Effective Live Migration of Virtual Machines Using Partitioning and Affinity Aware-Scheduling

Anis Yazidi, Frederik Ung, Hårek Haugerud, Kyrre Begnum

Department of Computer Science,
Oslo Metropolitan University,
Oslo, Norway.

Abstract

During maintenance and disaster recovery scenarios, Virtual Machine (VM) inter-site migrations usually take place over limited bandwidth—typically Wide Area Network (WAN)—which is highly affected by the amount of inter-VM traffic that becomes separated during the migration process. This causes both a degradation of the Quality of Service (QoS) of inter-communicating VMs and an increase in the total migration time due to congestion of the migration link. We consider the problem of scheduling VM migration in those scenarios. In the first stage, we resort to graph partitioning theory in order to partition the VMs into groups with high intra-group communication. In the second stage, we devise an *affinity*-based scheduling algorithm for controlling the order of the migration groups by considering their inter-group traffic. Comprehensive simulations and real-life experimental results show that our approach is able to decrease the volume of separated traffic by a factor larger than 30%.

Keywords:

Live Migration, Graph Partitioning, Migration Scheduling, Separated Traffic.

1. Introduction

Modern computing infrastructures run on virtual platforms. A prominent form of virtualization enables a complete and fully usable *operating system* (OS) to run in virtualized form. This principle is commonly referred to as *OS virtualization*. Most of the state-of-the-art data centers use virtualization technology to provide flexibility and simplicity for their customers in terms of provisioning and managing VMs. This concept is referred to as *cloud computing*, and it is expected to be one of the most essential aspects of future computing. Flexibility and scalability are important characteristics of this form of computing.

Live migration is a unique feature of cloud computing that makes it possible to move VMs between different physical locations, without having to shut them down. This feature minimizes the Service Disruption Time (SDT) compared to

Email address: anis.yazidi@hioa.no (Anis Yazidi)
Frederic Ung is a former student at Oslo Metropolitan University.

the case of offline migration, where the VM is shut down instead, migrated and then powered on again. SDT refers to the time period where a VM is unavailable due to being suspended at source, and not yet up and running at the target destination. The vast majority of virtualization platforms, such as Xen, VMware (VMotion), and Hyper-V support live migration. Server consolidation, load balancing and system maintenance are among the most popular use-case scenarios of live migration [1, 2]. Load balancing refers to the principle of distributing the computational load evenly between physical nodes. In a cloud environment, roughly speaking, this takes place by live migrating VMs from overloaded physical nodes to underloaded physical nodes [3]. In the case of system maintenance, where Physical Machines (PMs) regularly need hardware upgrades or replacement of failing components, live migrations can be used to migrate running VMs from a PM that needs to be powered off.

Live migration can be used for the purpose of server consolidation, where the aim is to use as few PMs as possible by migrating VMs from lightly loaded PMs in order to power those PMs off and thus save energy [2, 4]. This would lead to more "tightly packed" consolidated PMs. Live migration has also been used to enable greener computing by migrating VMs to data centers powered by renewable energy based on the intermittent availability of such green energy. This paradigm is referred to as the *follow the wind, follow the sun* paradigm [5].

In order to avoid noticeable service degradation, the process of migrating VMs should take as little time as possible, preferably in the order of a few seconds [6].

In this paper, we consider the problem of inter-site migrations of "chatty" VMs over limited bandwidth. The goal of our research is to reduce the volume of separated traffic that might arise when chatty VMs become separated during the migration process. In order to solve the problem, we first use graph-partitioning algorithms to identify groups of intensively inter-communicating VMs that should not be separated during the process of live migration. The deployed graph-partitioning algorithm is based on the theory of adaptive learning and, more particularly, on Learning Automata (LA) [7]. VMs within the same group are migrated in parallel. However, there might still be some amount of traffic between the different groups of VMs. Thus, in the second phase, we devise a greedy scheduling algorithm that decides the order of migration of the different groups using a novel concept called network affinity.

1.1. Contributions

We crystallize the contributions of the article as follows:

- We devise an affinity-aware algorithm to schedule the migration of VMs for the purpose of minimizing the volume of separated traffic. The affinity-aware scheduling algorithm is based on the observation that, whenever a VM is migrated

to a destination PM, there is a resultant gain in terms of traffic that goes through memory[2] and a down-cost in terms of separated traffic. The affinity scheduling algorithm provides a near optimal solution to an NP hard scheduling problem if minimizing the amount of separated traffic is the criterion.

- We provide insights into the real-life implementation of our solution that will help interested researchers to replicate our results and test new solutions.
- We present a novel application of the theory of LA to the problem of scheduling inter-site migration of VMs. We hope that the current work will fuel research interest in applying the theory of LA in the field of cloud computing.

1.2. Structure of the article

The remainder of the article is structured as follows. In Section 2, we provide useful background on live migration and review prominent related research in order to render the article self-contained. In Section 3, we present our approach to solving the problem, which consists of two main stages: partitioning VMs and affinity-aware scheduling. In Section 4, we present the results of our test-bed and simulations. Section 5 concludes the article and outlines future research directions that are worth pursuing.

2. Background and Related Work

2.1. Live migration

Current hypervisors in virtual cloud environments include different functionality for migrating virtual machines. Migration is performed either *sequentially* or in *parallel*. The sequential method migrates one VM at a time. In parallel migration, multiple VMs assigned to the same migration task are moved simultaneously.

2.1.1. Pre-copy Migration

The most common way of carrying out virtual machine migration is the *pre-copy method*. During such a process, the complete disk image of the VM is first copied over to the destination. If anything is written to the disk during this process, the changed disk blocks are logged. Next, the changed disk data are migrated. Disk blocks can also change during this stage, and, once again, the changed blocks are logged. The migration of changed disk blocks is repeated until the generation rate of changed blocks is lower than a given threshold or until a certain number of iterations have taken place. After the virtual disk is transferred, the RAM is migrated, using the same principle of iteratively copying changed content. Next, the VM is suspended on the source machine, and resumed on the target machine. The states of the virtual processor are also copied

[2]This concerns the communication traffic with the VMs located at the destination PM

over, ensuring that the machine is exactly the same in both operation and specifications once it has been migrated to the destination. The rate at which disk or memory changes during the migration is referred to as the *Dirty Rate* (DR). It is important to note that the disk image migration phase is only necessary if the VM does not have its image on a network location, such as an NFS share, which is quite common in data centers. Figure 1 depicts the different phases of the pre-copy live migration method.

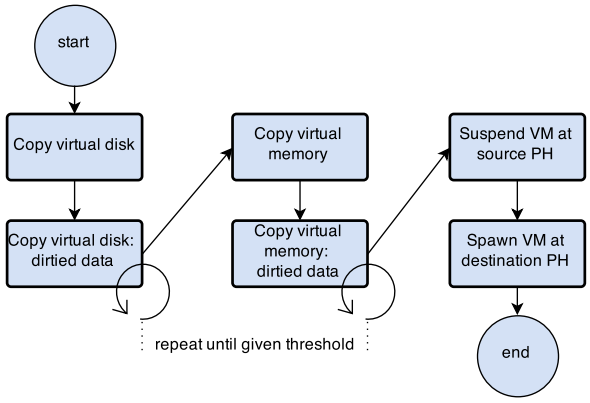

Figure 1: Pre-copy method for live migration

2.1.2. Post-copy Migration

Post-copy migration is the most primitive form of virtual machine migration [8]. The basic outline of the post-copy method is as follows. The VM is suspended at the source PM. The minimum required processor state, which allows the VM to run, is transferred to the destination PM. Once this is done, the VM resumes running at the destination PM. This first part of the migration is common to all post-copy migration schemes. Once the VM resumes running at the destination, memory pages are copied over the network as the VM requests them, and this is where the post-copy techniques differ. The main goal in this latter stage is to push the memory pages of the suspended VM to the newly spawned VM, which is running at the destination PM. In this case, the VM will have a short SDT, but a long Performance Degradation Time (PDT).

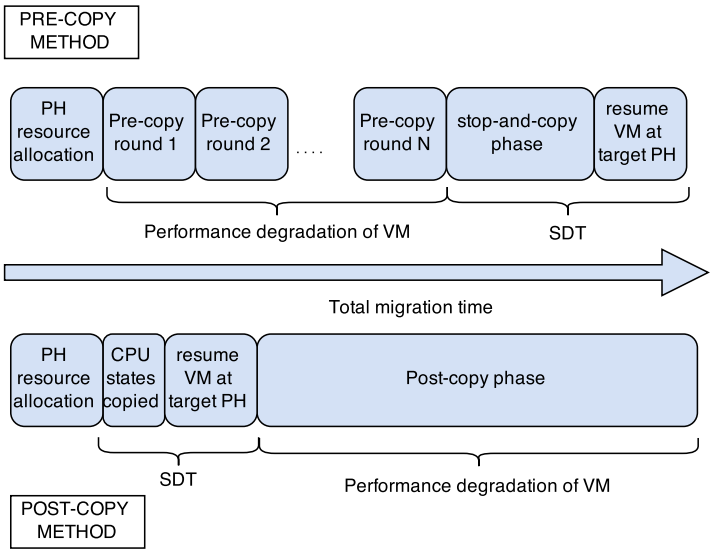

Figure 2: Pre- vs. Post-copy migration sequence

Figure 2 illustrates the difference between the two migration techniques. The diagram only depicts memory and CPU state transfers, and not the transfer of the disk image of the VM. The latter is performed in a similar manner in both the migration techniques, and does not affect the performance of the VM. It is therefore disregarded in the comparison. The "performance degradation of VM" in the pre-copy method is a consequence of the hypervisor having to keep track of the dirty pages, the RAM that has changed since the last pre-copy round. In the post-copy scenario, the degradation is greater and lasts longer. In essence, the post-copy method activates the VMs at the destination faster, but all memory is still located at the source. When a VM migrated using the post-copy method requests a specific portion of memory not yet local to the VM, the relevant memory pages will have to be pushed over the network. The "stop-and-copy" phase in the pre-copy method is the period during which the VM is suspended at the source PM and the last dirtied memory and CPU states are transferred to the destination PM. SDT is the period during which the VM is inaccessible.

2.1.3. Issues with live migration

Moving virtual machines between physical hosts introduces a set of challenges that we will briefly review in this section.

In a cloud environment, a multi-tier application is an application that involves many inter-communicating VMs. They are typically configured with the different functionality spread over multiple VMs [9]. For example, the database part of an application might be stored on one set of VMs, and the web server functionality on another set. In a scenario where an entire application is to be moved to a new site that has a limited bandwidth network link to the original site, the application will experience deteriorated performance during the migration period for the following reason. If one of the application's member VMs resumes running at the destination site, any data exchange destined for that machine might experience more delay

than usual due to the limited inter-site bandwidth and because the rest of the application is still running at the source site. Several researchers have proposed different approaches to remediating the problem of geographically split VM traffic during migration. Following the same nomenclature as in [10], this paper will refer to the latter problem as the *split components problem*.

Within a data center, it is common to use dedicated *management links*, which enable management operations such as live migrations to proceed on different links than the traffic generated by the applications running on the VMs. The occurrence of some SDT is unavoidable, but it is less than in the case where migration takes place over non-dedicated links. However, such implementation could be costly. In a setting where management links are absent, live migrations would directly affect the total available bandwidth on the links they use. One issue that could arise as a result is that several migrations could end up using the same migration paths, effectively overflowing one or more network links, and hence slowing the performance of multi-tiered applications.

In a scenario where a system administrator needs to shut down a physical machine for maintenance purposes, all the VMs currently running on that machine will have to be moved, so that they can continue to serve the customers. In such a scenario, it would be favorable if the migration took as little time as possible.

2.2. Related Work

In [11], the authors designed a system called CQNCR (reads "sequencer"), with the goal of enabling a planned migration to take place as quickly as possible, given a source and target destination of the VMs. CQNCR deals with intra-site migrations. The authors state that their approach is able to significantly increase the migration speed, reducing the total migration time by up to 35%. They also introduce the concept of *Virtual Data Centers* (VDCs) and *residual bandwidth*. In practical terms, a VDC is defined as a logically separated group of VMs and their associated virtual network links. Since each VM has a virtual link, that link also has to be moved to the target PM. When this occurs, the bandwidth available to the migration process changes. The CQNCR-system takes this continuous change into account and provides an algorithm that results in efficient bandwidth usage.

Another related system, COMMA [10], groups VMs together and migrates one group at a time. A group consists of VMs that have a significant amount of internal communication. After the migration groups are decided, the system performs inter- and intra-group scheduling. The former is about deciding the order of the groups, while the latter optimizes the order of VMs within each group. The goal of COMMA is to address the issue mentioned in Section 2.1.3. The main goal of COMMA is to migrate associated VMs at the same time, in order to minimize the amount of traffic that has to go through a slow network link. The system is therefore especially suitable for inter-site migrations. It is structured so that each VM has a process running, which reports to a centralized controller that performs the calculations and scheduling.

The COMMA system defines the network impact as the amount of inter-VM traffic that becomes separated because of migrations. In a case where a set of VMs, $\{VM_1, VM_2, ..., VM_n\}$, is to be migrated, the traffic levels running between them are measured and stored in matrix TM. Let the migration completion time for VM_i be t_i. Equation 1 gives the network impact that that should be minimized.

$$impact = \sum_{i=1}^{n} \sum_{j>i}^{n} |t_i - t_j| \cdot TM[i,j] \tag{1}$$

In the rest of this paper, we will use the words network impact and volume of separated traffic interchangeably to refer to the quantity in Equation 1. Please note that the rate of separated traffic is different from the volume of separated traffic since the formula takes into account the time during which the VMs are physically separated. The VMbuddies system [12] also addresses the challenges of migrating VMs that are part of multi-tier applications. The authors formulate the problem as a *correlated VM migration problem* in multi-tier applications. Correlated VMs are machines that work closely together, and therefore exchange a lot of data. An example would be a set of VMs hosting the same application, where two or three VM subsets perform different roles in different tiers, as described in Section 2.1.3. The authors propose an algorithm for efficient use of the network bandwidth during migration, and a mechanism for reducing the cost of a live migration. The experiments conducted show clear improvements compared to current migration techniques, including a reduction of 36% in migration time, compared to Xen.

A system called Clique Migration [13], also migrates VMs based on their level of interaction. It is specialized for inter-site migrations. When Clique migrates a set of VMs, the first operation it performs is to analyze the traffic patterns between them and try to profile their mutual traffic affinities. This is similar to the COMMA system. It then proceeds to create groups of VMs. All VMs within a group will be scheduled for migration at the same time. The order of the groups is also calculated to minimize the cost of the process. The authors define the migration cost as the volume of inter-site traffic caused by the migration. Because a VM will end up at a different physical location (a remote site), the VM's disk is also transferred along with the RAM. A closely related problem to creating "clique" of VMs is the task of clustering VMs that exhibit similar behavior in terms of resource usage [14]. Roughly speaking, clustering can be used for consolidation purposes, for example by consolidating VMs that exhibit complementary resource usage patterns on the same PM.

A *Time-bound thread-based Live Migration* (TLM) technique was proposed in [6]. The focus was on handling large migrations of VMs running RAM-heavy applications by allocating additional processing power at the hypervisor level to the migration process. TLM can also slow down the operation of such instances to reduce their dirty rate, which will help to reduce the total migration time. The completion of a migration in TLM always takes place within a given time period that is proportional to the RAM size of the VMs. The idea behind TLM differs from the other previously mentioned techniques,

in that it actually changes the behavior of running instances if necessary. Slowing the operations of VMs will, of course, subsequently decrease the performance of the applications hosted by them.

All the aforementioned solutions migrate groups of VMs simultaneously, in one way or another, hence utilizing parallel migration to reduce the total migration time. According to [1], it was observed that, when running parallel migrations within data centers, an optimal sequential approach is preferable. The authors have implemented a migration system called vHaul for this purpose. vHaul is optimized for migrations within data centers that have dedicated migration links between physical hosts.

In [15], Deshpande et al. introduce a novel non-standard migration method called scatter-gather VM migration. The migration is done via an intermediate node. The main idea is to push the state of the memory of the source host to one intermediate node in the network. At the same time, the destination hosts retrieve the memory state from the intermediate host by using a modified version of the post-copy VM migration.

2.3. Minimum-cut graph partitioning

The graph-partitioning problem [16] is defined as a graph composed of *vertices* and *edges* divided into smaller parts with certain properties. The goal of a minimum cut, when applied to a weighted graph, is to cut the graph across the vertices in the way that leads to the smallest sum of weights. The resulting subsets of the cut are not connected after this. One application of graph partitioning could be to group inter-communicating VMs together in such a way that the VMs with a high degree of inter-communication traffic are placed together. In this case, the VMs can be modelled as vertices, while the amount of their mutual communication corresponds to the the edges in the graph. Figure 3 shows an example of inter-communicating VMs where the communication pattern is modelled as graph. The "weight" in the illustration could, for example, represent the average amount of traffic between two VMs in a given time interval. This can be calculated for the entire network, so that every network link (edge) would have a value. The dashed line represents a possible way of cutting the network into two parts using a *minimum cut*. The cut must go through the entire network, effectively crossing edges so that the output consists of two disjoint subsets of nodes. Minimum cut is achieved if the sum of the weights of the crossed edges is minimized.

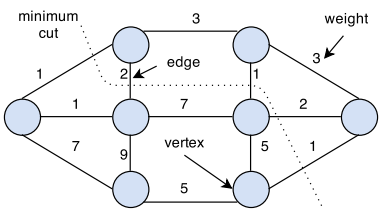

Figure 3: Minimum cut in a weighted graph

However, if the VMs belonging to the same cut or group are marked for simultaneous migration, while the sum of the

their dirty rates is greater than the bandwidth available for the migration task, the migration will fail to converge [10]. It is therefore imperative to divide the network into smaller groups of VMs, so that each group can be migrated successfully. In a similar problem called the *uniform graph-partitioning problem*, the number of nodes belonging to each group has to be equal. The problem is known to be NP-complete [17].

In order to give some idea of the dramatic complexity of the graph-partitioning problem and its exponential increase as the size of the graph increases, we will consider two networks where one has 10 nodes and the other has 100. The number of possible cuts, and hence the solution space, is 126 in the former case, and 10^{29} in the latter [17]. This clearly shows that a brute force approach is unfeasible even for relatively small graphs.

LA [7] is a field of research concerned with adaptive decision-making in random environments. According to the theory of LA, the environment offers a set of actions that the learning mechanism has to choose between. The aim is to converge to an optimal action using trial and error. The response from the environment to choosing an action is generally speaking binary: either a reward or a penalty. However, other types of continuous response than the latter binary feedback can be defined.

In [17], Oommen and Croix proposed a learning algorithm based on OMA for splitting any graph into equally sized subgroups, where the result is such that the sum of the edges that go between the subgroups is as small as possible. In other words, the proposed algorithm ensures that a minimum cut has been reached between any two resulting subgroups of the input graph. It is worth mentioning that the OMA algorithm, which is a known algorithm developed by Oommen and Ma [18], allows for a large number of application domains, including graph partitioning. The way in which the OMA algorithm has been adapted by Oommen and Croix [17] to solve the graph-partitioning problem relies on introducing the notion of similarity between two nodes of the graph based on the weight of their connecting edge. Two nodes are deemed similar if the weight of the edge connecting them is above a certain similarity threshold. Similarly, two nodes are deemed dissimilar if the weight of their edge is below a dissimilarity threshold. Note that, by design, the similarity threshold is superior to the dissimilarity threshold.

3. Our Solution: Cost-effective and Affinity-Aware Live Migrations of Virtual Machines Using OMA

In this section, we present our approach to solving the problem. This section is organized as follows. First, in Section 3.1, we provide some terminology required for the design of our solutions. Then, we formally define the time needed for the migration in Section 3.3. In Section 3.4, we present our scheduling algorithm.

3.1. Traffic Terminology

A traffic matrix will be used to represent the mutual traffic between any two pairs of VMs. The diagonal terms are zero, since "loopback" traffic on the machines in not considered. Let n be the total number of VMs to be scheduled for migration. The indexes i and j refer to the column and row numbers, respectively. We could then end up with a $n \cdot n$ matrix in the form $VM_{i,j}$ denoting the amount of traffic between VM i and VM j:

$$\begin{bmatrix} 0 & VM_{1,2} & ... & ... & VM_{1,n} \\ VM_{2,1} & ... & ... & ... & VM_{2,n} \\ ... & ... & ... & ... & ... \\ ... & ... & ... & 0 & ... \\ VM_{n,1} & VM_{n,2} & ... & ... & 0 \end{bmatrix} \tag{2}$$

In this paper, we consider both dedicated management network links and non-dedicated links. For the latter, the migration traffic will have to share the bandwidth with VMs that are communicating with each other from different hosts, which means that both the VMs' dirty rates and shared traffic will affect migration performance. It is likely that the advantage of our migration solution will be more evident when migration traffic and inter-VM traffic are running on the same link. This is because the grouping of the VMs is done using the OMA algorithm, so that only inter-group traffic is exchanged over the migration link during the migration process.

In order to simplify the design of our algorithm, we chose to use the average traffic between VMs over a relatively large time interval that spans a few hours. Some evidence from the literature [19] supports such a choice, as observed by Meng et al.: "the rate for a large proportion of VMs is found to be relatively stable" over time intervals varying between 30 mn and 4 hours. Therefore, our decision to base our algorithm on the average traffic rates is considered realistic in many real-life scenarios. In practice, however, in order to exactly and precisely compute the volume of separated traffic, we would need full knowledge of instantaneous traffic rates during the whole period of the migration process, i.e., from the time when the first VM is migrated to the destination until the time when the last VM in the source has reached the destination machine. Devising a more sophisticated algorithm that takes into account possible changes in the VM traffic matrix over time instead of merely considering the average traffic rate is a relatively challenging task. In fact, such a sophisticated algorithm needs to *predict* the instantaneous VM traffic rates over the whole period of the experiment in order to yield optimal results.

3.2. Remark on the choice of the graph-partitioning algorithm

There are numerous graph-partitioning algorithms available in the literature [16], which might result in unbalanced groups of VMs that we could have used in this study instead of the OMA algorithm. In this paper, we opted to use the

graph-partitioning algorithm developed by Oommen and Ma [18]. The algorithm is easy to implement and was shown to be computationally efficient in the literature. The algorithm focuses on minimizing the cross communication between groups and maximizing internal communication within the same group. However, by default, the algorithm results in equally sized groups, with the number of groups being a user-defined input. The reason why the resulting groups are equal in size is intrinsic to the OMA itself. In fact, in the OMA algorithm, pairs of nodes are only allowed to swap groups until convergence of the OMA algorithm. Consequently, the sizes of the groups are kept constant along the iterations of the algorithm. Furthermore, the OMA algorithms are still able to function even when the number of VMs is not divisible by the number of groups. The important question that needs to be answered is how to determine the appropriate size of a group. In this perspective, an important factor that should be taken into account is the cumulative dirty rate in the resulting group. Creating too large groups is clearly not feasible. The reason for this is that, in order to ensure that the migration process converges and does not result in a deadlock, the aggregate dirty rate of a group to be scheduled for migration should not be higher than the residual bandwidth. Furthermore, creating too small groups results in increased cross communication between the groups, and thus an increased amount of separated traffic. There is thereby a trade-off when deciding the size of a group in which the aggregated dirty rate of the group is a key parameter. In a future study, we would like to investigate a more sophisticated approach to the partitioning phase that takes into account the aggregate dirty rate of the group when deciding its size. Moreover, in future work, we would like to omit the constraint that the group sizes need to be equal as a consequence of resorting to the OMA algorithm.

3.3. Migration time

In a two separate studies, Bari et al. [11] and Mann et al. [20] use the following mathematical formulas to calculate the time it takes to complete the different parts of the migration. Let W be the disk image size in megabytes (MB), L the bandwidth allocated to the VM's migration in MBps and T the predicted time in seconds. X is the amount of RAM that is transferred in each of the pre-copy iterations.

The time it takes to copy the image from the source PM to the destination PM is:

$$T_i = W/L \tag{3}$$

Once the VM's image is copied over, the pre-copy phase is initiated. Its time duration can be calculated as follows:

$$T_{p+s} = \frac{M \cdot \frac{1-(R/L)^N}{1-(R/L)}}{L} \tag{4}$$

The stop-and-copy period is the last phase of a pre-copy live migration, where a VM is suspended at the source PM and

resumes running at the destination PM. The completion time for this final phase is given by:

$$T_s = M/L \cdot (R/L)^N \tag{5}$$

The N in the Equations 4 and 5 is given by:

$$N = \min(\lceil \log_{R/L} \frac{T \cdot L}{M} \rceil, \lceil \log_{R/L} \frac{X \cdot R}{M \cdot (L - R)} \rceil) \tag{6}$$

We provide the following formulas [12] to describe the total amount og network traffic and the total migration duration, respectively. The number of iterations in the pre-copy phase (N) is not defined here, but is calculated based on a given threshold.

Variable	**Description**
V	Total network traffic during migration
T	Time it takes to complete migration
N	Number of pre-copy rounds (iterations)
M	Size of VM RAM
d	Memory dirty rate during migration
r	Transmission rate during migration

Table 1: Variables used in formulas in the VMbuddies system

The expressions for v_i and t_i are given by:

$$v_i = \frac{M \cdot d^i}{r^i} \tag{7}$$

$$t_i = \frac{M \cdot d^i}{r^{i+1}} \tag{8}$$

Then the total network traffic during migration is:

$$V = \sum_{i=0}^{N} v_i = M \cdot \sum_{i=0}^{n} \frac{d_i}{r_i} \tag{9}$$

Table 1 denotes the variables used in Equation 9.

3.4. Subgroup scheduling based on affinity concept

The first and most important part of the migration scheduling consists of applying the OMA algorithm in order to identify groups of VMs performing intensive inter-communication and thereby to mitigate the *split components problem* by migrating those VMs within the same group in parallel . The performance of a multi-tiered application will deteriorate because of the *split components problem*. In other words, the OMA will group VMs together in a manner that maximizes the *intra-group* traffic within each group. The solution proposed in this paper will therefore migrate VMs from one group at a time.

The order in which the groups of VMs are migrated can affect the total amount of separated traffic during the migration. In fact, there might still be some *inter-group* traffic. We propose a smart scheduling algorithm that decides which subgroup is to be migrated next at any time, until all subgroups (and hence all VMs) are running at the target PM. Coupled with the OMA algorithm, this should result in an efficient migration scheme, which has a low network impact. Please note that, in the event of migration failure, it is usually only VMs within the same group that were not migrated that need to be rescheduled. In such case, it is possible to recompute the graph-partitioning solution based on the remaining non-migrated VMs. Let G be subgroups, S and D be source and destination PM, respectively, and T the amount of inter-traffic between subgroups. For any two subgroups G_i and G_j, the exchanged traffic between these groups is the sum of the exchanged traffic between the VMs belonging to these two groups. In formal terms, this is defined as.

$$T(G_i \rightarrow G_j) = \sum_{VM_i \in G_i, VM_j \in G_j} VM_{ij} \tag{10}$$

Algorithm 1 migrates groups of VMs based on "affinity", which is a term that has been used to some degree by other researchers [21, 10]. High affinity between two VMs reflects the fact that the pair of VMs in question communicate intensively over the network.

According to our algorithm 1, we measure the amount of "net gain" in terms of the reduction in separated traffic by resorting to the concept of affinity, which is a novel and subtle way of using affinity. We will explain this more precisely in the following. The algorithm works in the following way. Initially, the list of already migrated VMs is empty, and all the VMs marked for migration are in the S list. The algorithm then enters a loop where, in each iteration, the subgroup with the highest affinity to the destination PM is migrated. The group is then removed from the list S and appended to list D.

The affinity of a VM to a physical machine measures the amount of inter-communication between a VM and the VMs located in that physical machine. The amount of gain measured as the reduction in the separated traffic achieved by moving a group G_i from a source machine to a destination is the difference between the affinity of the group to the destination server

Data: List of groups to be migrated: $S = \{G_1, G_2, ..., G_N\}$
Result: All VMs from each subgroup are migrated
$D = \emptyset$
while $S \neq \emptyset$ **do**
  for $G_i \in S$ **do**
    $T(G_i \rightarrow D) = \sum_{G_k \in D} T(G_i \rightarrow G_k)$
    $T(G_i \rightarrow S \setminus G_i) = \sum_{G_k \in S \setminus G_i} T(G_i \rightarrow G_k)$
    $\Delta_i = T(G_i \rightarrow D) - T(G_i \rightarrow S \setminus G_i)$
  end
  i_0 = First element of list S
  $i_{max} = i_0$
  $\Delta_{max} = \Delta_0$
  for i *in* $S \setminus i_0$ **do**
    if $\Delta_i > \Delta_{max}$ **then**
      $\Delta_{max} = \Delta_i$
      $i_{max} = i$
    end
  end
  Migrate $G_{i_{max}}$
  $D = D \cup G_{i_{max}}$
  $S = S \setminus G_{i_{max}}$
end

Algorithm 1: Affinity algorithm

$(T(G_i \rightarrow D))$, on the one hand, and the respective affinity to the source server ($T(G_i \rightarrow S \setminus G_i)$), on the other. In other words, it is the difference between the co-located traffic resulting from migrating G_i to the destination machine that goes through the memory[3] expressed as $T(G_i \rightarrow D)$ minus the traffic that goes through the network due to other VMs located in the source site S and communicating with the migrated group, which is expressed as $T(G_i \rightarrow S \setminus G_i)$.

It is worth mentioning that the latter "net gain", which corresponds to a difference of affinities is a novel concept that, to the best of our knowledge, has not been used before in the literature and that distinguishes between the traffic that goes through memory, and which is regarded as a gain, and the traffic that goes through the shared link and which is regarded as a loss (negative sign).

This continues until all groups have been migrated, i.e., until S is empty. Migrating the group with the highest affinity to the destination will intuitively reduce the amount of separated traffic. If a group with a high degree of affinity to the destination is not migrated for a long time, then VMs residing in this group will congest the network link between the source and destination PM during this period. As mentioned previously, according to our algorithm, the affinity here measures the amount of "gain" in terms of the reduction in separated traffic that is achieved by moving a group from a source machine to a destination.

Algorithm 1 will hereafter be referred to as the "Affinity algorithm". The amount of inter-site traffic changes after each group of VMs arrives at the destination. In Figure 4, we can imagine that subgroups $\{G3, G4, G5\}$ have been migrated and

[3]The latter quantity is the affinity to the destination machine. It is known that traffic between co-located VMs in the same physical machine goes through memory by default, and thus has no cost.

$\{G1, G2, G6\}$ are still running at the source PM. In this scenario, all subgroups G are transferred live to PM2, which means that either $G1$, $G2$ or $G6$ is the next to migrate. In this case, Algorithm 1 will compute the net gain when migrating $G1$, $G2$ or $G6$, and will then select the subgroup with the highest net gain. This subgroup will then be scheduled next for migration.

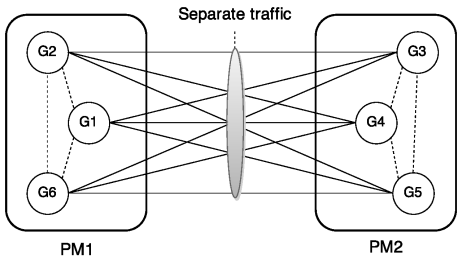

Figure 4: Separated inter-site traffic occurring during migrations

At this juncture, it is worth mentioning that an appropriate size of the group can be determined by running the graph partitioning using different group sizes, and then choosing the size that results in the least amount of separated volume of traffic by simulating the scheduling algorithm offline, i.e., Algorithm 1.

4. Experiences and Results

We will first present the test-bed on which the real-case experiments are conducted before presenting our real-life and simulation results.

4.1. The test-bed

A physical lab has been set up with the following specifications given by Table 2 :

Hardware / attribute	Details
Processing	Intel Core 2 Dual Core CPU @ 2.93 GHz
Memory	4 x 2048 MB DIMM @ 1066 MHz
Operating System	Ubuntu 14.04.2 (x64) LTS
Virtualization Solution	Libvirt, version 1.2.2
Storage	2 x 250 GB 7200 RPM hard drives
Networking	Interface 1 (eth0): up to 1000 Mb/s Interface 2 (eth1): up to 1000 Mb/s

Table 2: Physical lab hardware specifications

This is a paravirtualized environment running KVM QEMU emulator version 2.0.0. Figure 5 shows the interconnection between the VMs and the hosts. As we can see, the VMs that are spawned in this habitat are attached to a *virtual bridge*,

where a physical interface is connected to provide the migration capabilities. The dotted unidirectional arrows show the possible migration path of a node VM3 from PM1 to PM2. The non-dedicated migration path is also the path over which VMs communicate with each other, if they are located on different PMs.

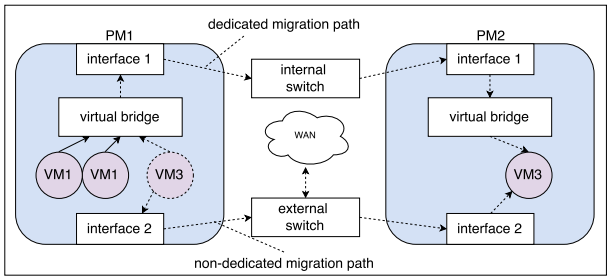

Figure 5: Physical lab

A small program is responsible for continuously generating the traffic among the hosts. Four different traffic rates have been set up. They are based on different *sleep timers* in the program that sends the data. The code snippet below illustrates how this works. The variable *sleep length* is based on input from the script that initiates the traffic on the VMs, which, in turn, is based on the traffic matrix sent as an input parameter to the script. *Message* is the variable containing the string that is transmitted on the network by UDP. *Level* is the parameter that indirectly determines the transmission rate, since the loop that sends traffic sleeps for a certain amount of time based on it.

```
while True:
    sock.sendto(str.encode(message),
    server_address)
    sleep(sleep_len)
```

The lowest sleep timer that can be set by the script is 0.05 seconds. This equals a bit rate of 0.216 mbit per second, which is given by $1350 \cdot (1 \div 0.05) * 8/1000000$, or $PacketSize \cdot (1 \div level) * 8 \div 1000000$. This rate will be the fastest any VM can communicate in the test environment, per connection. This is not a particularly fast data rate compared to today's standards. However, the most important thing is that the test environment can run different traffic levels, and that it is possible to distinguish them. With multiple VMs in a test, the amount of consumed network bandwidth will quickly become significant. The other three levels are 0.1, 0.072, and 0.054 mbit per second. The different levels in kbit/s are 216, 108, 72, and 54.

4.2. Experimental results

In this section, the real-case live migration results will be presented. Our preliminary testing in the physical lab suggests that the maximum number of VMs that could be hosted in the PM is around 20 running and communicating TinyCore

VMs at once. We decided to work with groups of 16 VMs. With this amount of running instances, the CPU consumption averaged around 80%, after traffic initiation.

Non-dedicated link scenario. Here, the results of migrations carried out over a non-dedicated migration link are outlined. The following experiment was run on the test-bed with the specifications listed in Table 3.

Attribute	**Value**
Amount of RAM	200 MB
Number of nodes	16
Group size	4
Highest traffic level	216 kbit/s
Migration link type	Non-dedicated
Migration link speed	1000 mbit/s

Table 3: Baseline test configuration

Figure 6 represents the sums of traffic, based on migration data from 11 tests. The first migration test was for our "calculated subgroup", which is the same migration sequence as "scheduled" in the previous experiment. When this is run multiple times, the level of separated traffic will be identical each time, as the same VMs will always spawn on the destination PM at regular time intervals. The ten other tests are for the random subgroups. The value presented is the average of these tests.

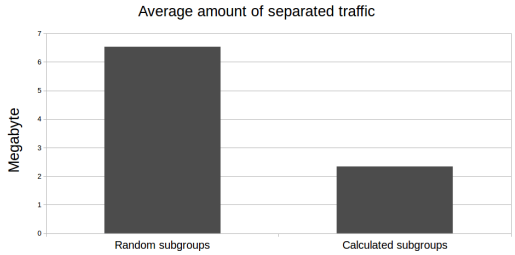

Figure 6: Network impact of migration random groups

This result shows that there is a clear benefit, in terms of separated traffic, of running the minimum-cut algorithm on the network graph. The random subgroups generated an average of 6.53 MB (52.24 mbit) of inter-site communication during migration, while our solution yields 2.34 MB (18.72 mbit). Random grouping creates 179% more traffic. It can be argued that the proposed solution is 64.16% more cost-effective in this scenario.

Dedicated link scenario. The following experiments have been conducted on a dedicated migration link. This means that any separated inter-VM traffic would not affect the migration, since it would not inhabit the migration link. The purpose of this test is to find the number of VMs to put in each migration subgroup in order to minimize the volume of separated traffic defined in formula (1). If the subgroup size is too large, the sum of the dirty rate in the group could exceed the available migration bandwidth. Although choosing a large group size would reduce the amount of separated VM traffic, it

would result in a failed migration, or a so-called "deadlock" [22].

A program was written to produce a certain amount of dirty rate on each VM. It works in a similar manner to the script that starts UDP traffic in the sense that it resorts to a sleep timer. When started, it copies pseudo-random bits to a file of a certain size, sleeps for a given time period and repeats. Controlling parameters such as *block size* and *count* (number of blocks) in the Linux program "dd", we could start a write operation at a specific dirty rate. As an example, when the script is initiated with a block size and block count of 100, and a sleep time of 1, 10,000 Bytes would be written in 1 second. From this, we can calculate what the dirty rate will be, by subtracting the time it takes to complete the write operation from this one second, and sleeping for the remaining period. This leads to a burst-like writing pattern, but it is sufficient to calculate the average dirty rate.

The only aspect of parallel migration that necessitates the determination of subgroup size is migration convergence (avoiding deadlock). The reason for deadlocks is the changing memory pages (dirty rate). The following experiment investigates the correlation between subgroup size and migration time, under varying dirty rates. A group of 16 VMs, each with 200 MB RAM, was migrated using parallel migration with two different subgroup sizes, 2 and 4, and four different dirty rate levels. The results are shown in Figure 7.

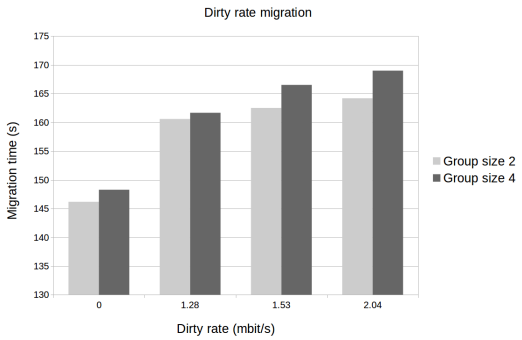

Figure 7: Effect of increasing the dirty rate on migration time

These VMs also sent traffic between each other, in addition to the dirty rate. The same matrix file as in the previous experiment was once again applied. Table 4 shows the volume of separated traffic calculated based on the VM communication matrix and on estimations of the migration time for each group.

Dirty rate (mbit/s)	**Subgroup size 2**	**Group size 4**
0	135626.4	84556.8
1.28	149040.0	92851.2
1.53	149040.0	96076.8
2.04	151275.6	96307.2

Table 4: Volume of separated traffic using different dirty rates

4.3. Simulation results

Since it is time-consuming and impractical to perform physical tests manually, a simulation script was used to facilitate all the variables needed to run virtual migrations, and to observe effects related to separated traffic. The simulation program takes as input the dirty rate of each VM, the available migration bandwidth, VM memory size, and the traffic matrix.

For all these simulations, we consider a 16 by 16 VM traffic matrix with high traffic levels between groups of four VMs. The simulation will demonstrate the effects of adjusting the parameters dirty rate and group size. In order to be able to compare these results with the actual physical migration, the memory size of each VM is set to 1600 mbit (200 MB) and the migration link to 100 mbit/s, as in the real test scenarios. The simulations have collected data for three cases:

- No dirty rate
- Dirty rate of 5 mbit per VM
- Dirty rate of 10 mbit per VM

All dirty rate cases are tested on subgroup sizes 2, 4 and 8.

Abbreviations used in the graphs are shown in Table 5:

Abbreviation	**Explanation**
LA	OMA has been used to decide the subgroups
AF	The affinity-based algorithm has been used to decide the subgroup migration order
RG	Random subgroups have been used

Table 5: Abbreviations of the different approaches

4.3.1. Simulation 1: no dirty rate

This first simulation is merely used as a baseline to understand the effects of changing the group size. Figure 8 shows that the optimal subgroup size for achieving low amounts of separated traffic is 4. It also shows that the best performance, based on the same metric, is achieved by a combination of the two implemented algorithms (LA+AF).

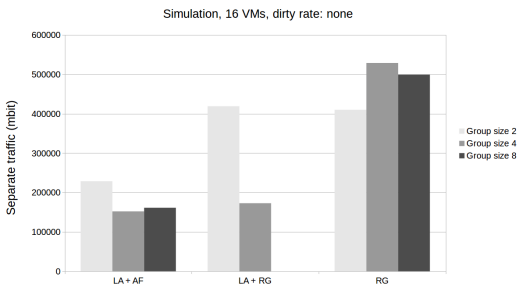

Figure 8: Volume of separated traffic with no dirty rate

4.3.2. Simulation 2: with dirty rates

Figure 9 shows the results when we fixed the dirty rate at n 5 mbit/s on each VM. The proposed solution still produces the best result with all the different subgroup sizes.

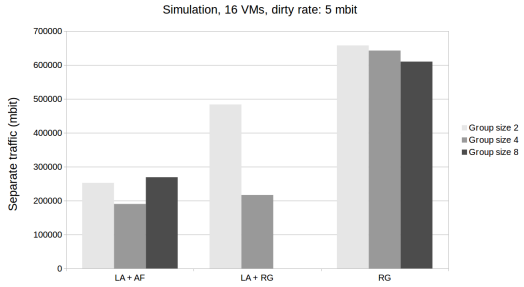

Figure 9: Volume of separated traffic with a dirty rate of 5 mbit/s

Figure 10 shows the impact of doubling the dirty rate to 10 mbit/s.

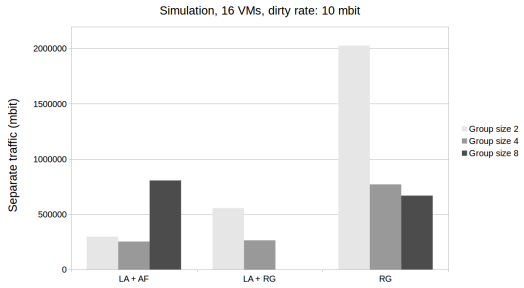

Figure 10: Volume of separated traffic with a dirty rate of 10 mbit/s

Simulation analysis.

According to the results of Section 4.2, the optimal group size is 4. Sizes 4 and 8 are comparable, but it can be concluded that a grouping of 2 is inefficient as it leads to more traffic separation. Moreover, when migrating only two nodes at a time *without* applying the affinity scheduling (hence random grouping), the separated traffic is more than doubled, which means that the affinity algorithm can double the cost-effectiveness in this case. Lastly, it can be observed that random grouping is ineffective.

When comparing Figure 9 and Figure 10 with the non-dirty rate scenario, it is clear that the optimal group size is 4, when both algorithms are used. We had also expected this as a consequence of the choice of communication matrix that we deployed in our experiments. In fact, the communication pattern is such that intra-communication within groups (each consisting of four VMs) is intense, while the inter-group communication between the groups is considerably less intense.

Focusing on this size, we also see that it is only marginally favorable to include the affinity algorithm. The LA+AF reduces cost by 4%, with the measurements being 253440 mbit for LA+AF and 264192 with random grouping.

When random groups of two VMs are migrated with the highest dirty rate, a massive amount of separated traffic occurs, which can be seen in Figure 10.

4.4. Affinity algorithm

In cases where VMs have high dirty rates, parallel migrations could be rendered impossible, since grouping is out of the question due to violation of the following constraint.

$$\sum_{VM_i \in subgroup} DirtyRate_i \leq Bandwidth \tag{11}$$

Such a scenario could occur when planning migrations between data centers located in different geographical areas, where the migration link is significantly slower than local management links.

So far, only the LA algorithm has been tested with or without the Affinity algorithm to decide the sequence. The simulation script was altered so that the affinity method, which is an implementation of the pseudo code in Algorithm 1, could be used in the simulation program by itself. It would be interesting to examine the performance of this algorithm in isolation. In this experiment, a list of migration-marked VMs is fed into the affinity algorithm, the output of which should be the recommended sequential migration sequence.

The traffic matrix our calculation is based on is the same as in the previous experiment. In order to determine the optimal sequence for single VM migration, we performed a brute force search among the 16! combinations where 16 is the number of considered VMS.

Figure 11 shows that the proposed solution will produce less separated traffic during a sequential migration. The right column represents the average of 30 random sequences.

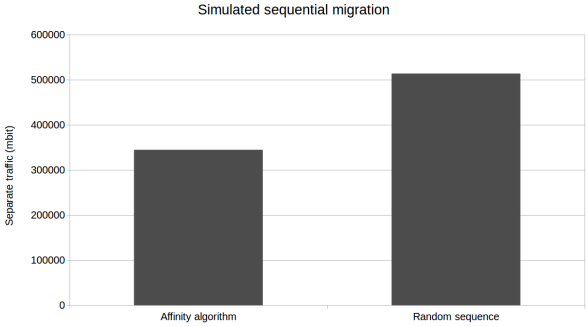

Figure 11: Less separated traffic when using the affinity algorithm

We have computed the volume of separated traffic for all possible sequences of individual VM migrations, namely 462 possible unique sequences. The affinity algorithm produces a schedule that yields near-optimal results. Only three sequences produce less separated traffic. It can therefore be said that, in 99% of the cases, the affinity algorithm yields better results

than random sequential scheduling.

Figure 12 shows the distribution of the volume of separated traffic generated by all the 462 possible sequences using a histogram representation.

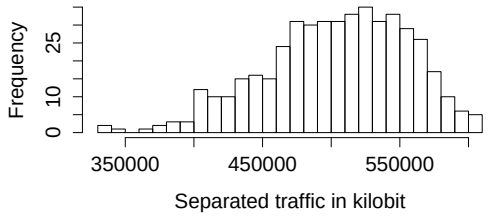

Figure 12: Histogram illustrating the volume of separated traffic for random sequences.

5. Conclusion

In this paper, we tackled multiple-virtual machine live migrations, using a combination of two algorithms, namely minimum cut graph partitioning and affinity-aware scheduling. Using the proposed solution, we were able to observe a significant reduction in the volume of separated traffic, as well as a reduction in the total migration time. Using both real and simulated virtual machine migrations, the memory- and network-related properties were examined and understood, and the main obstacle was defined as a "split components problem". The algorithms were tested on a KVM-based virtualization platform using Libvirt as the management tool.

At the heart of our contribution, we find the affinity-aware scheduling algorithm, which iteratively migrates Virtual Machines with strong connections to the target host and little affiliation to virtual machines at the source. Experiments show that, on average, more than 30% of this traffic can be eliminated by using it, as well as a reduction in the migration time. In a future study, we would like to test more realistic ways of generating the inter-Virtual Machines traffic by either resorting to real-life traffic traces extracted from data centers or by using known distributions available in the literature. Furthermore, a more sophisticated approach should be developed for deciding the size of a group in the partitioning phase that takes into account the aggregate dirty rate of the group in order to decide its size.

References

[1] H. Lu, C. Xu, C. Cheng, R. Kompella, and D. Xu, "vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications," in *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pp. 453–460, IEEE, 2015.

[2] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 267–274, IEEE, 2011.

[3] Y. Zhao and W. Huang, "Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud," in *2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 170–175, IEEE, 2009.

[4] K. Ye, X. Jiang, D. Ye, and D. Huang, "Two optimization mechanisms to improve the isolation property of server consolidation in virtualized multi-core server," in *2010 IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 281–288, IEEE, 2010.

[5] F. F. Moghaddam, M. Cheriet, and K. K. Nguyen, "Low carbon virtual private clouds," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 259–266, IEEE, 2011.

[6] K. Chanchio and P. Thaenkaew, "Time-bound, thread-based live migration of virtual machines," in *2014 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 364–373, IEEE, 2014.

[7] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[8] P. Lu, A. Barbalace, and B. Ravindran, "Hsg-lm: hybrid-copy speculative guest os live migration without hypervisor," in *Proceedings of the 6th International Systems and Storage Conference*, p. 2, ACM, 2013.

[9] A. Ashraf, B. Byholm, J. Lehtinen, and I. Porres, "Feedback control algorithms to deploy and scale multiple web applications per virtual machine," in *2012 EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 431–438, IEEE, 2012.

[10] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, "Comma: Coordinating the migration of multi-tier applications," *SIGPLAN Not.*, vol. 49, pp. 153–164, Mar. 2014.

[11] M. Bari, M. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," in *Networking Conference, 2014 IFIP*, pp. 1–9, June 2014.

[12] H. Liu and B. He, "Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 1192–1205, 2015.

[13] T. Lu, M. Stuart, K. Tang, and X. He, "Clique migration: Affinity grouping of virtual machines for inter-cloud live

migration," in *2014 IEEE International Conference on Networking, Architecture, and Storage (NAS)*, pp. 216–225, IEEE, 2014.

[14] C. Canali and R. Lancellotti, "A comparison of techniques to detect similarities in cloud virtual machines," *International Journal of Grid and Utility Computing*, vol. 7, no. 2, pp. 152–162, 2016.

[15] U. Deshpande, D. Chan, S. Chan, K. Gopalan, and N. Bila, "Scatter-gather live migration of virtual machines," *To appear in IEEE Transactions on Cloud Computing*, 2017.

[16] J. Díaz, J. Petit, and M. Serna, "A survey of graph layout problems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 313–356, 2002.

[17] B. J. Oommen and D. S. Croix, "Graph partitioning using learning automata," *IEEE Transactions on Computers*, vol. 45, no. 2, pp. 195–208, 1996.

[18] B. Oommen and D. Ma, "Deterministic Learning Automata Solutions to the Equipartitioning Problem," *IEEE Transaction Computer*, vol. 37, no. 1, pp. 2–13, 1988.

[19] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of 2010 IEEE INFOCOM,*, pp. 1–9, IEEE, 2010.

[20] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state vm management for data centers," in *NETWORKING 2012*, pp. 190–204, Springer, 2012.

[21] J. Chen, K. Chiew, D. Ye, L. Zhu, and W. Chen, "Aaga: Affinity-aware grouping for allocation of virtual machines," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 235–242, IEEE, 2013.

[22] T. K. Sarker and M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," in *Proceedings of the 2013 IEEE International Conference on Granular Computing (GrC)*, pp. 253–258, IEEE, 2013.

Vitae

Anis Yazidi received the M.Sc. and Ph.D. degrees from the University of Agder, Grimstad, Norway, in 2008 and 2012, respectively. He is currently an Associate Professor at the Department of Computer Science, Oslo Metropolitan University, Oslo, Norway. He is leading a research group on Autonomous Systems and Networks.

Frederik Ung obtained his M.Sc. in Computer Science from University of Oslo in 2015. He is currently working as operations coordinator at Phonect AS, Oslo, Norway.

Hårek Haugerud is an Associate Professor at the Department of Computer Science, Oslo Metropolitan University, Oslo, Norway. He received his Ph.D. degree in Theoretical Physics from the University of Oslo in 1994.

Kyrre Begnum is an Associate Professor at the Department of Computer Science, Oslo Metropolitan University, Oslo, Norway. He teaches system adminstration courses at the MSc and BSc levels. He holds a Ph.D. from the University of Oslo with a focus on understanding the behavior of large systems.