Affinity Aware-Scheduling of Live Migration of Virtual Machines Under Maintenance Scenarios

Anis Yazidi, Frederik Ung, Hårek Haugerud and Kyrre Begnum
Oslo Metropolitan University (OsloMet),
Department of Computer Science,
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 real-life experimental results and simulations show that our approach in some cases is able to decrease the volume of separated traffic by a factor larger than 60%.**

***Index Terms*—Live Migration, Graph Partitioning, Migration Scheduling, Separated Traffic.**

I. Introduction

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 KVM, VMware and Hyper-V support live migration. Server consolidation, load balancing, system maintenance and green computing are among the most popular use-case scenarios of live migration [1], [2], [3], [4], [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.

II. Related Work

In [8], 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%. Another related system, COMMA [9], 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 VMbuddies system [10] 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. 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. A system called Clique Migration [11], also migrates VMs based on their level of interaction. It is specialized for inter-site migrations. 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. 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. In [12], 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.

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

In [13], 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 (Physical Machine). 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.

$$T(G_i \rightarrow G_j) = \sum_{VM_i \in G_i, VM_j \in G_j} VM_{ij} \qquad (1)$$

Algorithm 1 migrates groups of VMs based on "affinity", which is a term that has been used to some degree by other researchers [14], [9]. High affinity between two VMs reflects the fact that the pair of VMs in question communicate intensively over the network.

Algorithm 1: Affinity 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**
    $T(G_i \rightarrow D) = \sum_{G_k \in D} T(G_i \rightarrow G_k)$
    $T(G_i \rightarrow S \setminus G_i) = \sum_{G_k \in S \setminus G_i} T(G_i \rightarrow G_k)$
    $\Delta_i = T(G_i \rightarrow D) - T(G_i \rightarrow S \setminus G_i)$
  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

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. 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 ($T(G_i \rightarrow D)$), 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 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.

IV. Experiences and Results

A physical lab was set up consisting of two small servers with Intel Core 2 processors, 8 GB RAM, 500 GB HDDs and Gigabit Ethernet NICs running Ubuntu 14.04, libvirt and KVM/QEMU version 2.0. Figure 1 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.

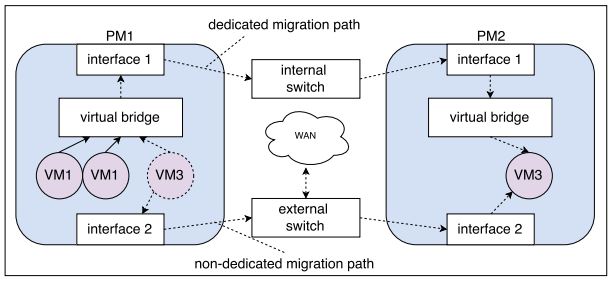

Figure 1: Physical lab

A small program is responsible for continuously generating the traffic among the hosts.

We use 0.216 mbit per second as 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.

a) Non-dedicated link scenario: In these experiments there are 16 communicating TinyCore VMs running simultaniously. With this amount of running instances, the CPU consumption averaged around 80%, after traffic initiation.

Figure 2 shows the avarage of traffic, based on migration data from 11 tests, when using random subgroups and calculated subgroups.

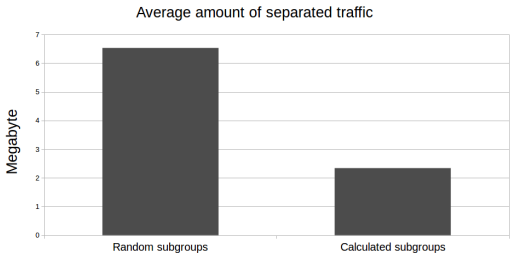

Figure 2: Network impact of migration random groups

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 of inter-site communication during migration, while our solution yields 2.34 MB. The proposed solution is 64% more cost-effective in this scenario.

b) Dedicated link scenario: The following experiment was 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 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 3.

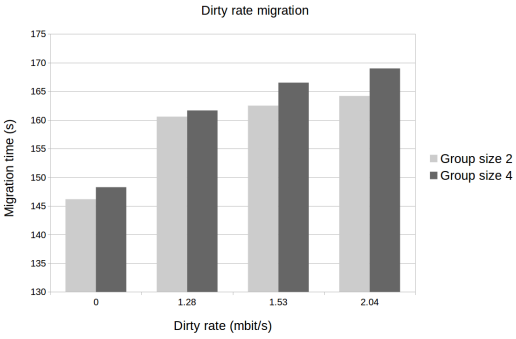

Figure 3: Effect of increasing the dirty rate on migration time

These VMs also sent traffic between each other, in addition to the dirty rate. The result of the experiments is that a subgroup size of 2 leads to roughly 50% more separated traffic compared to a subgroup size of 4, for all the tested dirty rates.

A. 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.

Figure 4 shows the results when we fixed the dirty rate at 5 mbit/s on each VM. The proposed solution produces the best result with all the different subgroup sizes. The results are similar for zero dirty rate.

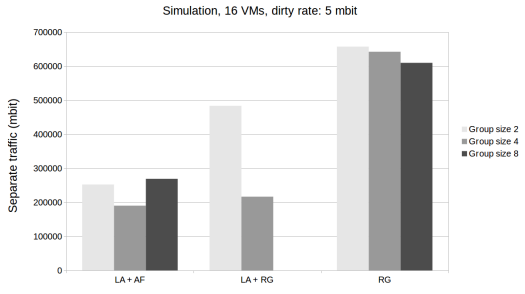

Figure 4: Volume of separated traffic with a dirty rate of 5 mbit/s. LA means OMA Learning Automata, AF Affinity based algorithm and RG random subgroups.

Figure 5 shows the impact of doubling the dirty rate to 10 mbit/s.

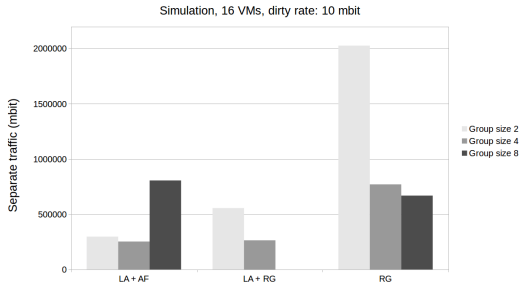

Figure 5: Volume of separated traffic with a dirty rate of 10 mbit/s

V. 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. 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.

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] 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.

[4] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.