# Minimum-Impact First: Scheduling Virtual Machines Under Maintenance Scenarios

Anis Yazidi
anisy@oslomet.no
Oslo Metropolitan University (OsloMet)
Oslo, Norway

Hårek Haugerud
haugerud@oslomet.no
Oslo Metropolitan University (OsloMet)
Oslo, Norway

Fredrik Ung
fredrik.ung@gmail.com
Oslo Metropolitan University (OsloMet)
Oslo, Norway

Kyrre Begnum
kyrre@oslomet.no
Oslo Metropolitan University (OsloMet)
Oslo, Norway

## ABSTRACT

Virtual Machine (VM) migration is an important feature for ensuring smooth operations during maintenance and disaster recovery scenarios. The migration might be inter-site and in such a case the inter-site bandwidth which is typically Wide Area Network (WAN) might be a bottleneck. In such a case, the bandwidth is affected by the amount of inter-VM traffic that becomes separated during the migration process. The amount of separated traffic might not only cause degradation of the of the Quality of Service (QoS) of inter-communicating VMs but can also delay the migration process due to the congestion of the migration link. The state-of-the-art algorithm due to Yazidi et al. is an affinity aware algorithm that does not consider the completion time of the migration. The first stage of our algorithm is identical to Yazidi et al. where 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 a greedy algorithm for controlling the order of the migration groups by considering their inter-group traffic that greedily selects groups with the lowest impact in terms of volume of separated traffic which we denominate Minimum-Impact First (MIF). We also design a latency-aware algorithm that only schedules the quickest migration first. The latter simple heuristic interestingly outperforms legacy works in the case of migration over a non-dedicated link. We find that our MIF algorithm consistently outperforms the state-of-the-art algorithms by a clear margin using real-traffic traces by a margin larger than 40%. We show that the MIF algorithm ensures the lowest amount of separated traffic in both dedicated-link and non-dedicated-link scenarios.

## 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 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 [6, 13]. 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 [15]. 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 [13, 14]. 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 [9].

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 [3].

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) [10]. 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 the idea of minimizing the impact of the separated traffic. Our algorithm denoted as Minimum-Impact First (MIF) outperforms the state-of-the-art algorithms for the cases of dedicated migration link and non-dedicated migration link. Furthermore, we devise a latency-aware algorithm that is shown to outperform legacy algorithms for migration over a non-dedicated link.

## 2 RELATED WORK

In [1], 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 [16], 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 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, $\{VM_1, ..., VM_n\}$, is to be migrated, the traffic levels running between them are measured and stored in matrix whose elements are $VM[i,j]$ denoting traffic between $VM_i$ and $VM_j$. Let the migration completion time for $VM_i$ be $t_i$. Equation 1 gives the network impact that that should be minimized.

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

The VMbuddies system [5] 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. 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 [7], 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 *Time-bound thread-based Live Migration* (TLM) technique was proposed in [3]. 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 [6], 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 [4], 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.

## 3 OUR SOLUTION: IMPACT-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, we formally define the time needed for the migration in Section 3.1. In Section 3.2, we present our scheduling algorithm.

## 3.1 Migration time

In a two separate studies, Bari et al. [1] and Mann et al. [8] 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:

$$T_i = W/L \tag{2}$$

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

$$T_{p+s} = \frac{M \cdot \frac{1-(R/L)^N}{1-(R/L)}}{L} \tag{3}$$

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:

$$T_s = M/L \cdot (R/L)^N \tag{4}$$

The $N$ in the Equations 3 and 4 is given by:

$$N = \min(\lceil \log_{R/L} \frac{T \cdot L}{M} \rceil, \lceil \log_{R/L} \frac{X \cdot R}{M \cdot (L - R)} \rceil) \tag{5}$$

We provide the following formulas [5] 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:

$$v_i = \frac{M \cdot d^i}{r^i} \tag{6}$$

$$t_i = \frac{M \cdot d^i}{r^{i+1}} \tag{7}$$

Then the total network traffic during migration is:

$$V = \sum_{i=0}^{N} v_i = M \cdot \sum_{i=0}^{n} \frac{d_i}{r_i} \tag{8}$$

Table 1 denotes the variables used in Equation 8.

## 3.2 Subgroup scheduling based on minimum Impact

In [11], Oommen and Croix proposed a learning algorithm based on Object Migrating Automata (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. 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.

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.

$$T(G_i \rightarrow G_j) = \sum_{VM_i \in G_i, VM_j \in G_j} VM_{ij} \tag{9}$$

**Algorithm 1: MIF algorithm**

```
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
        $T(G_i \rightarrow D) = \sum_{G_k \in D} T(G_i \rightarrow G_k)$
        $T(G_i \rightarrow S) = \sum_{G_k \in S} T(G_i \rightarrow G_k)$
        $\Delta_i = T(G_i \rightarrow S) - T(G_i \rightarrow D)$
        $\delta T_i$=Migration Time of $G_i$
        $V_i = \Delta_i \delta T_i$
    end
    $i^* = \arg\min_i V_i$
    Migrate $G_{i^*}$
    $D = D \cup G^*$
    $S = S \setminus G_{i^*}$
end
```

Algorithm 1 migrates groups of VMs based on "the volume of separated traffic". The amount separated traffic resulting from migrating a group $G_i$ from source to destination is given by $\Delta_i$. The volume is the product of the traffic and completion time of the migration of $G_i$ given by $V_i = \Delta_i \delta T_i$. Please note that the notion of impact is also defined in Equation 1.

The amount of separated traffic in total given a source $S$ and destination $D$ is given by

$$\Delta = \sum_{G_j \in S, G_k \in D} T(G_j \rightarrow G_k)$$

Our algorithm 1 considers the change to the latter quantity as a result of the separated traffic resulting from migrating $G_i$ to the destination which is given by $\Delta_i = T(G_i \rightarrow S) - T(G_i \rightarrow D)$.

The amount of increase in the separated traffic achieved by moving a group $G_i$ from a source machine to a destination is the difference between the traffic of the group to the source server ($T(G_i \rightarrow S)$) due to separating $G_i$ from the VMs at the source $S$, on the one hand, and the respective traffic to the destination server ( $T(G_i \rightarrow D)$), on the other. In other words, it is the difference between the the traffic that goes through the network due to other VMs located at the source site S and communicating with the migrated group, which is expressed as $T(G_i \rightarrow S)$ and the co-located traffic resulting from migrating $G_i$ to the destination machine that goes through the memory. The reason why we deduct $T(G_i \rightarrow D)$ is that the traffic between co-located VMs in the same physical machine goes through memory by default, and thus has no cost.

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 yielding the minimum volume of separated traffic is migrated. The group is then removed from the list $S$ and appended to list $D$.

It is worth mentioning that the latter "impact", 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 lowest volume separated traffic to the destination will intuitively reduce the volume of separated traffic in a greedy manner.

## 4 EXPERIENCES AND RESULTS

In order to test the proposed algorithms on various *kinds* of data sets and to be able to retrieve reliable results, we performed two sets of experiments. The data is available at
http://pages.cs.wisc.edu/t̃benson/IMC10_Data.html. The traces from three of the data centers given in the above link are studied in [2]. We randomly selected the set of 500 VMs from the collected traffic traces, with the expectation that this set of 500 VMs will contain several VMs which have rather high mutual traffic while most of the VMs communicate with each other at a significantly lower rate. Based on the traces, we compute the traffic matrix which describes the communication rate.

We suppose the available bandwidth is 50 $Gb/s$ and that the memory size of each VM is set to $1Gb$. We run experiments on dedicated link and non-dedicated link and compare our algorithm denoted MIF to the following algorithms:

- Tao et al. algorithm found in [7].
- Yazidi et al. algorithm presented in our previous work in [12].
- Latency-aware algorithm, which migrates the group that has the Shortest Migration Time First. We denote this algorithm as SMTF.

### 4.1 Non-Dedicated Link

In this first experiment, we suppose that the migrations take place over a non-dedicated link. A typical example can be migrations that need to be performed between two data centers linked through WLAN as aforementioned in the introduction. In such a scenario, the migration traffic on one hand and the separated traffic resulting from inter-site communication between VMs still at the source and VMs migrated to the destination on the other hand will co-exist over the same link and thus both procedures will compete for the available bandwidth. Intuitively, the more congested the migration link as a result of the separated traffic, the higher the migration time of VMs. We shall investigate the effect of varying the group size of the migrated VMs and the effect of varying the dirty rate on the performance of the different algorithms.

In this scenario, we choose the dirty rate for each of the 500 VMs from a uniform distribution in the range of 500 $mb/s$. Therefore, the excepted dirty rate per VM is 250 $mb/s$. We apply first the LA algorithm for partitioning the VMs into groups. We run an experiment with size 10 VMs per group, which means 50 groups in total to schedule, and an experiment with size 20 VMs per group which means 25 groups to schedule.

In Figure 1, we compare the amount of separated traffic for the four algorithms. We remark that our proposed algorithm MIF achieves the lowest value of separated traffic. We also remark as we increase the group size, the amount of separated traffic is reduced as chatty VM are migrated as a group. Interestingly, the latency-aware algorithm, which is a simple algorithm outperforms two sophisticated legacy algorithms Tao et al. [7] and Yazidi et al. [12].

It seems that the group with the shortest migration time is strongly correlated to the least amount of separated traffic. In fact, a shorter migration time is a sign that the amount of separated traffic is low as the residual bandwidth for the migration is the difference between the link bandwidth and the amount of separated traffic.

For a group size of 10, and taking Yazidi et al. as baseline, we report in Table 2 the improvement in percentage of the different algorithms. we observe that the MIF algorithms is the most performant algorithm with an improvement of 60.1% while the latency aware SMTF algorithm has an improvement of 49.3%. The SMTF algorithm also outperforms the Tao et al. algorithm which is remarkable.

| | | Yazidi et al. | Tao et al. | SMTF | MIF |
|---|---|---|---|---|---|
| Improvement | Group of 10 | – | -5.3 % | 49.3 % | 60.1 % |
| | Group of 20 | – | -27.2 % | 41.6 % | 45.3% |

**Table 2: Improvement in percentage (positive or negative) compared to Yazidi et al. as baseline corresponding to the experiment in Figure 1.**

Now, we reduce the expected dirty rate of each VM to 150 $mb/s$ and the results are very similar and are shown in Figure 2. The

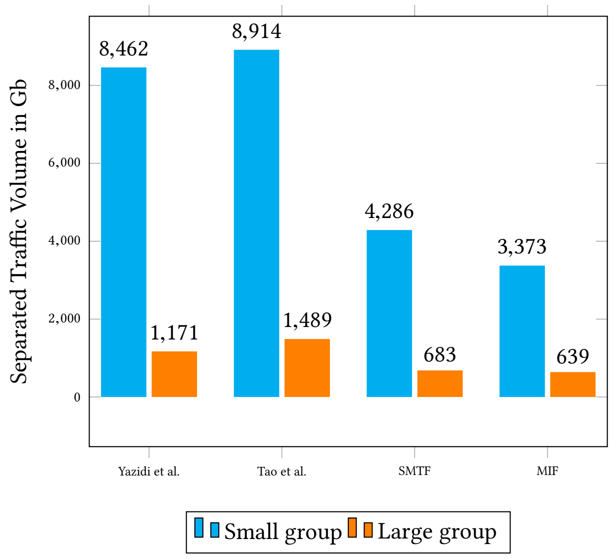

**Figure 1: Non-dedicated link: Average Dirty Rate = 250 $mb/s$**

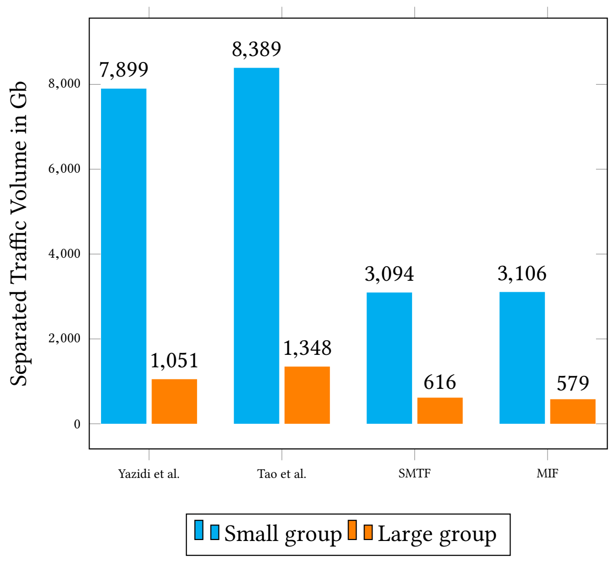

**Figure 2: Non-dedicated link: Average Dirty Rate = 150 $mb/s$**

main difference is that the Volume of Separated traffic is slightly decreased. Additionally the SMTF algorithm performs just as well as the MIF algorithm for small groups. In all other experiments the MIF algorithm performs best.

| | | Yazidi et al. | Tao et al. | SMTF | MIF |
|---|---|---|---|---|---|
| Improvement | Group of 10 | — | -6.1 % | 60.8 % | 60.7% |
| | Group of 20 | — | -28.2 % | 41.3 % | 44.8% |

**Table 3: Improvement in percentage (positive or negative) compared to Yazidi et al. as baseline corresponding to the experiment in Figure 2.**

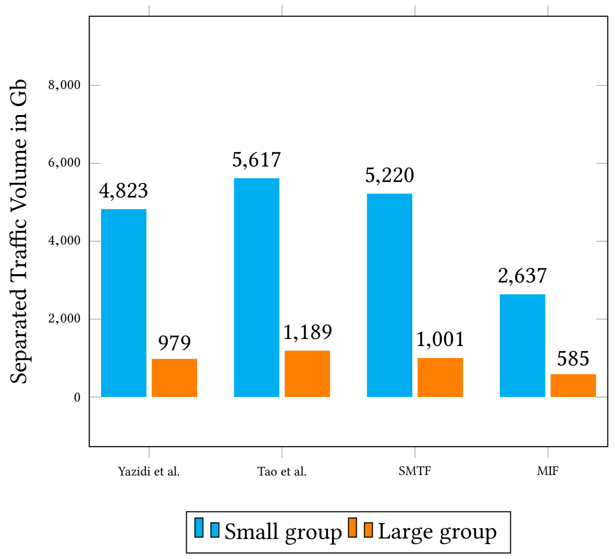

**Figure 3: Dedicated link: Average Dirty Rate = 250 $mb/s$**

## 4.2 Dedicated Link

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 excepted dirty rate per VM is 250 $mb/s$. The latency-aware SMTF algorithm does not yield as good results as in the previous case of non-dedicated link. In fact, in this case, the migration time over the dedicated link is not affected by the amount of separated traffic.

From the plots in Figure 3 If we take Yazidi et al. as baseline, the SMTF algorithm performs slightly worse with a change in the performance by $-8.2\%$, while the Tao et al. algorithm gives $-16.4\%$ worse performance. We observe that our MIF algorithm gives an improvement over Yazidi et al. [12] by as much as 45%.

| | | Yazidi et al. | Tao et al. | SMTF | MIF |
|---|---|---|---|---|---|
| Improvement | Group of 10 | — | -16.4 % | -8.2 % | 45.3 % |
| | Group of 20 | — | -21.3 % | -2.1 % | 40.2 % |

**Table 4: Improvement in percentage (positive or negative) compared to Yazidi et al. as baseline corresponding to the experiment in Figure 3.**

Now, we reduce the expected dirty rate of each VM to 150 $mb/s$. The results are similar. The only difference is that the Volume of Separated traffic is less. Again, the performance of the MIF algorithm is superior.

## 5 CONCLUSION

In this paper, we tackled multiple-virtual machine live migrations, using a combination of two algorithms, namely minimum cut graph partitioning in order to identify chatty VMs that should be migrated as a group and minimum impact-aware scheduling in order to schedule the migrations in a manner that minimizes the volume

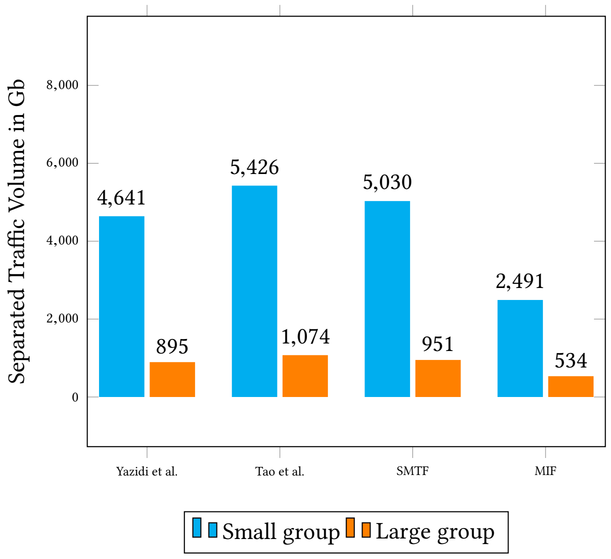

**Figure 4: Dedicated link: Average Dirty Rate = 150** $mb/s$

| | | Yazidi et al. | Tao et al. | SMTF | MIF |
|---|---|---|---|---|---|
| Improvement | Group of 10 | — | -16.9 % | -8.3 % | 46.3 % |
| | Group of 20 | — | -20 % | -6.3 % | 40.2 % |

**Table 5: Improvement in percentage (positive or negative) compared to Yazidi et al. as baseline corresponding to the experiment in Figure 4.**

of separated traffic. Using the proposed solutions, we were able to observe a significant reduction in the volume of separated traffic compared to the state-of-the-art algorithms.

## REFERENCES

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

[2] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 267–280.

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

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

[5] Liu, H., and He, B. Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments. *IEEE Transactions on Parallel and Distributed Systems 26*, 4 (2015), 1192–1205.

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

[7] Lu, T., Stuart, M., Tang, K., and He, X. Clique migration: Affinity grouping of virtual machines for inter-cloud live migration. In *2014 IEEE International Conference on Networking, Architecture, and Storage (NAS)* (2014), IEEE, pp. 216–225.

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

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

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

[11] Oommen, B. J., and Croix, D. S. Graph partitioning using learning automata. *IEEE Transactions on Computers 45*, 2 (1996), 195–208.

[12] Yazidi, A., Ung, F., Haugerud, H., and Begnum, K. Effective live migration of virtual machines using partitioning and affinity aware-scheduling. *Computers & Electrical Engineering 69* (2018), 240–255.

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

[14] Ye, K., Jiang, X., Ye, D., and Huang, D. 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)* (2010), IEEE, pp. 281–288.

[15] Zhao, Y., and Huang, W. 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* (2009), IEEE, pp. 170–175.

[16] Zheng, J., Ng, T. S. E., Sripanidkulchai, K., and Liu, Z. Comma: Coordinating the migration of multi-tier applications. *SIGPLAN Not. 49*, 7 (Mar. 2014), 153–164.