Contents lists available at ScienceDirect

Future Generation Computer Systems

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

bwSlicer: A bandwidth slicing framework for cloud data centers

Auday Al-Dulaimy [a,*], Wassim Itani [b], Javid Taheri [a], Maha Shamseddine [c]

[a] Department of Mathematics and Computer Science, Karlstad University, Sweden
[b] Department of Computer Science, University of Houston-Victoria, USA
[c] Department of Electrical and Computer Engineering, Beirut Arab University, Lebanon

ARTICLE INFO

Article history:
Received 21 October 2019
Received in revised form 21 April 2020
Accepted 14 June 2020
Available online 20 June 2020

Keywords:
Bandwidth allocation
Bandwidth Slicing
Cloud Computing
CloudLet
Fog Computing
VM management

ABSTRACT

Bandwidth allocation is an important and influential factor in enhancing the performance of the data centers' nodes. In this paper we propose bwSlicer, a framework for bandwidth slicing in cloud data centers, that sheds light on the virtues of effective dynamic bandwidth allocation on improving the system performance and energy efficiency. Three algorithms are investigated to deal with this issue. In the first algorithm, called Fair Bandwidth Reallocation (FBR), two virtual machines co-hosted on the same node conditionally exchange bandwidth slices based on their requirements. The second algorithm, called Required Bandwidth Allocation (RBA), periodically monitors the co-hosted virtual machines and adds/removes bandwidth slices for each of them based on their bandwidth utilization. The third algorithm, called Divide Bandwidth Reallocation (DBR), divides the bandwidth of the virtual machine into slices once it finishes its execution, and distributes the slices among the co-hosted running virtual machines according to a specific policy. The proposed bandwidth slicing algorithms are emulated in a virtualized networking environment using the Mininet network emulator. The emulation results demonstrated a promising improvement ratio in execution time and energy consumption reaching up to 30%. These results present a call for action for further research into bandwidth slicing and reallocation as a viable complement to other energy-saving techniques for enhancing the energy consumption in cloud data centers.

1. Introduction

Cloud computing is the most studied model in the last decade due to its innovative characteristics and wide range of services. Many studies focused on investigating the effect of virtualization management on improving the performance and energy efficiency of the nodes on the cloud data centers. However, most studies solely considered task resource requirements of processor, memory, and storage without considering bandwidth requirements and its effects on performance and energy consumption. Performance and energy efficiency in the data centers can be enhanced by observing how resources of such data centers are utilized to serve the jobs that request some services.

In the cloud computing model, Virtual Machines (VMs) are owned by different users or enterprises. This implies that the resulting workload is of mixed types of jobs. The mixed workload is formed by combining various types of jobs. These jobs request and utilize the resources at the same time. Each job has its own specific requirements. Bandwidth is one of the resources utilized by the VM allocated to the job. Accordingly, two or more VMs with different bandwidth requirements can be hosted on the same Physical Machine (PM). Unfair initial bandwidth allocation among the VMs definitely affects the performance of the system, as well as influences the energy efficiency. Therefore, the bandwidth must be fairly distributed among the VMs of the same PM.

This paper presents and discusses the concept of *"Bandwidth Slicing"*. Three algorithms are proposed under the umbrella of this approach to tackle the bandwidth requirements on data centers' nodes. The first algorithm, called Fair Bandwidth Reallocation (FBR), reallocates the bandwidth between a pair of VMs based on requirements of the jobs executed on these VMs at a specific period of time. The core idea of FBR is to maximize the utilization of the bandwidth of the PM. One VM (The *Granter VM*) provides a specific amount of its bandwidth to another VM (The *Grantee VM*) when it is idle, and then regains this amount after a period of time. The second algorithm, called Required Bandwidth Allocation (RBA), is proposed to dynamically allocate bandwidth to the VMs that are hosted on the PM. RBA treats bandwidth of VMs as a shared resource and allocates only the amount of bandwidth required by the VM(s) at that specific time. The core idea here is to monitor the bandwidth of the VMs periodically and to reallocate the bandwidth among these VMs according to their actual required bandwidth. The third algorithm, called

* Corresponding author.
E-mail address: auday.aldulaimy@kau.se (A. Al-Dulaimy).

https://doi.org/10.1016/j.future.2020.06.021
0167-739X/© 2020 Published by Elsevier B.V.

Divide Bandwidth Reallocation (DBR), is proposed to divide the amount of the bandwidth allocated to the VM that has finished its execution among the VMs that are hosted on the same PM. DBR divides the bandwidth of the VM that has finished its execution (*The Terminated*) once it finishes the execution of the job allocated to it among the VMs that are hosted on the same PM (*The Working VMs*). DBR can divide the bandwidth amount based on several suggested policies. The emulation results demonstrated major improvements in execution time and energy consumption reaching up to 30% improvement ratio. These results shed light on the importance of directing further research towards bandwidth slicing and reallocation as a means for reducing energy consumption in the cloud data centers together with other complementary energy saving techniques.

The main contribution of this paper lies in defining the concept of *"Bandwidth Slicing"*, and in designing and implementing the bandwidth slicing framework, which comes with three algorithms, aiming to reduce the total execution time and enhance the energy efficiency. These algorithms are:

(1) FBR for reallocating bandwidth among a pair of VMs hosted on the same PM,
(2) RBA that dynamically allocates the bandwidth to VMs when required, and
(3) DBR that divides the bandwidth of the terminated VM among the other VMs hosted on the same PM.

The three proposed algorithms, lead to enhance the performance which, in turn, increases the energy efficiency.

The rest of this paper is organized as follows: Section 2 is the motivation for this work. Section 3 presents some previous research works that stresses on the concept that bandwidth is a shared resource which requires better utilization. Section 4 describes the system model we follow in this work. The problem description and the proposed algorithms are detailed in Section 5, while the performance analysis and system simulation are described in Section 6. Conclusions are presented in Section 7.

2. Motivation

The wide adoption of cloud computing services is predicated by the ability to meet the diverse user's requests with varied requirements for latency, scalability, and availability while delivering multiplicity of use cases.

The cloud services are offered to serve a huge number of users through a pooling of heterogeneous physical resources (CPU, Memory, Storage, and Bandwidth) hosted in cloud data centers. Supported by the virtualization concept, users are expected to efficiently share/utilize the data centers resources, so that many VMs could be consolidated onto a fewer number of PMs. Beside its benefits, in term of utilization and energy efficiency, consolidating VMs on PMs can also create technical challenges that may affect the performance of the provisioned services. To avoid these challenges, hypervisors must allocate enough physical resources (CPU, Memory, Storage, and Bandwidth) to each virtual entity in order to meet its performance goals.

A *"Fixed-Size-Fits-All"* approach for co-hosted VMs serving different users/clients with different requirements is not viable anymore, so the resources should be granted and adjusted dynamically. Focusing on bandwidth as a physical resource, the key to this shift lies in how the co-hosted VMs can cooperate with each other in developing a *"Win-Win"* bandwidth sharing strategy.

The core idea of the bandwidth slicing approach is to divide the bandwidth allocated to the running VMs into slices in such a way that all VMs are mutually benefited and satisfied. Accordingly, all VMs consider the host PM as a cooperative operating environment, rather than a competitive one. Bandwidth slicing can be defined as the ability to orchestrate the capabilities of the bandwidth, as a physical resource, among different VMs co-hosted on the same host.

To clarify our approach, consider the following two scenarios:

Scenario one: A data intensive application (e.g. web application) is co-located with a compute intensive application (e.g. scientific application) on the same host. A data intensive application requires high bandwidth capabilities in the data stage in/out processes, whilst a compute intensive application mostly request compute resources. Data intensive and compute intensive applications do not comprehensively use the same physical resources of their host. A traditional bandwidth division will not be able to cater above demands efficiently. Hence cloud providers/operators will benefit from a more flexible sliced bandwidth design to meet the requirements of both types of applications.

Scenario two: Different service providers/vendors may provide the compute and/or bandwidth for different companies which operate or manufacture connected cars. Providers/vendors are able to allocate dedicated virtual resources to different car manufacturers and operate them independently. Each car manufacturer may be equipped with different services such as traffic efficiency and emergency support. Providers/vendors can create bandwidth slices to serve all car manufacturers in a mutually beneficial manner, such that one car manufacturer can grant a bandwidth slice from its already dedicated bandwidth to another car manufacturer for a specific period of time in some circumstances. The granting process is done under the management of the providers/vendors.

The proposed bandwidth slicing approach, capitalizes on the hypervisor's abilities in reconfiguring the VMs configurations to match the needs of the applications served by the VMs co-hosted on the same PM. This approach can offer several benefits by enhancing the ability of providers and operators to deploy, when possible, only the specific amount of bandwidth needed to serve specific use cases and users. Adopting bandwidth slicing enables providers and operators to provide service differentiation, which is most desirable because cloud providers usually serve an unprecedented diversity of users and applications.

In the proposed bandwidth slicing approach, all parties are mutually benefited due to the following facts:

- When a VM does not need a specific amount of bandwidth (the slice) for a specific period, it can lend the slice to another VM on the same host, in a competitive cost. Thus, it can lend a bandwidth slice and get part of the amount already paid for reserving its own total bandwidth.
- The VM which needs extra bandwidth can pay less if it gets a bandwidth slice from another co-hosted VM rather than buying it from the provider directly.
- The provider can get some fees for carrying out the management processes.

Although bandwidth slicing seems a promising approach in optimizing cloud and edge services, a lot of issues need to be investigated to enable it, including: *Bandwidth slicing criterion*, *Bandwidth slicing division*, and *Coordinating the slices*.

This work presents algorithms under the umbrella of BWSLICER framework to investigate and enable the bandwidth slicing approach.

3. Related works

Many previous works were dedicated to the allocation of resources to the VMs that are requested by the users' jobs and applications. Most of the existing works in the literature only consider the availability of CPU and memory as resources allocated to

Table 1
Related works.

Related work	Network BW	VM BW	Design objective(s)
[1]	✓		Maximize the throughput.
[2]	✓		Minimize the communication overhead, Minimize the execution time.
[3]	✓		Maximize the network utilization.
[4]	✓		Maximize the network utilization.
[5]	✓		Maximize the network utilization.
[6]	✓		Minimize the execution time.
[7]	✓		Minimize the communication overhead.
[8]	✓		Maximize the Throughput, Minimize the communication overhead.
[9]	✓		Maximize the network utilization.
[10]	✓		Maximize the network utilization.
[11]	✓		Maximize the throughput, Maximize the network utilization.
[12]	✓		Maximize the throughput, Maximize the network utilization.
[13]	✓		Minimize the latency, Maximize the network utilization.
[14]	✓		Minimize the number of running PMs.
[15]		✓	Minimize the execution time, Maximize the BW utilization.
[16]		✓	Maximize the network utilization.
[17]		✓	Maximize the network utilization.
[18]		✓	Maximize the network utilization.
bwSlicer		✓	Maximize the BW utilization, Minimize the execution time, Minimize the energy consumption.

VMs, but only a limited number of them focused on the concept of bandwidth limitation in enhancing the VMs performance.

Moreover, the revolution of big data adds new challenges to the traditional cloud solutions (e.g., latency, network bandwidth) [19], and identifies the need for novel approaches which take into account the bandwidth as an essential resources to overcome such challenges.

To date, approaches which are proposed to investigate the bandwidth limitation and its effects from different perspectives (see Table 1) can be categorized into the following two main themes.

3.1. Network bandwidth

Approaches in this group focus on the network bandwidth to enhance the system performance. The work in [1] proposes a model that allocates bandwidth requested by a specific job in a way that maintains the job requirements, and maximizes the system throughput by using a linear programming formulation. However, this model suits the heterogeneous computing paradigms, and thus,it is not convenient for cloud computing. In [2], a decentralized affinity-aware migration technique is presented. The technique reduces the network communication cost and improves the applications' runtime. By monitoring the network affinity between pairs of VMs, the authors integrate the dynamism and heterogeneity in the communication patterns and network topology, aiming to place heavily data-dependent or communicating VMs on the available physical resources as close to each other as possible.

The authors in [3] propose a bandwidth allocation algorithm to categorize and place VMs of the same application into same virtual networks, considering the network link in the scheduling process. The algorithm follows a minimum congestion model to avoid, as much as possible, sharing the same network link by multiple sources. It should be mentioned here that the research works presented in [2] and [3] do not consider the limitation of other system resources such as CPU or memory.

In [4] and [5], the authors present a multi-port bandwidth-bounded model for scheduling the divisible jobs/workloads. The proposed model works on a heterogeneous master/worker computing platform. After dividing the jobs/workloads, the master sends out chunks to workers over a network in a single or multiple rounds, such that the size and the number of chunks is changed to optimize the usage of the total available bandwidth of the network. The authors state that scheduling jobs is NP-complete when CPU and memory resources are considered under the bounded multi-port model. Thus, adding network bandwidth as a parameter into the mix of resources further increased the complexity of the scheduling process.

Aiming to reduce the total execution time of a set of running VMs, the work presented in [6] proposes a bandwidth-aware algorithm, called bandwidth-aware task-scheduling (BATS), for divisible task scheduling in the cloud. The algorithm utilizes a nonlinear programming method to solve the bounded multi-port model in order to allocate the proper number of jobs to each VM. In [7], the authors present an application-aware bandwidth guarantee framework, called AppBag, to allocate the bandwidth to VMs using only one-step ahead information. In addition, they propose a VM migration algorithm to adjust the bandwidth allocation and corresponding VM placement, subject to the network demands variation in future execution phases. In [8], the authors propose a model, called eBA, for bandwidth allocation that provides end-to-end bandwidth guarantee for VMs under large numbers of short flows and massive bursty traffic in cloud data centers. eBA leverages a distributed VM-to-VM rate control algorithm based on the logistic model under the control-theoretic framework. The authors of [9] from Microsoft present a model to reschedule VMs bandwidths. The model, called Measurement

Based Fair Queuing (MBFQ), consists of two levels of scheduling: the first level is the Microscheduler level which operates cheaply and paces VM transmissions, and the second level is a Macroscheduler that periodically redistributes tokens to Microschedulers based on the measured bandwidth of VMs. In [10], the authors design a bandwidth allocation system embedded in the cloud computing platform. The allocation system analyzes some statistical data collected from the running VMs and predicts the bandwidth utilization of each VM for the next period. Then, the system allocates the VMs bandwidth based on that prediction. The work in [11] proposes a VM-friendly architecture that achieves inner-VM switching and bandwidth allocation functions based on the network hardware. The bandwidth allocation process for each VM is performed depending on some statistics, significantly to avoid any performance cost on the CPU. To avoid any waste of bandwidth resources, the authors of [12] present a bandwidth allocation approach. The approach is working through two layers: static bandwidth guarantees at the tenant layer, and a dynamic rate allocation at the application layer to realize predictable performance. It comprises of three components: the first component is a network abstraction model that provides a simple and flexible way for tenants to specify network requirements, the second component is a two-phase VM placement algorithm that provides optimal combinations of ordering policies and dispatching policies to meet multiple goals, and the third component can achieve the fairness between guaranteed and unguaranteed tenants in utilizing the unused bandwidth resources. In [13], the authors propose a system framework, called Trinity, to meet three goals simultaneously: bandwidth guarantees for throughput-intensive applications, low latency for latency-sensitive applications, and maintaining a fully utilize network bandwidth. The framework consists of two parts: one Rate Controller (RC) to determine the traffic rates, and a number of VM-to-VM channels. Each pair of VMs has one VM-to-VM channel to cooperate with each other for some measurements. The controller and channels collaborating together to meet the targeting goals. The work in [14] formulates the VM placement problem as a bin packing problem, and solves it using a modified version of the Whale Optimization Algorithm (WOA). The placement decision depends on the available bandwidth of the nodes.

However, the works in [6–13], and [14] did not shed emphasis on the internal or inner bandwidth of the VMs, instead, they focus on the network bandwidth.

3.2. Virtual machine bandwidth

Approaches in this group focus on the internal VM bandwidth to enhance the system performance. The work in [15] proposes an algorithm that utilizes the bandwidth of one idle VM to increase the quote of another VM allocated to serve the so-called urgent job. The proposed algorithm computes the necessary bandwidth amount required to meet the deadline of a VM allocated to an urgent job, this amount is taken from another VM hosted on the same PM. However, the work only aims to allocate additional bandwidth amount to a specific job, which may negatively affect other metrics, such as the energy efficiency of the PM. A new bandwidth allocation protocol, called *Falloc*, was proposed in [16]. *Falloc* targets providing fairness across VMs hosted on the cloud data centers. To design *Falloc*, the authors model the data center's bandwidth allocation as a bargaining game and propose a distributed algorithm to achieve the asymmetric Nash Bargaining Solution (NBS). Falloc is designed to guarantee bandwidth for the VMs based on their bandwidth requirements and to share residual bandwidth in proportion to specific weights given to the VMs. The authors of [16] validate *Falloc* in [17] with experiments under different scenarios and show that by adapting to different network requirements of VMs, Falloc can achieve fairness among VMs and balance the tradeoff between bandwidth guarantee and proportional bandwidth sharing. In our previous work in [18], two methods are presented to fairly share bandwidth among the VMs co-hosted on the same PM, one method exchanges slices of bandwidth between a pair of VMs, and the other exchanges slices of bandwidth among all VMs on the host. Our current work extended [18] with the addition of another algorithm. Also, all algorithms proposed to reallocate the bandwidth amount among VMs are discussed with further details, and extensive performance evaluations are conducted to assess the effectiveness of the bandwidth slicing algorithms.

However, the works presented in [15,16], and [17] did not aim to enhance the energy efficiency in the data centers, while in [18], the work did not evaluate the proposed methods.

It is worth mentioning that some previous works discussed and tackled comparable concepts. The concept, called *Network Slicing*, defined in [20] as a network service deployment model that consists of three layers: (1) Service Instance Layer which represents the supported services as instances, (2) Network Slice Instance Layer which includes a set of network functions as logical network instances to serve the Service Instance(s), and (3) Resource layer which includes the available resources in the network. Then, the work in [21] discusses Network Slicing concept in details and presents its challenges and future research directions. The concept of *Edge Slicing* is presented in [22]. The authors propose a modified version of the predictive placement strategy for Virtualized Network Functions (VNF) in edge computing model.

However, they did not focus on the internal or inner bandwidth of the VMs.

4. Bandwidth slicing framework

This section sheds light on the importance of slicing the bandwidth and reallocating it in virtualized computing environments aiming to enhance the performance and energy efficiency, and describes the algorithms proposed in BWSLICER framework to achieve this aim.

4.1. The aim of virtual machine bandwidth slicing

Estimating an exact amount of bandwidth required to execute users' jobs is not constantly guaranteed in cloud/fog computing environments. In general, imprecise bandwidth estimation affects the system performance and, on the long run, results in energy waste.

Imprecise bandwidth estimation leads to two cases:

(1) Over bandwidth provisioning: this case results in low bandwidth utilization. This may influence the performance of the other VMs hosted in the same PM which, in turn, wastes energy.
(2) Under bandwidth provisioning: this case results in reducing the throughput of the system by expanding the total execution time of the jobs, which also wastes energy.

For both of the above cases, it is highly recommended to utilize the bandwidth as a resource optimally.

It is worth mentioning that the network interface card (NIC) in the PM uses nearly as much power in the idle state as when it is active. This is fundamentally due to the requirements of maintaining the network state for the purpose of ensuring availability and responsiveness [23]. Thus, it is reasonable to enhance the bandwidth usage in order to maximize the NIC utilization by reducing idle state time periods. In this case, the energy consumed by the NIC is exploited optimally.

Optimal (or near optimal) bandwidth utilization can be achieved by proposing algorithms to improve the ways of allocating bandwidth to VMs dynamically.

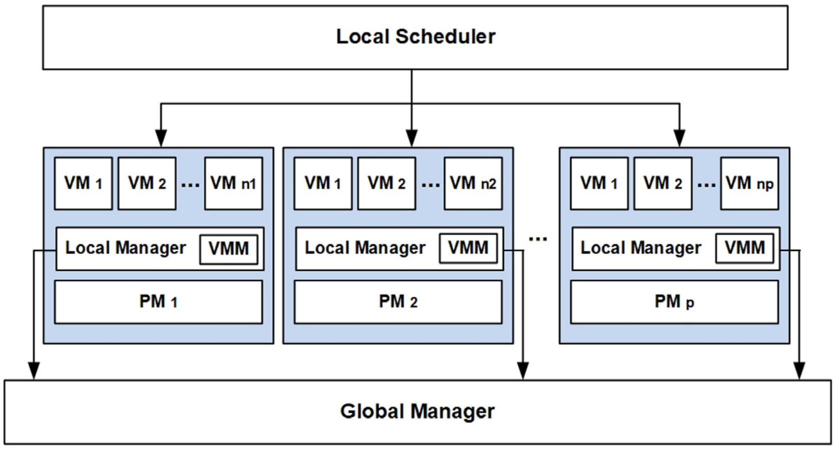


Fig. 1. The data center model.

4.2. System model

The model, which relies on the model presented in [24], is represented by:

(1) A data center in the Cloud Computing Layer, or
(2) A micro data center in the Fog Computing Layer or in the CloudLet.

As described in Fig. 1, each data center (or micro data center) consists of n heterogeneous PMs. Each PM_i has the following resources: a multi-core CPU, an amount of memory, and internal network bandwidth. PMs do not have a direct-attached storage, whereas storage is provided by a Network Attached Storage (NAS) or Storage Area Network (SAN) to enable any future live VM migration procedures. The concept of virtualization creates a virtual version from these resources to be used by different VMs hosted on the same PM.

The data center (or micro data center) has an essential unit called *Local scheduler*, *Local managers* (one local manager resides on each PM), and a *Global manager* (e.g., OpenStack or VMware vCenter) which maintains the overall system's resource utilization of a set of PMs through interaction with their local managers. The *Local scheduler* is responsible for allocating the demanded VMs to the jobs submitted by cloud users, and then, scheduling the VMs among the PMs in the data center. In serving the submitted jobs, the *Local scheduler*'s role stops in the scheduling process, and at this point, the bandwidth reallocation role begins, which is the responsibility of the *Local managers* and *Global manager*. The *Local managers* and *Global manager* cooperate with each other in order to utilize each PM's full bandwidth capacity. The *Local manager* specifies the bandwidth utilization of: (1) the PM, as well as (2) all its hosted VMs. Based on the bandwidth utilization, the granter and grantee VMs are nominated. At this point, the VMM selects the VM that can grant its bandwidth (or part of it) for a certain period of time. The *Local manager* informs the *Global manager* about the Granter VM and the period of time for this grant. Then, the *Global manager* informs the *Local manager* about the amount of the granted bandwidth. The VMM selects another VM co-hosted on the same PM and decides that it can receive a specific amount of bandwidth for a certain period of time. More details about the roles of the *Local managers* and the *Global manager* are discussed in Procedures I, II, and III in Section 5.

Notably, the number of nodes hosted on the micro data centers is relatively small compared to the number of nodes hosted on the data centers in the cloud. Also, the resource limitation in the micro data centers is recognizable compared with cloud data centers. Thus, adopting bandwidth slicing framework enhances the performance of the micro data centers' nodes.

Formally, the system represented by data center/micro data center D consists of two main sets: *PMs* and *VMs*.

The set of *PMs* has p elements, $PM = \{PM_1, PM_2, \ldots, PM_p\}$, each PM has the following resources:

$$PM_i = (PM_i^{CPU}, PM_i^{Speed}, PM_i^{CPI}, PM_i^{Storage}, PM_i^{Memory}, PM_i^{BW})$$

where:

- PM_i^{CPU}: the number of cores in PM_i.
- PM_i^{Speed}: the speed of the cores in MHertz (Million cycles per second).
- PM_i^{CPI}: the average number of cycles per instruction of the PM_i cores.
- PM_i^{Memory}: the memory capacity of PM_i in Mbytes.
- $PM_i^{Storage}$: the storage capacity of PM_i in Mbytes.
- PM_i^{BW}: the bandwidth capacity of PM_i in Mbytes per second.

The set of *VMs* consists of v elements, $VM = \{VM_1, VM_2, \ldots, VM_v\}$, each VM is configured as:

$$VM_i = (VM_i^{vCPU}, VM_i^{Speed}, VM_i^{CPI}, VM_i^{Storage}, VM_i^{Memory}, VM_i^{BW})$$

where:

- VM_i^{vCPU}: the number of cores in VM_i.
- VM_i^{Speed}: the speed of the VM_i cores in MHertz (Million cycles per second).
- VM_i^{CPI}: the average number of cycles per instruction of the VM_i cores.
- VM_i^{Memory}: the memory size dedicated to VM_i in Mbytes.
- $VM_i^{Storage}$: the storage dedicated to VM_i in Mbytes.
- VM_i^{BW}: the bandwidth dedicated to VM_i in Mbytes per second.

VM_i^{BW} is dedicated to VM_i, such that:
VM_i is placed on PM_j, and $VM_i^{BW} \leq PM_j^{BW}$

However, this work is more concerned with bandwidth as a resource in the PMs. The virtualized shared bandwidth is described as follows: Each PM in the data center has a specific bandwidth capacity. If the PM hosts n VMs, then, the PM bandwidth capacity is shared by n VMs. The amount of combined VMs bandwidth capacities is less than or equal to the total bandwidth capacity of the PM. This description is represented formally in Eq. (1) and illustrated in Fig. 2.

$$PM^{BW} = VM_1^{BW} + VM_2^{BW} + \cdots + VM_n^{BW} + R \quad (1)$$

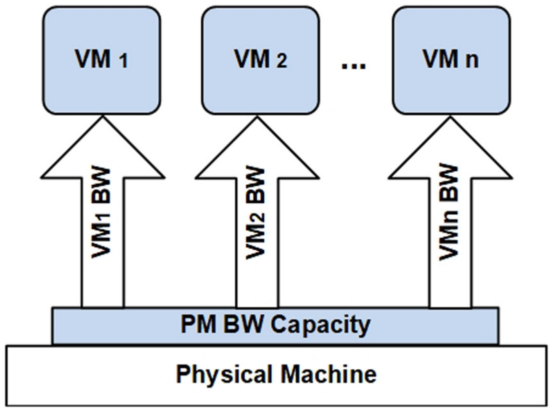


Fig. 2. The shared bandwidth for n VMs in one PM.

where:

- PM^{BW}: the total PM bandwidth capacity
- VM_i^{BW}: the bandwidth dedicated to VM_i hosted on the PM
- R: the residual bandwidth after sharing the total bandwidth capacity of the PM among VMs.

The resources of VMs are requested to serve cloud users. These requests are represented as a set of independent j jobs, $J = \{j_1, j_2, \dots, j_j\}$. Each job has specification parameters represented as:

$$j_i = \{L_i, DL_i, InputSize_i, OutputSize_i\}$$

where

- L_i is the length of job_i in MI (Million instructions)
- DL_i is the deadline of job_i in seconds
- $InputSize_i$ is the size of the input files of job_i in Mbytes
- $OutputSize_i$ is the size of the output files of job_i in Mbytes

4.3. Optimization objective

The problem can be considered as an optimization function which involves minimizing makspan of the VMs co-hosted on the same PM, for all PMs on the data center. The optimization function can be expressed as follows:

$$Min \sum_{j=1}^{p} \sum_{i=1}^{v} T_{pm_j}^{vm_i} \tag{2}$$

such that:

$$T_{pm_j}^{vm_i} = \left(\frac{InputSize_i}{VM_i^{BW}}\right) + \left(\frac{L_i}{VM_i^{Speed}} \times VM_i^{CPI}\right) + \left(\frac{OutputSize_i}{VM_i^{BW}}\right) \tag{3}$$

where:

- $T_{pm_j}^{vm_i}$: the total execution time when executing VM_i on PM_j.
- L_i: the length of job_i measured in Million Instructions
- $InputSize_i$ and $OutputSize_i$: the sizes of stage-in and stage-out data of job_i respectively, measured in Mbyte. The stage-in and stage-out data sizes are fixed for each job (static values), and they can vary from one job to another (dynamic values).
- VM_i^{CPI}: the average number of cycles needed to execute an instruction using VM_i.

Subject to the following constraints:

$$\forall i \in \{1, 2, \dots, v\} : \sum_{j=1}^{p} x_{i,j} \ , \ x \in \{0, 1\} \tag{4}$$

$$\forall i \in \{1, 2, \dots, v\} : ExT^{vm_i, pm_j} \le DL_i \tag{5}$$

$$\forall j \in \{1, 2, \dots, p\} : \sum_{i=1}^{v} VM_i^{BW} \times x_{i,j} \le PM_j^{BW} \tag{6}$$

Constraint (4) ensures that each VM is hosted on only one PM. The decision variable x is equal to '1' if the VM is assigned to the node, otherwise it is '0'. Constraint (5) makes sure that the actual total execution time of all VMs does not exceed the specified deadline. Constraint (6) is a capacity constraint to ensure that, for all PMs hosted on the data center, the total amount of VMs' bandwidths is less than or equal to the physical PMs's bandwidth.

5. Bandwidth slicing algorithms

To tackle the problems of performance degradation and energy waste in the data centers' nodes, this work proceeds in two ways:

(1) Speeding up the execution time of the jobs submitted to cloud data centers: the PMs running time is reduced, and therefore, energy consumption is decreased on the long run.
(2) Overlapping bandwidth of the VMs: Overlapping means that the bandwidth of the VMs can overlap and cooperate with each other in order to maximize the bandwidth utilization, and thus enhance the energy efficiency as well.

This work proposes a bandwidth slicing framework which comes with three directions, each has its own algorithm aiming to reduce the total execution time and enhance the energy efficiency. The three proposed algorithms, which are described in details in the next sections, lead to enhancing the performance which in turn increases energy efficiency.

5.1. Jobs execution

Job execution is mainly divided into data processes (data stage-in and data stage-out) and compute processes (actual execution) [25]. The FBR, RBA, and DBR algorithms exploit the sequence of the job execution process to improve the utilization of the bandwidth of the cloud and fog nodes. The stage-in process depends on the VM storage and bandwidth. In modern data centers, storage is provided as a NAS or a SAN that is accessible by all VMs in order to enable live VM migration. Therefore, the stage-in process for any job is speeded up when increasing the amount of bandwidth of the VM allocated to that job. Refer to Fig. 3.

In cloud and fog paradigms, VMs are owned by independent individuals or enterprises with various requirements. This implies that the resulting workload consists of mixed types of jobs. The mixed workload is formed by combining various types of applications, such as compute intensive applications and web applications. These applications request and intensively utilize different resources at an exact period of time (specific time slice). A VM allocated to a compute intensive job (usually its bandwidth can be idle for a certain period of time since the compute intensive job utilizes compute resources rather than storage and/or bandwidth resources) can be combined with another VM allocated to a data intensive job (VM allocated to data intensive job usually requires more bandwidth during data stage-in and/or stage-out process, as the data intensive job utilizes storage and/or bandwidth resources rather than compute resources) on the same PM.

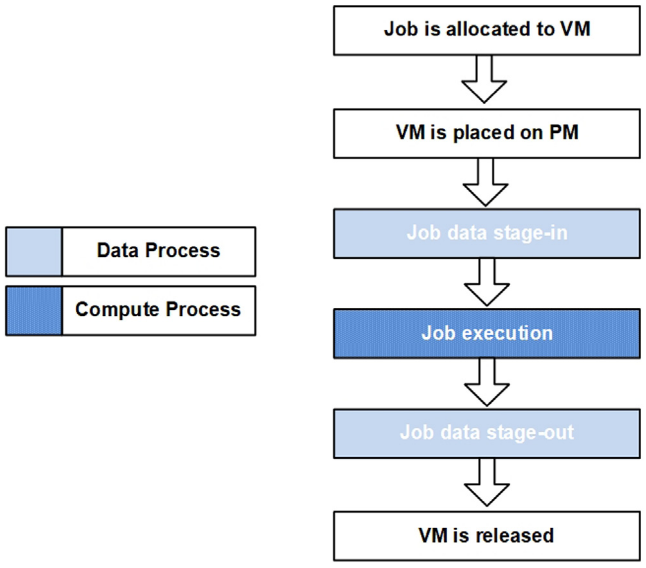


Fig. 3. The job execution sequence.

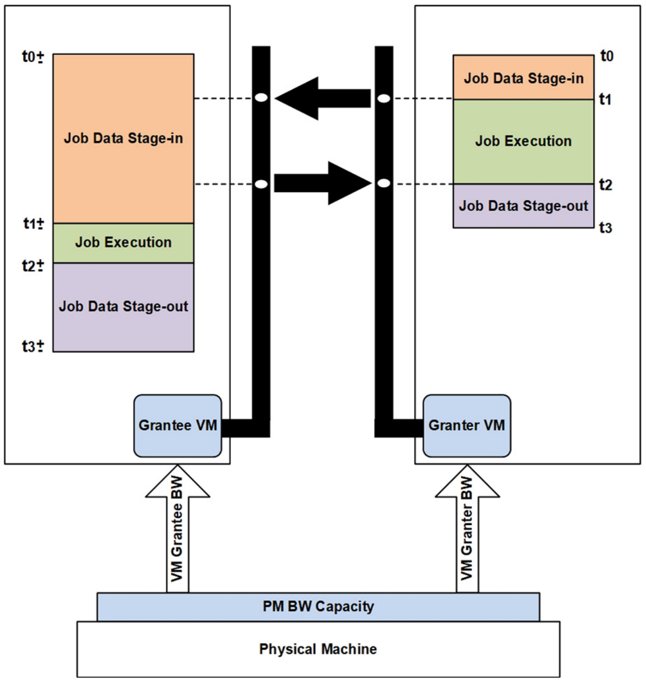


Fig. 4. The process of BW granting in the FBR Algorithm.

5.2. The FBR algorithm

As explained in the previous section, when co-hosting a compute intensive application and a web application on the same host, different resources at an exact period of time are requested to serve those applications. In this case, the VM of idle bandwidth (*Granter VM*) can grant its idle bandwidth amount (or part of it) to another VM (*Grantee VM*) for a certain period of time, this amount is called a bandwidth slice. This will decrease the duration of time needed for executing the job served by the *Grantee VM*, as the time needed to finish the data stage-in for any job is reduced. This enhances performance of the cloud PM hosting the *Granter* and *Grantee VMs*. Consequently, energy efficiency of the data center (or micro data center) is enhanced.

In order to prevent the idle state of all the VMs' bandwidth, FBR proposes a solution to utilize the PM's full bandwidth capacity. Bandwidth utilization for any VM ($VM_i^{BW\,util}$) can be calculated as presented in Eq. (7):

$$VM_i^{BW\,util} = \left(\frac{usedVM_i^{BW}}{VM_i^{BW}}\right) \times 100 \tag{7}$$

When the $VM_{Granter}^{BW}$ amount that is dedicated to a specific VM, termed as the $VM_{Granter}$, is idle while executing the job (in other words, the bandwidth utilization is low), releasing some amount of $VM_{Granter}^{BW}$ from $VM_{Granter}$ (bandwidth slice) for a certain period of time can be very efficient. The released amount from $VM_{Granter}^{BW}$ is added to another amount of bandwidth, $VM_{Grantee}^{BW}$, that is dedicated to another VM, called $VM_{Grantee}$, to speed up the data stage-in process of $VM_{Grantee}$. This process takes place when the *Grantee VM*, supplied with additional bandwidth, finishes the job execution in less time without affecting the execution of the Granter VM.

5.2.1. FBR description

This algorithm works with a pair of VMs (*Granter VM* allocated to job j_x and *Grantee VM* allocated to job j_y) as demonstrated in Fig. 4.

The granting is performed using the following procedures:
Procedure I (a): The main events at the *Granter VM* side are:

(1) The *Granter VM* is allocated to Job j_x.
(2) The *Granter VM* is placed on PM, say PM_{pm}.
(3) The *Granter VM* starts the stage-in process for the data of j_x (stage-in of $InputSize_x$).
(4) At the end of stage-in process, the bandwidth of the granter is idle (at time t_1).
(5) The VMM decides that this VM can grant its bandwidth (or part of it) for a certain period of time, which is approximately the time remaining to finish job execution. The *Local manager* informs the *Global manager* about the *Granter VM* and the period of time for this grant.
(6) The Granter releases its bandwidth and starts executing j_x.
(7) The Granter finishes the execution of j_x and gets back its bandwidth (at time t_2).
(8) The Granter starts the stage-out process for data of j_x (stage-out $OutputSize_x$).

Procedure I (b): On the other side, the main events at the *Grantee VM* are:

(1) The *Grantee VM* is allocated to Job j_y.
(2) The *Grantee VM* is placed on PM_{pm}.
(3) The *Grantee VM* starts the stage-in process for the data of j_y (stage-in of $InputSize_y$).
(4) The Grantee waits to the end of the data stage-in process to start execution.
(5) The *Global manager* informs the *Local manager* about the granted bandwidth. VMM decides that this VM can receive a specific amount of bandwidth for a certain period of time.
(6) The Grantee receives the added bandwidth from the granter for a specific period of time.
(7) The Grantee returns the bandwidth to the Granter.
(8) The Grantee either suspends its execution process until more data is staged-in, or starts the job execution process followed by the stage-out process for the data of j_y (stage-out $OutputSize_y$).

To speed up the execution, the main events at the *Grantee VM* can be applied at data stage-out phase of j_y (stage-out

Algorithm 1: The FBR Algorithm.

```
Input: Set $LIST_{vm}$ of VMs running and co-hosting on the same host
Output: Executed VMs
Begin
    While $LIST_{vm} \neq null$
        Check $VM_i^{BW}$ of all $VM_i \in LIST_{vm}$;
        If ( $VM_y \in LIST_{vm}$ running in StageIn/StageOut phase) and ( $VM_y^{BW_{util}}$ is highly utilized )
            $VM_y$ is a Grantee VM;
            Select $VM_x \in LIST_{vm}$ running in its execution phase as a Granter VM according to a specific policy;
            Calculate a bandwidth slice $bwSLICE$ such that: $bwSLICE \leq VM_x^{BW}$;
            $VM_x^{BW} = VM_x^{BW} - bwSLICE$;
            $VM_y^{BW} = VM_y^{BW} + bwSLICE$;
            Repeat
                Continue serving $VM_x$ and $VM_y$ with the current configuration
            Until $VM_x$ ends its execution phase;
            $VM_y^{BW} = VM_y^{BW} - bwSLICE$;
            $VM_x^{BW} = VM_x^{BW} + bwSLICE$;
End
```

$OutputSize_y$) as same as data stage-in phase (stage-in of $InputSize_y$). The time duration that the *Granter VM* can grant a specific amount of its bandwidth is represented in Eq. (8):

$$Time^{Granting} = t_2 - t_1 \tag{8}$$

which is the time required to execute the job served by the *Granter VM* and calculated in Eq. (9):

$$Time^{Granting} = \left(\frac{L_i}{VM^{Speed}_{Granter}} \right) \times VM^{CPI}_{Granter} \tag{9}$$

The value of $Time^{Granting}$ should be long enough compared to the time needed to reconfigure the bandwidth of the *Granter* and *Grantee VMs* to render the bandwidth slicing process merited. In addition, DL_x and DL_y should be met.

At a particular time, for instance t_0, every VM_i has a dedicated bandwidth, say VM_i^{BW}. To gain the maximum utilization of the bandwidth as a resource, the VMM monitors all the VMs which are co-hosted on the same PM. Once the VMM records a maximum bandwidth utilization for one of the VMs, then this VM is selected to be the *Grantee VM* which is in need for more bandwidth to complete its execution efficiently. The main reason of selecting the VM of maximum bandwidth utilization as a *Grantee VM* is due to the fact that when the VM which serves a specific job is experiencing the 100% bandwidth utilization, the data stage in/out phases of that job will be bounded by the bandwidth capacity of the VM; therefore, the VM will not be provided with the required performance level. Another VM hosted on the same PM must be selected as the *Granter VM*. The *Granter VM* grants a specific amount from its dedicated bandwidth to the *Grantee VM*. Algorithm 1 describes the generic structure of FBR algorithm.

In case more than one VM can be considered as a *Granter VM*, FBR algorithm suggested some policies to select the *Granter VM*, as explained in the next section.

5.2.2. FBR policies

FBR specifies the following policies to select the *Granter VM*: Random Selection (RS), Minimum bandwidth Utilization (MinU), Shortest Stage in/out (SS), and Longest Stage in/out (LS).

The RS policy: In RS policy, the VMM randomly selects one VM from the VMs running in their compute process to be the *Granter VM*. The RS policy is the easiest policy to be implemented.

The MinU policy: Here the VMM specifies the *Working VMs*, estimates their bandwidth utilization periodically, and selects the VM with the minimum bandwidth utilization during the last period to be the *Granter VM*. MinU policy needs to calculate the bandwidth utilization of each Working VM before selecting the *Granter VM*. However, selecting the VM with the maximum (or high) bandwidth utilization to be the *Granter VM* is not feasible, as it mainly utilizes most of the bandwidth allocated to it, and cannot grant its bandwidth amount to another VM.

The SS policy: In this policy, the VMM specifies the *Working VMs*, estimates the time needed to complete the data stage in/out process, and selects the VM with the shortest data staging time to be the *Granter VM*. SS policy needs to calculate the data staging time of each working VM before selecting the *Granter VM*.

The LS policy: In this policy, the VMM specifies the *Working VMs*, estimates the time needed to complete the data stage in/out process, and selects the VM with the longest data staging time to be the *Granter VM*. The LS policy needs to calculate the data staging time of each working VM before selecting the *Granter VM*.

5.3. The RBA algorithm

In this algorithm, bandwidth is dynamically allocated to the VMs hosted on the same PM during the execution time. It only allocates the amount of bandwidth required for the VM at a specific time. In FBR, the idea is to benefit from the unused bandwidth of the *Granter VM* by granting it to the *Grantee VM*. Whereas in RBA, the bandwidth of all VMs is periodically monitored while unutilized amounts of bandwidth are released. In this case, the value of R in Eq. (1) can be increased to be used by any VM within the same PM. RBA proposes a solution to utilize the PM's full bandwidth capacity by avoiding the idle state of all the VMs' bandwidth. Bandwidth utilization for any VM ($VM_i^{BW_{util}}$) can be calculated using Eq. (2).

5.3.1. The RBA description

This algorithm works on all VMs. In other words, by using the RBA algorithm, all VMs can be considered as *Granters* to give a specific bandwidth slice, or *Grantees* to get a specific bandwidth slice, according to their usage at a particular time.

At a certain time instance, say t_0, every VM_i has a dedicated bandwidth, VM_i^{BW}. To gain the maximum utilization of the bandwidth as a resource, the VMM monitors the way VMs use this resource at a certain period time t_0 to t_1 and analyzes whether to adjust their bandwidth or not.

The time slot in the operations of the RBA algorithm (i.e., the time between t_i and $t_i + 1$) is a dynamic value, and it is up to the VMM to decide this value. The decision must be made to be in line with the goal of minimizing the total execution times of the running VMs. As explained earlier, VMM is part of the *Local manager* which exists on every PM.

Two cases result from monitoring the bandwidth dedicated to the VMs:

Algorithm 2: The RBA Algorithm.

```
Input: Set $LIST_{vm}$ of VMs running and co-hosting on the same host
Output: Executed VMs
1  Begin
2    While $LIST_{vm} \neq null$
3      Repeat
4        Check $VM_i^{BW}$ of all $VM_i \in LIST_{vm}$;
5        If $VM_x^{BW_{util}}$ of $VM_x \in LIST_{vm}$ is lowly utilized
6          $VM_x$ is a Granter VM;
7          Calculate a bandwidth slice $bwSLICE^+$ such that: $bwSLICE^+ \leq VM_x^{BW}$;
8          $VM_x^{BW} = VM_x^{BW} - bwSLICE^+$;
9          $R = R + bwSLICE^+$;
10       If $VM_y^{BW_{util}}$ of $VM_y \in LIST_{vm}$ is highly utilized
11         $VM_y$ is a Grantee VM;
12         Calculate a bandwidth slice $bwSLICE^-$ such that: $bwSLICE^- \leq R$;
13         $R = R - bwSLICE^-$;
14         $VM_y^{BW} = VM_y^{BW} + bwSLICE^-$;
15     Until all $VM_i \in LIST_{vm}$ are executed;
16 End
```

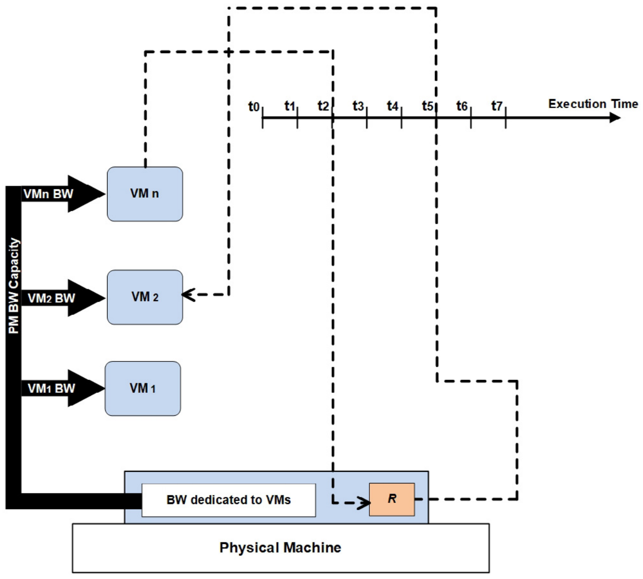


Fig. 5. The operations of the RBA algorithm.

- When the bandwidth VM_i^{BW} that is dedicated to VM_i is more than the required bandwidth by VM_i for slice β^+, then:

$$VM_i^{BW} = VM_i^{BW} - \beta^+ \tag{10}$$

$$R = R + \beta^+ \tag{11}$$

- When the bandwidth VM_j^{BW} that is dedicated to VM_j is less than that required by VM_j for slice β^-, then:

$$R = R - \beta^- \tag{12}$$

$$VM_j^{BW} = VM_j^{BW} + \beta^- \tag{13}$$

In both cases:
$\forall\, VM_i \in VM$ and hosted on PM_j:

$$VM_i^{BW} \leq PM_j^{BW} \text{ and } R \leq PM_j^{BW}$$

The VMM continues monitoring and updating the bandwidth allocated to the VMs at the second period of time t_1 to t_2, then at period t_2 to t_3 and so on until all jobs are executed and VMs are released, as illustrated in Fig. 5.

The granting process in RBA is performed using the following procedures:

Procedure II (a): The main events during the bandwidth granting process after m VMs = $\{VM_1, VM_2, \ldots, VM_m\}$ are allocated to m Jobs = $\{j_1, j_2, \ldots, j_m\}$, and the VMs are placed on the PM, say PM_{pm}:

(1) All VMs start serving their jobs (*StageIn* $\rightarrow$ *Execution* $\rightarrow$ *StageOut*) at t_0.
(2) The VMM monitors the way VMs use the bandwidth amount dedicated to them periodically.
(3) According to the bandwidth utilization, the VMM specifies the VMs which can grant a specific part of their BW (say x VMs, such that: $x \leq m$) for a certain period of time (t_i to t_{i+1}). The *Local managers* of the *Granter VMs* inform the *Global manager* about the *Granter VMs* and each grant duration.
(4) *Granter VMs* release a specific part of their bandwidth for the period (t_i to t_{i+1}), or even for additional time period(s) based on their bandwidth utilization.
(5) $R = R + \sum_{j=1}^{x} \beta_j^+$, such that: $\beta_j^+ \neq \beta_{j+1}^+$.

Procedure II (b): On the other side, the main events during the bandwidth receiving process:

(1) All VMs start serving their jobs (*StageIn* $\rightarrow$ *Execution* $\rightarrow$ *StageOut*).
(2) The VMM monitors the way VMs use the BW amount dedicated to them periodically.
(3) According to the bandwidth utilization, the VMM specifies the VMs which need a specific part to be added to their bandwidth (say y VMs, such that: $y \leq m$) for a certain period of time (t_i to t_{i+1}).
(4) Grantee VMs receive a specific bandwidth amount for the period (t_i to t_{i+1}), or even for additional period(s) based on their bandwidth utilization.
(5) $R = R - \sum_{j=1}^{x} \beta_j^-$, such that: $\beta_j^- \neq \beta_{j+1}^-$.

Any reduction in jobs' execution time leads to enhancing the overall PM performance. This also leads to reduction in the energy consumption in the cloud data center (or in the fog micro data center). Algorithm 2 describes the generic structure of RBA algorithm.

If there is a contention for a specific bandwidth slice between two (or more) VMs, RBA algorithm suggested some policies to select the *Grantee VM*, as explained in the next section.

5.3.2. RBA Policies

The following policies are proposed to select the *Grantee VM* and grant the BW to it in the RBA algorithm: Equally Division (ED), Maximum bandwidth Utilization (MaxU), and Priority Granting (PG).

The ED policy: In the ED policy, the VMM specifies the competitors VMs, and divides the amount of the bandwidth available in R equally among them. The ED policy is the easiest policy to be implemented.

The MaxU policy: The VMM specifies the competitors VMs, estimates their bandwidth utilization, and selects the VM with the maximum bandwidth utilization during the last period to be the *Grantee VM*.

The PG policy: In this policy, VMM grants the amount of bandwidth available in R to the VM with top priority. Within the same PM, each VM is given a priority, PG policy gives the bandwidth amount requested from R to the VM of the highest priority. Priority can be determined by one of the following mechanisms: (1) based on the job type served by the VM (the types of the jobs are discussed with more details in our works presented in [26] and [27]), or (2) based on the bandwidth utilization (or even any other resource utilization) of the VM. However, if two VMs (or more) are with the same priority, the bandwidth can be granted to any VM randomly, or divided among the VMs based on any of the other proposed policies.

5.4. The DBR algorithm

This algorithm divides the bandwidth amount of the VM that finish its execution (*The Terminated VM*) once it finishes the execution of the job allocated to it among the VMs (*The Working VMs*) that are hosted on the same PM. In DBR, the VMs are periodically monitored by the VMM, any VM that finishes its execution time is considered as the Terminated VM and its bandwidth, termed as the bandwidth slice, is released. In this case, the value of R in Eq. (1) can be increased by the released amount to be divided and used by the other VMs (*The Working VMs*) within the same PM.

5.4.1. The DBR description

As illustrated in Fig. 6, the DBR algorithm works on all VMs. In other words, by using the DBR algorithm, all VMs can be "*The Terminated VM*" or "*The Working VMs*" based to their total execution time when they are co-hosted together on the same PM.

At a particular time, for instance t_0, every VM_i has a dedicated bandwidth, say VM_i^{BW}. To gain the maximum utilization of the bandwidth as a resource, the VMM monitors all VMs which are hosted together on the same PM, once one of the VMs finishes its execution, DBR considers it as "*The Terminated VM*" and divides its bandwidth amount into slices among the other VMs "*The Working VMs*" based on a specific policy.

The granting process in DBR is performed using the following procedures:

Procedure III (a): The main events during the bandwidth granting process after m $VMs = VM_1, VM_2, \ldots, VM_m$ are allocated to m $Jobs = j_1, j_2, \ldots, j_m$, and the VMs are placed on the PM, say PM_{pm}:

(1) All VMs start serving their jobs ($stageIn \rightarrow Execution \rightarrow StageOut$) at t_0.
(2) The VMM monitors all working VMs to specify which job is finish its life time.
(3) When j_x finished its life time, the VM allocated to serve it, say VM_x, is marked as terminated.

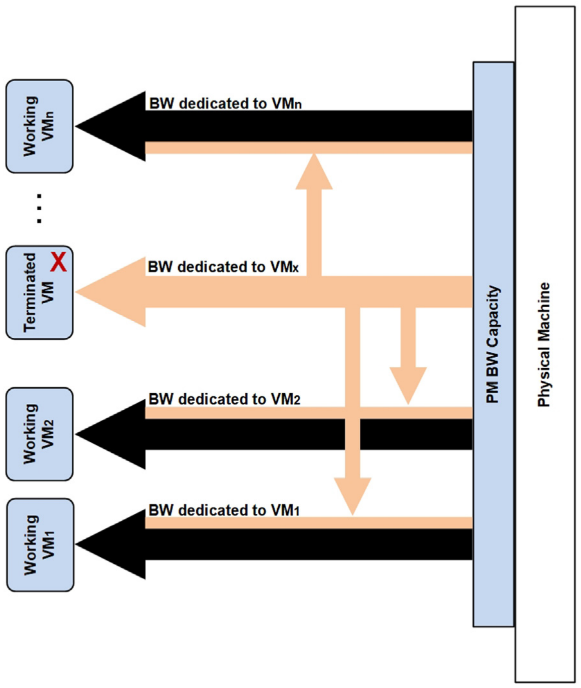


Fig. 6. The operations of the DBR algorithm.

(4) The VMM modifies the configuration of the terminated VM by granting its bandwidth to the other VMs hosted on the same PM. The *Local manager* of the terminated VM informs the *Global manager* about the details of the granted bandwidth.

Procedure III (b): On the other side, the main events during the bandwidth receiving process:

(1) All VMs start serving their jobs ($StageIn \rightarrow Execution \rightarrow StageOut$) at t_0.
(2) The VMM monitors all working VMs to specify which jobs are still working.
(3) The VMs are marked as Working VMs except the terminated VM_x.
(4) The VMM modifies the configuration of the Working VMs by receiving an extra bandwidth amount to their original bandwidth configuration. The *Local manager* of the working VMs informs the *Global manager* about the details of the received bandwidth.

Algorithm 3 describes the generic structure of DBR algorithm. Some policies are proposed to grant bandwidth in the DBR algorithm, as explained in the next section.

5.4.2. DBR Policies

The following policies are proposed to grant BW in the DBR algorithm: Equally Division (ED), Shortest Working VM (SW), Longest Working VM (LW), and Priority Granting (PG).

The ED policy: In the ED policy, the VMM calculates the number of Working VMs and divides the amount of the bandwidth available in R equally among the Working VMs. It is similar to the ED policy described RBA algorithm, but here it aims to divide the bandwidth of the Terminated VM among the Working VMs, while in RBA it aims to find a way to share a bandwidth slice among a competitive VMs at a specific time. The ED policy is the easiest policy to be implemented.

Algorithm 3: The DBR Algorithm.

```
Input: Set $LIST_{vm}$ of VMs running and co-hosting on the same host
Output: Executed VMs
1  Begin
2      While $LIST_{vm} \neq null$
3          Repeat
4              Monitor all Working VMs $\in LIST_{vm}$;
5              If $VM_x \in LIST_{vm}$ is terminated
6                  $VM_x$ is a Granter VM;
7                  $bwSLICE = VM_x^{BW}$;
8                  Distribute $bwSLICE$ among Working VMs according to a specific policy;
9          Until all $VM_i \in LIST_{vm}$ are executed;
10 End
```

The SW policy: In this policy, the VMM specifies the Working VMs, estimates their execution times, selects the VM that needs the shortest remaining time to finish its execution among them, and gives the amount of bandwidth in R to the selected VM. The SW policy needs to calculate how much time each Working VM will take to finish it execution.

The LW policy: the VMM specifies the Working VMs, estimates their execution times, selects the VM that needs the longest remaining time to finish its execution among them, and grants the amount of bandwidth available in R to the selected VM. As in the SW policy, the LW policy needs to calculate how much time each Working VM will take to finish it execution.

The PG policy: In this policy, one of the Working VMs is given a top priority and the VMM grants the amount of bandwidth available in R to the top priority VM. Within the same PM, each VM is given a priority, PG policy gives the bandwidth amount of the Terminated VM to VM of the highest priority first and so on. The PG policy here is very similar to PG policy presented previously in RBA. Also, as in RBA, priority can be determined by a specific mechanism, such as depending the job type served by the VM, or on the bandwidth utilization (or even any other resource utilization) of the VM. However, if two VMs (or more) are with the same priority, the bandwidth can be granted to any VM randomly, or divided among the VMs based on any of the other proposed policies.

6. Performance evaluation

In this section, a performance evaluation of the proposed bandwidth slicing approach, with all its three directions, is presented. We conducted both hypothesis scenarios and simulation with real workload traces driven by the Google clusters. In both the hypothesis scenarios and simulation parts, the following models and equations are used to measure the time reduction, energy saving, and the improvement ratio when applying the algorithms of the bandwidth slicing approach.

6.1. Performance metrics

The performance metrics used in this work to evaluate the performance of the bandwidth slicing approach are: execution time, energy consumption, and the improvement ratio.

6.1.1. Execution time

Relying on Eq. (3), we calculate the total execution time for each job before and after applying the proposed algorithms.

Moreover, Eqs. (8) and (9), discussed before in this paper, are used to calculate the granting time.

6.1.2. Energy consumption

There is a relation between the total execution time and the consumed energy, as stated in the model presented in [28]. This model illustrates the impact of CPU utilization on total power consumed by a PM. The power consumed by any PM grows almost linearly with the CPU utilization, from the value of power consumption in the idle state up to the power consumed when the server is fully utilized. This relation is illustrated in Eq. (14) [28].

$$P(u) = P_{idle} + (P_{busy} - P_{idle}) \times u \tag{14}$$

The total energy consumption of the PM for a period of time $[t_0, t_1]$ is illustrated in Eq. (15) [28], which shows that time influences the total amount of consumed energy:

$$E = \int_{t_0}^{t_1} P(u(t)).dt \tag{15}$$

However, the reduction in energy consumption per iteration in Joules (E_s) is calculated based on Eq. (16):

$$E_s = \left(\frac{PM_i^{Speed}}{PM_i^{CPI}} \right) \times \delta \times EPI \tag{16}$$

where:

- δ: the saving in execution time in seconds resulting from the application of the bandwidth slicing algorithms,
- EPI: stands for Energy Per Instruction, is the average energy consumed by the CPU to execute an instruction.

To approximate the EPI, we followed the energy model presented by Intel in [29]. Since the experiments in [29] were applied on a kind of outdated processors, this model is further leveraged in [30] on the relatively modern high-speed Intel Xeon Phi processor. Basing the *EPI* approximations on the Xeon Phi sprouts from the fact that it is the first production x86-based processor with multi-core and multi-threading support targeting supercomputers and high-end servers and workstations. Moreover, Xeon Phi belongs to the high-performance computing processor family designed by Intel for massive parallelism and vectorization support together with high-end energy efficiency. All this makes it a highly feasible choice for operation in cloud computing environments. Since the *EPI* mainly depends on the type of instructions executed by the processor and the location of the instruction operands, we utilized the model in [30] to get an approximate average of the EPI using scalar and vector instruction subtypes with a variety of operand locations and data movement combinations among the registers, L1 and L2 caches, and memory (with and without prefetching). The average energy consumption per instruction was found to be 59.6 nJ.

Table 2
A set of three jobs with their requirements.

job_i	$InputSize_i$ (MB)	$OutputSize_i$ (MB)	L_i (MI)	DL_i (Sec)
job_1	100	100	20 000	220
job_2	10 000	200	500	1025
job_3	100	400	30 000	350

Table 3
Samples of VMs configurations.

VM_i	VM_i^{BW} (MBPS)	VM_i^{Speed} (MHz)	VM_i^{CPI} (Cycle/instruction)
VM_1	10	1500	15
VM_2	10	1500	15
VM_3	10	1500	15

6.1.3. *Improvement ratio*

Eqs. (17) and (18) calculate the Improvement Ratio (*IR*) in performance and energy efficiency respectively:

$$IR = \left(\frac{Time_{old} - Time_{new}}{Time_{old}}\right) \times 100 \tag{17}$$

$$IR = \left(\frac{Energy_{old} - Energy_{new}}{Energy_{old}}\right) \times 100 \tag{18}$$

6.2. *Hypothesis scenarios*

This section presents some hypothesis examples to evaluate the performance of the proposed algorithms. The examples only consider the savings in execution time for motivational proof-of-concept purposes. A thorough simulation of the algorithms in a virtualized network environment considering the performance as well as the energy consumption aspects is presented in Section 6.3. Let us assume that there is a set of three jobs, $J = job_1, job_2, job_3$. Each job has four related parameters (the size of stage-in data in Mbytes, the size of stage-out data in Mbytes, the job length in Million instructions, and the deadline to be met when serving the job), as described in Table 2. And there is a set VM of three VMs, $VM = \{VM_1, VM_2, VM_3\}$, having the configurations described in Table 3. The VMs (VM_1, VM_2, and VM_3) are allocated to serve the jobs (job_1, job_2, and job_3) respectively. Note that VM_i^{BW} is in Mbytes per second (MBPS), VM_i^{Speed} is in Million instruction per second (MIPS), and VM_i^{CPI} is in Cycles per Instruction.

The effects of applying FBR, RBA, and DBR algorithms in reducing the total execution time for a set of jobs (and consequently enhancing the energy efficiency) are described in the next subsections.

6.2.1. *Applying FBR*

In the normal execution (without reallocating bandwidth dynamically), the total execution time (makespan), based on Eq. (3), for two jobs Job_1 and Job_2 (as FBR works with a pair of VMs) is 1025 s, as explained below:

$$T_{pm_j}^{vm_1} = \left(\frac{100}{10}\right) + \left(\frac{20000}{15} \times 15\right) + \left(\frac{100}{10}\right) = 220 \text{ s}$$

$$T_{pm_j}^{vm_2} = \left(\frac{10000}{10}\right) + \left(\frac{500}{1500} \times 15\right) + \left(\frac{200}{10}\right) = 1025 \text{ s}$$

If FBR is applied during the execution of VM1 (*The Granter*) and VM2 (*The Grantee*) which are allocated to Job_1 and Job_2 respectively, then the total execution time (makespan) for them will be reduced to 925 s. The process is described as follows:

$$T_{pm_j}^{vm_1Granter} = \left(\frac{100}{10}\right) + \left(\frac{20000}{15} \times 15\right) + \left(\frac{100}{10}\right)$$
$$= 220 \text{ s}$$

$$T_{pm_j}^{vm_2Grantee} = \left(\left(\frac{10000}{10} \times 0.8\right) + \left(\frac{10000}{20} \times 0.2\right)\right) + \left(\frac{500}{1500} \times 15\right) + \left(\frac{200}{10}\right) = 925 \text{ s}$$

When finishing its data stage-in, VM1 (*The Granter*) starts the execution phase and grants its bandwidth amount to VM2 (*The Grantee*). So, the data stage-in phase for VM2 works in two different bandwidth amounts.

The granting time is 20 s (i.e. it is about 20% from the data stage-in phase time of VM2). The bandwidth amount of VM2 is 10 MBPS (which is the original bandwidth dedicated to VM2) for 80% of the stage-in phase time, and 20 MBPS (which is the granted amount of bandwidth added to the original bandwidth amount dedicated to VM2) for 20% of the stage-in phase time. Thus, the PM which hosts VM1 and VM2 can be switched off (or it can start serving other VMs) after 925 s instead of after 1025 s by applying FBR. See Fig. 7(a).

And, based on Eq. (16), the amount of the saved energy by applying FBR algorithm (Fig. 7(b)) can be estimated as follows:

$$E_s = \left(\frac{1500}{15}\right) \times 100 \times 59.6 = 596\,J$$

And the improvement ratio of time saving (which is equal to the improvement ratio of energy saving) is:

$$IR = \left(\frac{1025 - 925}{1025}\right) \times 100 = 9.756\%$$

6.2.2. *Applying RBA*

In the normal execution (without reallocating bandwidth dynamically), the total execution time (makespan) for Job_1, Job_2 and Job_3 is:

$$T_{pm_j}^{vm_1} = \left(\frac{100}{10}\right) + \left(\frac{20000}{15} \times 15\right) + \left(\frac{100}{10}\right) = 220 \text{ s}$$

$$T_{pm_j}^{vm_2} = \left(\frac{10000}{10}\right) + \left(\frac{500}{1500} \times 15\right) + \left(\frac{200}{10}\right) = 1025 \text{ s}$$

$$T_{pm_j}^{vm_3} = \left(\frac{100}{10}\right) + \left(\frac{30000}{1500} \times 15\right) + \left(\frac{400}{10}\right) = 350 \text{ s}$$

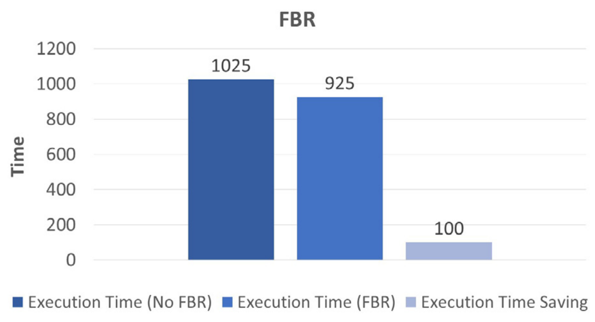
If RBA is applied during the execution of the VM1 (The Granter), VM2 (The Grantee) and VM3 (The Granter) which are allocated to Job_1, Job_2 and Job_3 respectively, then the total execution time (makespan) for them will be reduced to 841.667 s. Remarkably, there are two granter VMs and one grantee VM in this example. The process is described as follows:

$$T_{pm_j}^{vm_1Granter} = \left(\frac{100}{10}\right) + \left(\frac{20000}{15} \times 15\right) + \left(\frac{100}{10}\right) = 220 \text{ s}$$

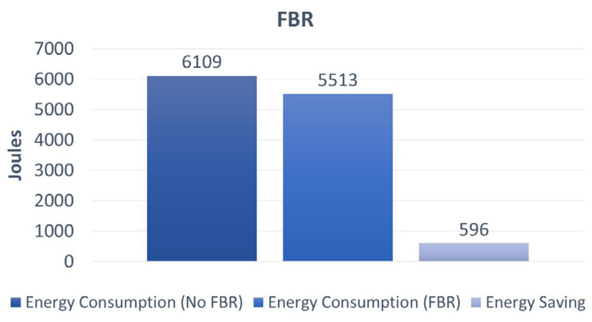
$$T_{pm_j}^{vm_2Grantee} = \left(\left(\frac{10000}{10} \times 0.7\right) + \left(\frac{10000}{30} \times 0.2\right) + \left(\frac{10000}{20} \times 0.1\right)\right) + \left(\frac{500}{1500} \times 15\right) + \left(\frac{200}{10}\right) = 841.667 \text{ s}$$

$$T_{pm_j}^{vm_3Granter} = \left(\frac{100}{10}\right) + \left(\frac{30000}{1500} \times 15\right) + \left(\frac{400}{10}\right) = 350 \text{ s}$$

When finishing their data stage-in, VM1 and VM3 (*The Granters*) start their execution phase and grant their bandwidth amounts to VM2 (*The Grantee*). So, the data stage-in phase for VM2 works in different bandwidth amounts.



(a) Execution time with and without applying the FBR algorithm.



(b) Energy consumption by applying the FBR algorithm.

Fig. 7. Results of applying the FBR algorithm.

From VM1, the granting time is 200 s (i.e. it is about 20% of the data stage-in phase time of VM2). And from VM3, the granting time is 300 s (i.e. it is about 30% of the data stage-in phase time of VM2). In this example, VM1 grants its bandwidth amounts in mutual time with VM3. So, the bandwidth amount of VM2 is 10 MBPS (which is the original bandwidth dedicated to VM2) for 70% of the stage-in phase time, 30 MBPS (which is the granted amount of bandwidth added to the original bandwidth amount dedicated to VM2 from both VM1 and VM3) for 20% of the stage-in phase time, and 20 MBPS (which is the granted amount of bandwidth added to the original bandwidth amount dedicated to VM2 from VM3) for 10% of the stage-in phase time.

Thus, the PM which hosts VM1, VM2 and VM3 can be switched off after 841.667 s instead of after 1025 s by applying RBA. See Fig. 8(a).

And, based on Eq. (16), the amount of the saved energy by applying FBR algorithm (Fig. 8(b)) can be estimated as follows:

$$E_s = \left(\frac{1500}{15}\right) \times 183.333 \times 59.6 = 1092.664\,J$$

And the improvement ratio is:

$$IR = \left(\frac{1025 - 841.667}{1025}\right) \times 100 = 17.886\%$$

6.2.3. Applying DBR

In the normal execution (without reallocating bandwidth dynamically), the total execution time (makespan) for Job_1, Job_2 and Job_3 is:

$$T_{pm_j}^{vm_1} = \left(\frac{100}{10}\right) + \left(\frac{20000}{15} \times 15\right) + \left(\frac{100}{10}\right) = 220\text{ s}$$

$$T_{pm_j}^{vm_2} = \left(\frac{10000}{10}\right) + \left(\frac{500}{1500} \times 15\right) + \left(\frac{200}{10}\right) = 1025\text{ s}$$

$$T_{pm_j}^{vm_3} = \left(\frac{100}{10}\right) + \left(\frac{30000}{1500} \times 15\right) + \left(\frac{400}{10}\right) = 350\text{ s}$$

If DBR is applied during the execution of the VM1, VM2 and VM3 which are allocated to Job_1, Job_2 and Job_3 respectively, then the total execution time (makespan) for them will be reduced to 711.667 s. Remarkably, in this scenario, VM2 acts as a Grantee VM for a period of time, and as a granter VM in another period. The process is described as follows:

$$T_{pm_j}^{vm_1Granter} = \left(\frac{100}{10}\right) + \left(\frac{20000}{15} \times 15\right) + \left(\frac{100}{10}\right) = 220\text{ s}$$

$$T_{pm_j}^{vm_2Grantee} = \left(\left(\frac{10000}{10} \times 0.22\right) + \left(\frac{10000}{15} \times 0.66\right) + \left(\frac{10000}{30} \times 0.12\right)\right) + \left(\frac{500}{1500} \times 15\right) + \left(\frac{200}{10}\right) = 711.667\text{ s}$$

$$T_{pm_j}^{vm_3Granter/Grantee} = \left(\frac{100}{10}\right) + \left(\frac{30000}{1500} \times 15\right) + \left(\frac{400}{15}\right) = 336.667\text{ s}$$

The first terminated VM is VM1 at second 220. So, it grants its bandwidth amount to VM2 and VM3 based on a specific policy. If we employ the ED policy in granting the bandwidth of the terminated VM, the bandwidth amount of VM1 is divided equally between VM2 and VM3. Thus, the data stage-in and stage-out phases for VM2 and VM3 work in different bandwidth amounts.

At second 220, VM1 divides its bandwidth amount, which is 10 MBPS, between VM2 and VM3 equally, 5 MBPS goes to each of VM_2^{BW} and VM_3^{BW}.

VM2 serves Job_2 with 10 MBPS till second 220 (about 22% from the data stage-in phase time of VM2). Then, it serves Job_2 with 15 MBPS after second 220.

Also, VM3 serves Job_3 with 10 MBPS till second 220 (i.e. till the start of the data stage-out phase time of VM3). Then, it serves Job_3 during the data stage-out phase time of VM3 with 15 MBPS as it starts after second 220. So, VM3 serves Job_3 with 10 MBPS in the data stage-out phase, and with 15 MBPS in all its data stage-out phase. With this bandwidth amount in the data stage-out phase, VM3 terminates its work at second 336.667, then VM3 becomes a bandwidth granter VM, and it grants its bandwidth to the other working VMs (only VM2 is still working at second 336.667).

So, VM2 serves Job_2 with 15 MBPS from second 220 till second 336.667 (about 12% of the data stage-in phase time of VM2). Then, it serves Job_2 with 30 MBPS after second 336.667 till the end of its life time (about 66% of the data stage-in phase, and 100% from the data stage-out phase time of VM2).

So, by applying DBR, the PM which hosts VM1, VM2 and VM3 can be switched off at second 711.667 in this scenario, instead of second 1025. See Fig. 9(a).

And, based on Eq. (16), the amount of the saved energy by applying FBR algorithm (Fig. 9(b)) can be estimated as follows:

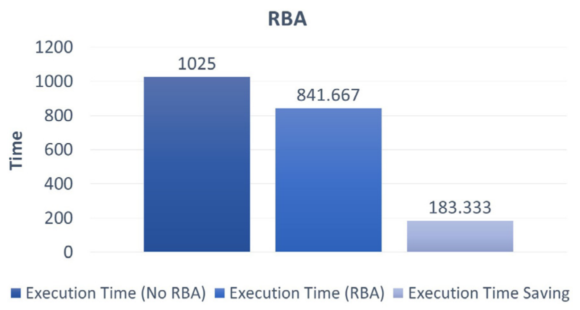
$$E_s = \left(\frac{1500}{15}\right) \times 313.333 \times 59.6 = 1867.465\,J$$

And the improvement ratio is:

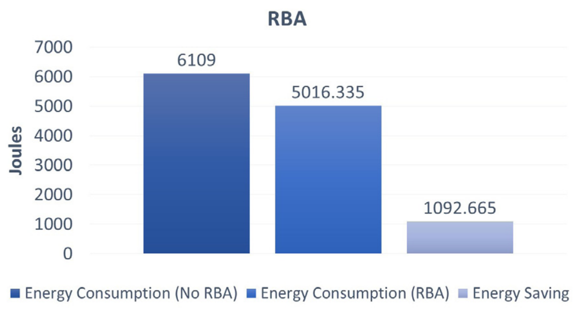
$$IR = \left(\frac{1025 - 711.667}{1025}\right) \times 100 = 30.569\%$$

6.3. Emulation

In this section, we present simulation results to evaluate and analyze the performance of the bandwidth slicing algorithms.

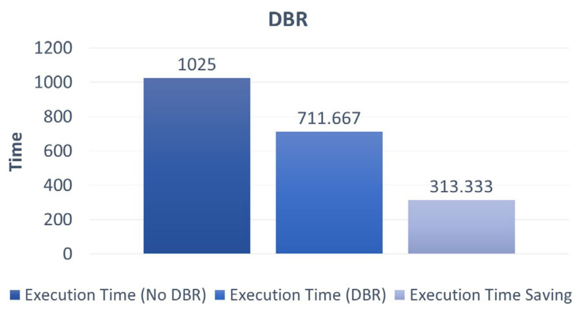


(a) Execution time with and without applying the RBA algorithm.

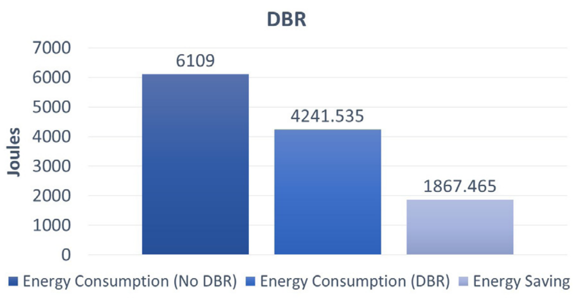


(b) Energy consumption by applying the RBA algorithm.

Fig. 8. Results of applying the RBA algorithm.



(a) Execution time with and without applying the DBR algorithm.



(b) Energy consumption by applying the DBR algorithm.

Fig. 9. Results of applying the DBR algorithm.

Table 4
Characteristics of the used physical machine.

PM	Characteristics
PM name	MacBook Pro
CPU family	Intel Core i7
Core speed	2.5 GHz
Memory	16 GB of DDR3 RAM

Table 5
Configurations of the used virtual machines.

VM	Configurations
VM core speed	2493.729 MHz CPU
Memory	2 GB RAM
Storage	16 GB
Bandwidth	10 MBPS

6.3.1. *Experimental testbed*

In order to demonstrate the feasibility of applying the proposed bandwidth slicing approach on real VMs, a proof-of-concept implementation is realized on top of the Mininet network emulator. Mininet [31] employs a process-based virtualization scheme to create a realistic network emulation environment consisting of virtual hosts (with real Linux kernels), switches/routers, Software-Defined Network controllers, and connectivity links. Mininet is deployed on a VMware Fusion [32] virtual machine running Ubuntu Linux 14.4. The physical machine employed to run Mininet is a MacBook Pro Mid 2015 laptop running OSX10.14 and supported with 2.5 GHz Intel Core i7 processors and 16 GB of DDR3 RAM. Table 4 summarizes the characteristics of the used PM.

For each bandwidth slicing algorithm, we created a set of Mininet hosts representing the set of $VMs = (VM_1, VM_2, \dots, VM_v)$, with v=10. Each VM is configured with 2 GB RAM, 2493.729 MHz CPU, 16 GB hard drive, and 10 MBPS network bandwidth. This results in a physical machine bandwidth of $\geq$ 100 MBPS. The VMM logic is executed in the SDN controller address space. The proposed bandwidth slicing algorithms are implemented in Python 3 and the communication between the VMM and the VMs is realized using standard TCP sockets. Table 5 summarizes the configurations of the used VMs.

The jobs dataset used is based on the Google Cluster-Usage Traces [33]. The main job attributes employed from the Google dataset are: (1) the mean CPU usage rate (attribute number 6 in the task resource usage table) together with the CPU speed to get the job length L_i, (2) the Cycles per Instruction (CPI) to calculate the energy savings per iteration (attribute number 16 in the task resource usage table), and (3) the maximum disk IO time (attribute number 15 in the task resource usage table) to get the job input size and output size ($InputSize_i$ and $OutputSize_i$ respectively). It is worth mentioning here that as we have two phases in the bandwidth slicing algorithms represented in the job data stage-in and data stage-out, we will divide the value of attribute 15 randomly between these two phases. We used a trace file consisting of 3.1 million job records randomly selected from the Google Cluster-Usage Traces dataset to feed the bandwidth slicing algorithms running on the Mininet VMs.

6.3.2. *Evaluating FBR*

In the FBR algorithm implementation, the granter VM is selected by the VMM based on the RS policy (refer to Section 5.1). Without loss of generality, the RS policy choice is adopted to simplify the simulation procedure. The jobs running on the Granter and Grantee VMs respectively are selected randomly from the Google Cluster-Usage Traces dataset for each iteration of the simulation experiments.

We executed the experimental setup for a period of 48 h where in each iteration the following features are calculated:

(1) the saving in execution time due to the application of the FBR algorithm, (2) the reduction in energy consumption resulting from the saving in execution time. This is equal to the amount of energy expended by the CPU if it were to execute the respective jobs during the saving in execution time period, and (3) the improvement ratio IR on a random pair of jobs Job_1 and Job_2 respectively executed on the Granter and Grantee VMs. At the end of the simulation period, the average saving in execution time, the average saving in energy consumption, and the average IR are computed by the VMM over all the iterations in the experimental setup.

Fig. 10(a) demonstrates (1) the average execution time per simulation iteration without applying the FBR algorithm, (2) the average execution time per iteration when FBR is applied, and (3) the average saving in execution time per iteration due to the FBR bandwidth slicing approach. FBR achieves an average of 19.53 s saving in execution time per iteration. Fig. 10(b) presents (1) the average energy consumption per simulation iteration without applying the FBR approach, (2) the average energy consumption per iteration when applying FBR, and (3) the average savings in energy consumption due to the application of the FBR algorithm. FBR reaches an average of 124.65 Joules energy savings per simulation iteration.

6.3.3. Evaluating RBA

In the RBA algorithm implementation, the two granter VMs are chosen randomly from the VM working set. The three jobs (2 jobs running on the two granter VMs and 1 on the Grantee VM) are chosen randomly from the Google Cluster-Usage Traces dataset for each iteration of the simulation. Analogous to the FBR implementation, we executed the experimental setup for an empirical period of 48 h where in each iteration the VMM calculates: (1) the saving in execution time due to the application of the RBA algorithm, (2) the reduction in energy consumption resulting from the saving in execution time. This is equal to the amount of energy expended by the CPU if it were to execute the respective jobs during the saving in execution time period, and (3) the improvement ratio IR on the 3 randomly selected jobs (Job_1 and Job_2 running on the two granter VMs while Job_3 running on the Grantee VM). At the end of the simulation period, the average saving in execution time, the average saving in energy consumption, and the average IR are computed by the VMM over all the iterations in the experimental setup. The energy savings are calculated based on the model in Eq. (12).

Fig. 11(a) shows (1) the average execution time per simulation iteration without applying the RBA algorithm, (2) the average execution time per iteration when RBA is applied, and (3) the average saving in execution time per iteration due to the RBA bandwidth slicing approach. RBA achieves an average of 45.72 s saving in execution time per iteration. Analogously, Fig. 11(b) presents (1) the average energy consumption per simulation iteration without applying the RBA approach, (2) the average energy consumption per iteration when applying RBA, and (3) the average savings in energy consumption due to the application of the RBA algorithm. RBA achieves an average of 301.25 Joules energy savings per simulation iteration.

6.3.4. Evaluating DBR

In the DBR algorithms implementation, a similar experimental setup to that of the RBA algorithm is applied. In each simulation iteration, three jobs are selected randomly from the Google Trace and executed on three VMs ($VM1, VM2\, and\, VM3$). As described in Section 5.3, in this scenario, VM2 acts as a grantee VM for a period of time, and as a granter VM in another period. In each iteration during the 48-hour simulation period, the VMM calculates the saving in execution time, the reduction in energy consumption, and the improvement ratio IR. Finally, at the end of the simulation period, the VMM computes the average execution time savings, the average energy reduction, and the average IR. The energy savings are calculated based on the model in Eq. (12). Again, without loss of generality and to simplify the experimental setup and analysis, the ED policy (refer to Section 5.3) is followed in the DBR implementation.

Fig. 12(a) presents (1) the average execution time per simulation iteration without applying the DBR algorithm, (2) the average execution time per iteration when DBR is enabled, and (3) the average saving in execution time per iteration due to the DBR bandwidth slicing approach. DBR achieves an average of 57.87 s saving in execution time per iteration. Similarly, Fig. 12(b) presents (1) the average energy consumption per simulation iteration without applying the DBR approach, (2) the average energy consumption per iteration when applying DBR, and (3) the average savings in energy consumption due to the application of the DBR algorithm. DBR reaches an average of 405.35 Joules energy savings per simulation iteration.

6.3.5. Improvement ratio measurements

The number of simulation iterations respectively executed in the FBR, RBA, and DBR implementations during the 48-hour simulation period was 132,484 iterations, 118,074 iterations, and 120,967 iterations respectively. The differences in the numbers of iteration for each algorithm because jobs are selected randomly, and each job has a specific length to be executed.

The average *IR* for reducing execution time and energy consumption is 8.72% in the FBR simulation, 22.65% in the RBA simulation, and 30.62% in the DBR simulation. This is demonstrated in Fig. 13.

The positive proof-of-concept testbed implementation results, though in a simulation environment, demonstrates the significance of applying bandwidth slicing and reallocation techniques and methodologies for improving the performance and energy efficiency in the data centers. Moreover, these results present a call for action for more research into bandwidth slicing and reallocation as a viable complement to other energy-saving techniques for enhancing data center energy consumption.

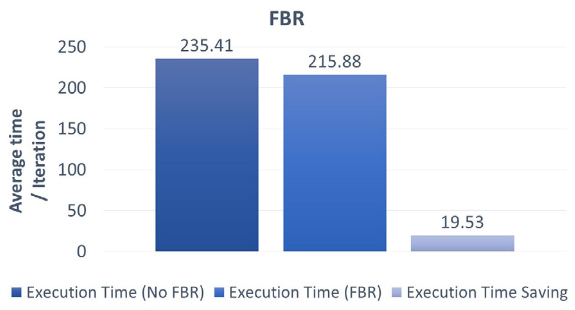
6.4. Discussion

By investigating how the proposed algorithms work, and observing the VMs' total execution time before and after applying the algorithms, the following notes can be concluded:

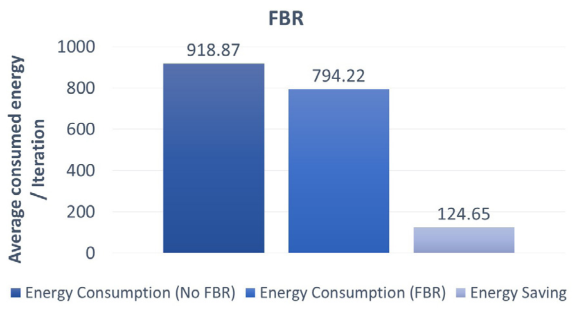
FBR is easier to be implemented than RBA and DBR. This is because monitoring the events of two VMs and changing the amount of their BW is easier than monitoring the events of all VMs on the host. Many events to be implemented periodically when applying RBA and DBR.

In RBA and DBR, all VMs can grant a part of their BW to a specific VM, while in FBR, only one VM can do the granting process. As a result, the RBA and DBR algorithms can results in a better improvement ratio in reducing the makespan of the involved VMs than the FBR algorithm. This is because every VM in the RBA and DBR algorithms can release their bandwidth amount (or part of it) to another VM on the same PM, while in FBR, only one VM can release its bandwidth (or part of it) to another VM which needs more bandwidth at a specific period of time to finish its execution faster.

Applying the proposed algorithms has no negative effect on the makespan of the involved VMs, and consequently on the PMs hosted those VMs, instead, it enhances the improvement ratio of the makespan with a specific percentage, or, at the worst case, do not affect the VMs execution. In other words, when applying the proposed BW reallocation algorithms, it either reduces the

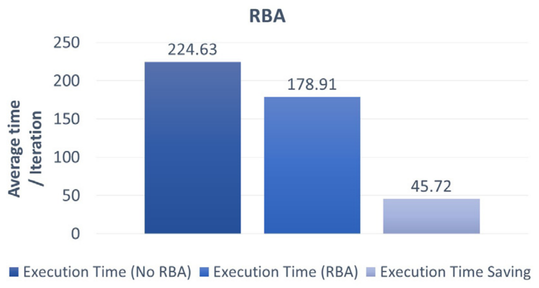


(a) Average execution time per emulation iteration by applying the FBR algorithm and the execution time savings achieved.

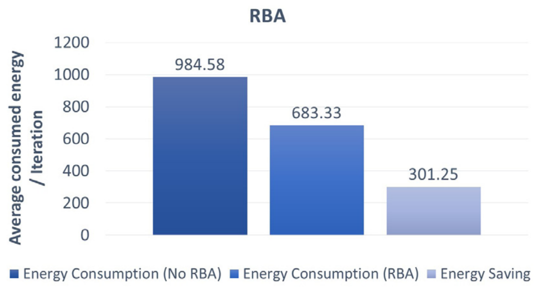


(b) Average energy consumption per emulation iteration by applying the FBR algorithm and the energy savings achieved.

Fig. 10. Emulation results of applying the FBR algorithm.

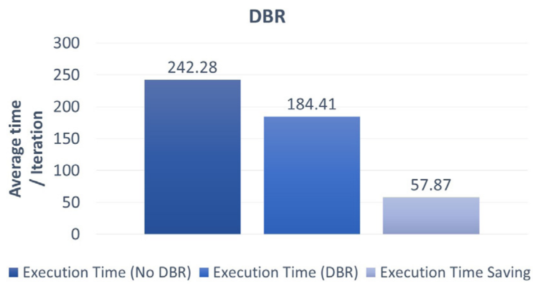


(a) Average execution time per emulation iteration by applying the RBA algorithm and the execution time savings achieved.

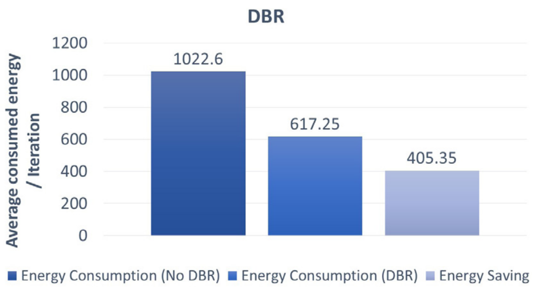


(b) Average energy consumption per emulation iteration by applying the RBA algorithm and the energy savings achieved.

Fig. 11. Emulation results of applying the RBA algorithm.



(a) Average execution time per emulation iteration by applying the DBR algorithm and the execution time savings achieved.



(b) Average energy consumption per emulation iteration by applying the DBR algorithm and the energy savings achieved.

Fig. 12. Emulation results of applying the DBR algorithm.

makespan if there is a plenty of bandwidth dedicated to one VM (or more), or the VMs execution continues with their pre-allocated bandwidth. If the VMs execution continues without any bandwidth reallocation, then the improvement ratio is zero, otherwise it will be more than zero with a specific enhancement value. The enhancement values vary due to the following: different VMs and PMs configurations, and different workloads served by the VMs as each workload has different requirements.

In the micro data centers environments, where most of the applications are real-time and need fast response, bandwidth slicing framework is very significant as it speeds up the execution of the VMs which serve the applications in the micro data center, in addition to the contribution in energy savings.

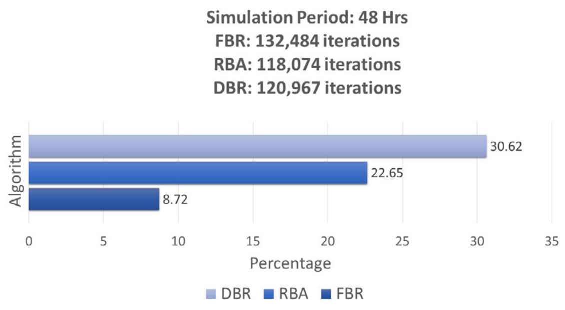


Fig. 13. Improvement Ratio (IR) achieved by applying the FBR, RBA, and DBR algorithms.

7. Conclusion

This work presents a bandwidth slicing approach that regards bandwidth as a shared resource whose effective management can enhance the performance and energy efficiency of cloud nodes. Bandwidth slicing can be defined as the ability to orchestrate the capacities of the bandwidth, as a physical resource, among different VMs co-hosted on the same host. The proposed bandwidth slicing approach, capitalizes on the hypervisor's abilities in reconfiguring the VMs configurations to match the needs of the applications served by the VMs co-hosted on the same PM. This approach can offer several benefits by enhancing the ability of providers and operators to deploy, when possible, only the specific amount of bandwidth needed to serve specific use cases and users. Adopting bandwidth slicing enables providers and operators to provide service differentiation, which is most desirable because cloud providers usually serve an unprecedented diversity of users and applications. Three algorithms are proposed to meet the goals of the bandwidth slicing approach: The FBR algorithm enforces a fair reallocation for the bandwidth amounts among VMs based on the requirements of the jobs executed on these VMs, The RBA algorithm allocates only the required amount of bandwidth to the VMs at a specific time, and the DBR algorithm divides the bandwidth of the terminated VM among the other VMs hosted on the same PM. The FBR, RBA, and DBR algorithms provided promising results for enhancing the overall PMs performance, leading to significant reduction in the consumed energy in the cloud data centers, and in the micro data centers as well.

The simulation results demonstrated major improvements in execution time and energy consumption reaching up to 30% improvement ratio. These simulation results shed light on the importance of directing further research towards bandwidth slicing and reallocation as a means for improving the performance of the data centers' nodes. Also, the results present a call for action for more and more research into bandwidth slicing and reallocation as a viable complement to other energy-saving techniques for enhancing data center energy consumption.

As a future extension, we are working on integrating the proposed algorithms with some of the external network algorithms from the literature in a real cloud environment. Moreover, on the same front, we are researching the different prominent cloud platforms to come up with a set of blueprints to integrate the BW slicing algorithms proposed in this paper to their data center architecture.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

A. Al-Dulaimy's and J. Taheri's contributions were partially funded by the Knowledge Foundation of Sweden through the HITS Profile.

References

[1] B. Hong, V. Prasanna, Bandwidth-aware resource allocation for computing independent tasks in heterogeneous computing systems to maximize throughput, in: The 15th Annual International Conference on Parallel and Distributed Computing and Systems, 2003.
[2] J. Sonnek, J. Greensky, R. Reutiman, A. Chandra, Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration, in: The 39th International Conference on Parallel Processing, 2010.
[3] C. Wang, C.R. Wang, Y. Yuan, Dynamic bandwidth allocation for preventing congestion in data center networks, in: The 8th International Symposium on Neural Networks, 2011.
[4] O. Beaumont, A. Legrand, Y. Robert, Scheduling divisible workloads on heterogeneous platforms, J. Parallel Comput. 29 (2003) 1121–1152.
[5] O. Beaumont, N. Bonichon, L. Eyraud-Dubois, Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model, in: The 22nd IEEE International Symposium on Parallel and Distributed Processing, 2008.
[6] W. Lin, C. Liang, J.Z. Wang, R. Buyya, Bandwidth-aware divisible task scheduling for cloud computing, J. Softw.: Pract. Exp. 44 (2) (2014) 163–174.
[7] D. Shen, J. Luo, F. Dong, J. Zhang, AppBag: Application-aware bandwidth allocation for virtual machines in cloud environment, in: The 45th International Conference on Parallel Processing, 2016.
[8] F. Liu, J. Guo, X. Huang, J. Lui, eba: Efficient bandwidth guarantee under traffic variability in datacenters, IEEE/ACM Trans. Netw. 25 (1) (2016).
[9] K. To, J. Padhye, G. Varghese, D. Firestone, Measurement based fair queuing for allocating bandwidth to virtual machines, in: The 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization, 2016.
[10] J. Cao, X. Zhu, F. Dong, B. Liu, Z. Ma, H. Min, Time series based bandwidth allocation strategy in cloud datacenter, in: The International Conference on Advanced Cloud and Big Data, 2016.
[11] L. Li, Y. Shi, J. Wang, Z. He, A VM-friendly NIC architecture for cloud computing, in: The 2nd IEEE International Conference on Cloud Computing and Big Data Analysis, 2017.
[12] H. Yu, J. Yang, H. Wang, H. Zhang, Towards predictable performance via two-layer bandwidth allocation in cloud datacenter, J. Parallel Distrib. Comput. 126 (2019) 34–47.
[13] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, H. Wu, Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud, in: The 35th Annual IEEE International Conference on Computer Communications, 2016.
[14] Mohamed Abdel-Basset, Laila Abdle-Fatah, Arun Kumar Sangaiah, An improved Lévy based whale optimization algorithm for bandwidth-efficient virtual machine placement in cloud computing environment, Cluster Comput. 22 (2019) 8319–8334.
[15] N. Mehdi, A. Mamat, H. Ibrahim, S. Syrmabn, Virtual machines cooperation for impatient jobs under cloud paradigm, Int. J. Inf. Commun. Eng. 7 (1) (2011) 13–19.
[16] J. Guo, F. Liu, H. Tang, Y. Lian, H. Jin, J.C. Lui, Falloc: Fair network bandwidth allocation in IaaS datacenters via a bargaining game approach, in: The 21st IEEE International Conference on Network Protocols, 2014.
[17] J. Guo, F. Liu, J. Lui, H. Jin, Fair network bandwidth allocation in iaas datacenters via a cooperative game approach, IEEE/ACM Trans. Netw. 24 (2) (2016) 873–886.
[18] A. Al-Dulaimy, W. Itani, R. Zantout, A. Zekri, The effect of bandwidth allocation on power efficiency in cloud data centers, in: The 11th International Computer Engineering Conference, 2015.
[19] Mostafa Ghobaei-Arani, Alireza Souri, Ali Rahmanian, Resource management approaches in fog computing: A comprehensive review, Grid Comput. 18 (2020) 1–42.
[20] Description of Network Slicing Concept, White paper, Next-Generation Mobile Networks (NGMN) Alliance, 2016.
[21] Xin Li, Mohammed Samaka, H. Anthony Chan, Deval Bhamare, Lav Gupta, Chengcheng Guo, Raj Jain, Network slicing for 5G: Challenges and opportunities, IEEE Internet Comput. 21 (5) (2017) 20–27.
[22] Abdelquoddouss Laghrissi, Tarik Taleb, Miloud Bagaa, Hannu Flinck, Towards edge slicing: VNF placement algorithms for a dynamic and realistic edge cloud environment, in: The IEEE Global Communications Conference, GLOBECOM, 2017.
[23] J. Mair, Z. Huang, D. Eyers, L. Cupertino, G.D. Costa, J. Pierson, H. Hlavacs, Power Modeling from Text Book: Large-Scale Distributed Systems and Energy Efficiency: A Holistic View, Wiley, 2015, pp. 131–156.
[24] A. Beloglazov, Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing (Ph.D. thesis), Department of Computing and Information Systems, The University of Melbourne, 2013.
[25] D. Weitzel, B. Bockelman, D. Swanson, Distributed caching using the htcondor cacheD, in: The Conference on Parallel and Distributed Processing Techniques and Applications, 2015.
[26] A. Al-Dulaimy, A. Zekri, W. Itani, R. Zantout, Paving the way for energy efficient cloud data centers: A type-aware virtual machine placement strategy, in: The IEEE International Conference on Cloud Engineering, 2017.
[27] A. Al-Dulaimy, W. Itani, R. Zantout, A. Zekri, Type-aware virtual machine management for energy efficient cloud data centers, Sustain. Comput.: Inf. Syst. 19 (2018) 185–203.
[28] X. Fan, W. Weber, L. Barroso, Power provisioning for A warehouse-sized computer, in: The 34th Annual International Symposium on Computer Architecture, 2007.
[29] E. Grochowski, M. Annavaram, Energy per instruction trends in Intel® Microprocessors, 2006, Technology@ Intel Magazine.
[30] Y.S. Shao, D. Brooks, Energy characterization and instruction-level energy model of Intel's Xeon Phi Processor, in: The International Symposium on Low Power Electronics and Design, 2013.
[31] Mininet, 2019, [Online]. Available: http://www.mininet.org. [Accessed 2019].
[32] Vmware, 2019, [Online]. Available: https://www.vmware.com/products/fusion.html. [Accessed 2019].
[33] R. Charles, J. Wilkes, J. Hellerstein, Google Cluster-Usage Traces: Format+Schema, Google Inc., 2011.

Auday Al-Dulaimy is a Postdoctoral researcher at the Department of Mathematics and Computer Science, Karlstad University, Sweden. He received the B.Sc and M.Sc degrees in Computer Science from Al-Nahrain University, Iraq in 2000 and 2003 respectively; and the PhD degree in Computer Science from Beirut Arab University, Lebanon in 2017. Auday has many publications in the international conferences, journals, and book chapters. Auday is the recipient of many academic awards. He is a reviewer in many journals, and has served as a member of the program and scientific committees in several international refereed conferences. He is a member of IEEE. His research interest includes Distributed Systems, Cloud Computing, Internet of Things, and Edge Computing.

Wassim Itani received his B.E. in electrical engineering, with distinction, from Beirut Arab University (BAU) in 2001, and the M.E. and Ph.D. degrees in electrical and computer engineering from the American University of Beirut (AUB) in 2003 and 2011 respectively. Currently he is an Associate Professor at the Department of Computer Science, University of Houston-Victoria (UHV), USA. Before joining UHV, Wassim worked as an Associate Professor of electrical and computer engineering at BAU from 2012 to 2019. During this period he directed the Center for Entrepreneurship and the Center for Continuing Education. Wassim has over 50 publications in conferences and journals. His research interests include cloud computing security protocols and cryptographic protocols performance evaluation. Wassim is a member of IEEE, and member of Beirut Order of Engineers and Architects (OEA).

Javid Taheri is a Professor at the Department of Computer Science, Karlstad University, Sweden. He received his PhD in Mobile Computing from University of Sydney (Australia) in 2007, and his Bachelor and Masters of Electrical Engineering from Sharif University of Technology, Tehran (Iran) in 1998 and 2000, respectively. He is the recipient of many awards including being selected as one of the top 200 young researchers in the world by the Heidelberg Forum in 2013, the recipient of several best paper awards since 2007, and the recipient of the prestigious IEEE Middle Career Researcher award from TSCS in Scalable Computing in 2019. He holds several cloud/networking related industrial certification from VMware, Cisco, Microsoft and IBM. His research interests includes Cloud Computing, Edge/Fog Computing, Network Function Virtualization, Software-defined Networking, and AI-based optimization techniques. He is the editor of a book entitled "Big Data and Software Defined Networks", and is currently co-editing another book entitled "Edge Computing: Models, Technologies and Applications". He coauthored >160 scientific articles and papers, has been serving as an editor for >15 journals, as well as a member of organizing team for >30 international conferences.

Maha Shamseddine received her B.E. in electrical engineering from Beirut Arab University (BAU) in 2001, the M.E. in computer and communications engineering from the American University of Beirut (AUB) in 2003, and the Ph.D. in Electrical and Computer Engineering from AUB in 2018. She is currently an Assistant Professor in electrical and computer engineering at BAU, Lebanon. Her research work is in cloud computing, network virtualization, softwarization, and machine learning. Maha is a member of Beirut Order of Engineers and Architects (OEA).