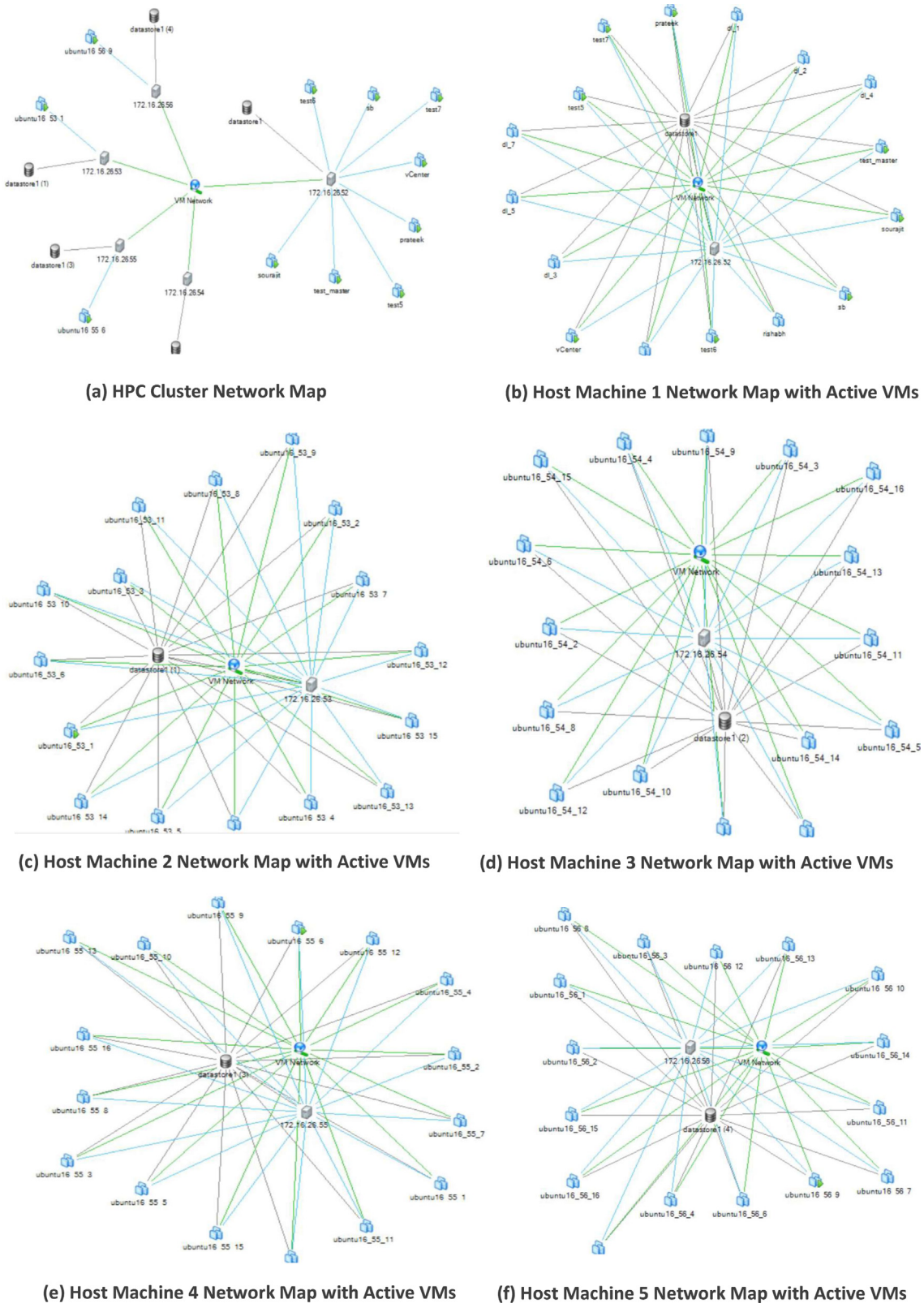

Fig. 24. HPC Cluster Network Map.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.jpdc.2019.11.012.

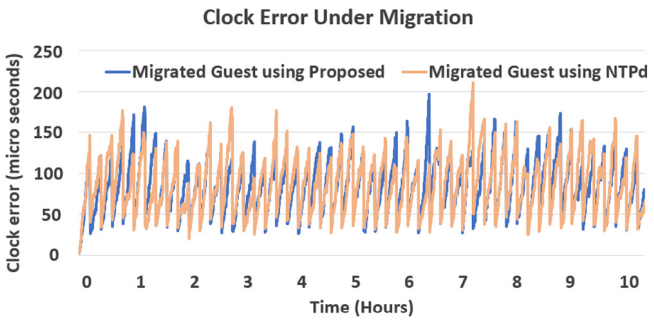

Fig. 25. Analysis of Clock Errors during Migration.

Acknowledgments

We thank the anonymous reviewers and the editor of Journal of Parallel and Distributed Computing for their expertise comments and valuable suggestions, which have helped us to improve the quality and presentation of the work significantly. It is acknowledged that the work of Y. S. Patel is partially supported by Department of Science & Technology (DST), Govt. of India, New Delhi, India under ICPS Programme through the Project Number: T-403, "Low-cost Energy-Efficient Cloud for Cyber–Physical Disaster Management Systems". He also acknowledges Visvesvaraya Ph.D. Scheme for Electronics and IT, an initiative of the Ministry of Electronics and Information Technology (MeitY), Government of India for support. The work of S. K. Das is partially supported by NSF grants CNS-1818942, CCF-1725755, and CBET-1609642.

References

[1] E. Ahmed, A. Naveed, A. Gani, S.H.A. Hamid, M. Imran, M. Guizani, Process state synchronization-based application execution management for mobile edge/cloud computing, Future Gener. Comput. Syst. 91 (2019) 579–589, http://dx.doi.org/10.1016/j.future.2018.09.018, URL http://www.sciencedirect.com/science/article/pii/S0167739X18307970.

[2] T. Broomhead, L. Cremean, J. Ridoux, D. Veitch, Virtualize everything but time, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 451–464, URL http://dl.acm.org/citation.cfm?id=1924943.1924975.

[3] D.M.J. Chauhan, A. Arkles, Is doing clock synchronization in a VM a good idea? in: Proc. IEEE Int. Perform. Comput. Commun. Conf., 2010, pp. 1–2.

[4] J. Chauhan, D. Makaroff, A. Arkles, VM clock synchronization measurements, in: 30th IEEE International Performance Computing and Communications Conference, 2011, pp. 1–2, http://dx.doi.org/10.1109/PCCC.2011.6108101.

[5] H. Cho, J. Jung, B. Cho, Y. Jin, S. Lee, Y. Baek, Precision time synchronization using IEEE 1588 for wireless sensor networks, in: 2009 International Conference on Computational Science and Engineering, Vol. 2, 2009, pp. 579–586, http://dx.doi.org/10.1109/CSE.2009.264.

[6] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Vol. 2, NSDI'05, USENIX Association, Berkeley, CA, USA, 2005, pp. 273–286, URL http://dl.acm.org/citation.cfm?id=1251203.1251223.

[7] S.Y. Geng, Exploiting a natural network effect for scalable, fine-grained clock synchronization, in: Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, NSDI '18, USENIX Association, 2018, pp. 81–94, URL http://dl.acm.org/citation.cfm?id=3307441.3307449.

[8] IEEE standard for a precision clock synchronization protocol for networked measurement and control systems, in: IEEE Std 1588–2008 (Revision of IEEE Std 1588–2002), 2008, pp. 1–300, http://dx.doi.org/10.1109/IEEESTD.2008.4579760.

[9] J.D. James C. Corbett, Spanner: Google's globally distributed database, ACM Trans. Comput. Syst. 31 (3) (2013) 1–22, http://dx.doi.org/10.1145/2491245, URL http://doi.acm.org/10.1145/2491245.

[10] H.W.K. Lee, Globally synchronized time via datacenter networks, in: Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, ACM, 2016, pp. 454–467, http://dx.doi.org/10.1145/2934872.2934885, URL http://doi.acm.org/10.1145/2934872.2934885.

[11] T. Lu, M. Stuart, SLM : Synchronized live migration of virtual clusters across data centers, 2013, pp. 1–2.

[12] A. Mahmood, R. Exel, H. Trsek, T. Sauter, Clock synchronization over IEEE 802.11—A survey of methodologies and protocols, IEEE Trans. Ind. Inf. 13 (2) (2017) 907–922, http://dx.doi.org/10.1109/TII.2016.2629669.

[13] J. Park, T. Kim, A method of logically time synchronization for safety-critical distributed system, in: 2016 18th International Conference on Advanced Communication Technology, ICACT, 2016, pp. 356–359, http://dx.doi.org/10.1109/ICACT.2016.7423390.

[14] D.A. Popescu, A.W. Moore, PTPmesh: Data center network latency measurements using PTP, in: IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS, 2017, pp. 73–79, http://dx.doi.org/10.1109/MASCOTS.2017.30.

[15] F. Ramos, J.L. Gutiérrez-Rivas, J. López-Jiménez, B. Caracuel, J. Díaz, Accurate timing networks for dependable smart grid applications, IEEE Trans. Ind. Inf. 14 (5) (2018) 2076–2084, http://dx.doi.org/10.1109/TII.2017.2787145.

[16] K. Tsakalozos, V. Verroios, M. Roussopoulos, A. Delis, Live VM migration under time-constraints in share-nothing iaas-clouds, IEEE Trans. Parallel Distrib. Syst. 28 (8) (2017) 2285–2298, http://dx.doi.org/10.1109/TPDS.2017.2658572.

Yashwant Singh Patel is currently working towards his Ph.D. degree in the Department of Computer Science and Engineering at Indian Institute of Technology (IIT) Patna, India. His Ph.D. work is supported by Visvesvaraya PhD scheme for Electronics & IT. He received his bachelors degree in Computer Science and Engineering from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India in 2011 and Masters degree in Computer Science and Engineering from KIIT Bhubaneswar, India in 2014. His research interests include Cloud Computing, Distributed Algorithms and Grid Computing.

Aditi Page is pursuing Master of Science in Computer Science from University of Florida, Florida, USA. She has a Bachelor's degree in Computer Science Engineering from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India in 2019. Aditi was a summer intern at the Indian Institute of Technology, Patna in 2018 during which she worked on the on-demand clock synchronization protocols and the algorithms for live VM migration used in parallel and distributed systems. Her interest areas are cloud computing, machine learning, data science and computational science.

Manvi Nagdev is pursuing Master in Computer Science from North Carolina State University, NC, US. She completed her Bachelor of Engineering from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India in 2019. She did a summer research internship at the Indian Institute of Technology, Patna, India in 2018 where she worked on clock synchronization protocols. Her interest areas include computer graphics, game development, and artificial intelligence.

Anurag Choubey is currently a Visvesvaraya research fellow in the department of Computer Science Engg at Indian Institute of Technology (IIT) Patna, India. He Completed his Masters degree from National Institute of Technology Patna, India and his bachelors degree from Tezpur Central University, India. His area of interest comprises of Distributed Systems, Peer-to-Peer network, Blockchain technology and Big Data. During his stay at IIT Patna, he has also worked as a teaching assistant for numerous courses like Distributed systems, Cloud Computing, Big data analytics and Foundations of Computer System. He is also working under a DST-DAAD sponsored inter-nation project at IIT Patna.

Rajiv Misra is currently working as an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Patna, India. He received the M.Tech degree in computer science and engineering from IIT Bombay and Ph.D. degree in the area of mobile computing from IIT Kharagpur. His research interests include Distributed Systems, Cloud Computing, Big Data Computing, Consensus in Blockchain, Cloud IoT Edge Computing, Adhoc Networks and Sensor Networks. He has contributed significantly to these areas and published more than 80 papers in high quality journals and conferences. He has authored papers in IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems etc. He is a senior member of IEEE.

Sajal K. Das is a professor of Computer Science and Daniel St. Clair Endowed Chair at Missouri University of Science and Technology, where he was the Chair of Computer Science Department during 2013–2017. His research interests include wireless sensor networks, mobile and pervasive computing, cyber–physical systems and IoTs, smart environments, cloud computing, cyber security, biological and social network. He has published more than 700 papers in high quality journals and refereed conference proceedings. He holds 5 US patents and coauthored 4 books. He is a recipient of 10 Best Paper Awards, IEEE Computer Society's Technical Achievement Award for pioneering contributions to sensor networks, and University of Missouri System President's Award for Sustained Career Excellence. He is the founding Editor-in-Chief of Elsevier's Pervasive and Mobile Computing, and Associate Editor of IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Mobile Computing, and ACM Transactions on Sensor Networks. He is an IEEE Fellow.