EDITOR: Ewa Deelman, deelman@isi.edu

Interhost Orchestration Platform Architecture for Ultrascale Cloud Applications

Sasko Ristov and Thomas Fahringer, *University of Innsbruck, 6020 Innsbruck, Austria*
Radu Prodan, *University of Klagenfurt, 9020 Klagenfurt, Austria*
Magdalena Kostoska and Marjan Gusev, *Ss. Cyril and Methodius University, Skopje 1000, Macedonia*
Schahram Dustdar, *TU Wien, 1040 Vienna, Austria*

Cloud data centers exploit many memory page management techniques that reduce the total memory utilization and access time. Mainly these techniques are applied to a hypervisor in a single host (intra-hypervisor) without the possibility to exploit the knowledge obtained by a group of hosts (clusters). We introduce a novel interhypervisor orchestration platform to provide intelligent memory page management for horizontal scaling. It will use the performance behavior of faster virtual machines to activate prefetching mechanisms that reduce the number of page faults. The overall platform consists of five modules—profiler, collector, classifier, predictor, and prefetcher. We developed and deployed a prototype of the platform, which comprises the first three modules. The evaluation shows that data collection is feasible in real-time, which means that if our approach is used on top of the existing memory page management techniques, it can significantly lower the miss rate that initiates page faults.

Dynamic memory page management techniques, such as memory deduplication, page faults management, memory overcommitment, memory ballooning, or hot-swapping, rely on the cooperation of the virtual machines (VMs) hosted on a single physical host.[1] Although all these techniques provide autonomous and automatic memory page management that reduces the total memory utilization and memory access time, their application domain is still limited within a single host. On the other hand, cloud environments use horizontal scaling such that hundreds or thousands of VMs (*VM-siblings*) of the same image work on the same problem. These VM-siblings use the same guest operating system, the same code segment, and many memory pages of the same data segment are identical or similar, thereby conducting similar memory access patterns. Since VM-siblings may be scheduled on multiple hosts, the state-of-the-art intrahost memory management methods cannot be fully exploited. The goal of this article is to introduce an interhypervisor orchestration platform, which uses the knowledge obtained by VM-siblings that are hosted on different hosts and potentially use it in real-time to reduce the memory page fault ratio, for negligible resource utilization overhead and latency.

1089-7801 © 2020 IEEE
Digital Object Identifier 10.1109/MIC.2020.3034293
Date of publication 30 October 2020; date of current version 18 June 2021.

RELATED WORK

Wei-Zhe Zhang[2] presented an automatic memory control of multiple VMs that dynamically adjusts allocation according to the used memory by the VMs, while Qi Zhang[3] used a shared memory pool for fast just-in-time memory page recovery. Although these techniques reduce the number of memory page faults, they apply on a single host without being aware of other VM-siblings at other hosts.

Hines[4] and Tasoulas[5] use the estimation of application memory requirements for memory balancing and distribution. Their techniques automate the

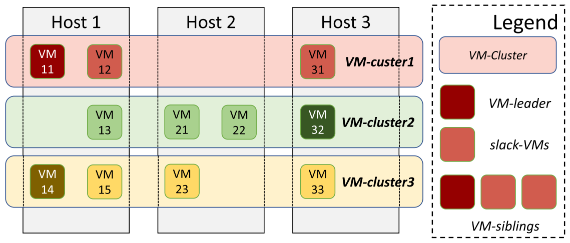

FIGURE 1. Definition of VM-cluster and VM-siblings (VM-leader and slack-VMs).

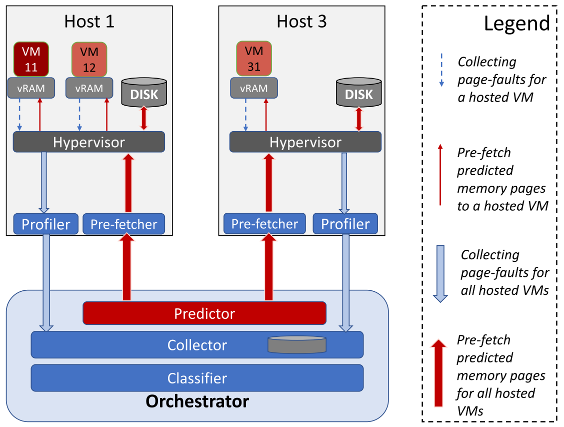
Host 1
Host 3
Legend
VM 11
VM 12
VM 31
vRAM
DISK
Hypervisor
Profiler
Pre-fetcher
Predictor
Collector
Classifier
Orchestrator
Collecting page-faults for a hosted VM
Pre-fetch predicted memory pages to a hosted VM
Collecting page-faults for all hosted VMs
Pre-fetch predicted memory pages for all hosted VMs

FIGURE 2. Design of the interhypervisor orchestration platform.

distribution of the memory across VMs, but again limit the hypervisor's level on a single host. Additionally, they quantify the amount of memory without a qualitative estimation of the specific page accesses in the near future for each VM-sibling.

Several researchers presented an orchestration for multiple hypervisors. Gopalan *et al.*[6] introduced the span virtualization, which allows multiple hypervisors to control the memory of guest's OS concurrently. Still, this orchestration is on a single host. Fecade *et al.*[7] orchestrated multiple hypervisors in mobile cloud computing. They used a Bayes-based classifier to predict failures in hypervisor and to prevent VM failures by migrating them to another host. However, the authors verified their approach with simulation, without real implementation and without considering the network latency.

Prefetching is a commonly used technique in memory management. Ren *et al.*[8] introduced an asynchronous prefetching mechanism to speculatively prefetch the dirty pages from a primary VM on a secondary VM on another host without disrupting its execution. While this algorithm shortens the sequential dependence when VM checkpoints are generated and transmitted to a VM on another host, still, it uses one-to-one mapping between the primary and secondary VMs without considering memory access patterns from other VM-siblings.

INTERHYPERVISOR ORCHESTRATION PLATFORM

Terminology

Before diving into details, we explain the used terminology. Let VM_{11}, VM_{12}, and VM_{31} denote three VMs hosted on two hosts (Host1 and Host3), as presented in Figure 1, such that the first index identifies the host, while the second one determines the VM on that host. For example, VM_{12} represents the second VM hosted on Host1. Let all VMs are a part of horizontal scaling, which means they run the same application, either for different input data or serve requests generated by different users.

Definition 1. (VM-cluster). *A* VM-cluster *is a set of VMs that are scaled horizontally and usually deployed across multiple hosts.*

For example, Figure 1 illustrates a part of a data center, where VMs are grouped in: VM-cluster1 with three VM-siblings, while VM-cluster2 and VM-cluster3 with four. Each VM-sibling runs the same application, which is scaled horizontally across three hosts.

Definition 2. (Specific VMs within a VM-cluster). *A* VM-leader *is the fastest VM-sibling that leads the execution within one VM-cluster.* Slack-VMs *are all other VM-siblings that perform slower than the VM-leader within a specific VM-cluster.*

Each VM-cluster identifies a single VM-leader at each point of time, while all the other VM-siblings are slack-VMs. However, these roles may change in time for a specific VM.

Without losing generality, we assume a heterogeneous environment where the VMs and hosts perform at different speeds. Figure 1 presents that VM_{11}, VM_{32}, and VM_{14} are VM leaders of the corresponding VM-clusters 1, 2, and 3. For example, besides the VM leader VM_{11}, the VM-cluster1 contains also other VM-siblings (*slack-VMs* including VM_{12} and VM_{31}).

Platform Architecture

The interhypervisor platform orchestrates hypervisors of a group of horizontally scaled VM-clusters, as presented in Figure 2. To orchestrate VM-siblings in the horizontal scaling, the first step of the orchestrator is to classify all VMs in VM-clusters according to the information from the cloud controller. Further on, the orchestrator communicates with the agents on each host to gather the necessary access data that caused page faults for each hosted VM-sibling. Finally, the orchestrator detects patterns in the page faults to determine the VM-leader of each VM-cluster, whose behavior will be used later for prediction and prefetching the memory pages of the *slack-VMs*.

The proposed interhypervisor orchestration platform consists of two main processes: *i*) a *bottom-up* data collection and management, and *ii*) a *top-down* control process. This paradigm is particularly suitable for concurrent management of these two processes. For example, hypervisors can manage VMs on the same host, while our orchestrator supports global views of VM-clusters, combining and analyzing data of all VM-siblings of a single VM-cluster scattered on different hosts. On the other side, the top-down control process allows flooding the information about prefetching memory pages that already generated page faults at the VM-leader, thereby reducing the page faults ratio for the slack-VMs and significantly improving their performance.

The orchestration platform consists of five modules: *classifier*, *collector,* and *predictor* within the central part of the orchestrator, together with *profiler* and *prefetcher* distributed on each host. Since the orchestrator collects data about page faults from profilers and sends the predicted memory pages for all hosted VMs back to prefetchers hosted on multiple hosts, the modules of the orchestrator should be deployed on servers with a higher bandwidth and a small network diameter to the orchestrated hosts. Our approach with the centralized orchestrator and distributed profilers does not require a direct communication between VMs' operating systems.

The orchestrator will have access to memory usage info for all orhestrated VMs, which may open security and privacy risks. In order to mitigate such risks, the profiler sends anonymized data that do not contain information about the owners of the VMs, their addresses, or credentials.

The orchestrator is only a logical representation and any of its three modules may be hosted on a separate server. Moreover, the modules may be containerized and managed with Kubernetes for higher scalability. The orchestrator implementation is not affected by the existing cloud infrastructure as it works on a lower (hypervisor) level.

Classifier

The classifier groups all VMs in VM-clusters, such that each VM member of horizontal scaling becomes a VM-sibling within a specific VM-cluster. Clustering is a dynamic process in the cloud ecosystem regularly performed by the classifier, where VMs are instantiated, replicated, migrated to another host and terminated. The classifier can be extended to cluster VMs that are not a part of horizontal scaling or VM-clusters without VM-leader, if they have a similar memory access pattern.[16]

Profiler

Each host deploys a profiler that communicates with the local hypervisor to collect data about page faults and swapping for each hosted VM, and sends it to the collector within the centralized orchestrator. The profiler can be built based on iBalloon,[9] which provides efficient intrahypervisor VM memory balancing within a consolidated host. iBalloon runs a lightweight monitoring process (daemon) in each VM of the host that gathers information about its memory utilization. This technique improves the overall performance for memory-intensive applications with less than 5% CPU overhead tradeoff, compensated due to the CPU under utilization compared to the main memory.

Collector

The collector is a simple module that gathers the data from all profilers in a single and persistent centralized *knowledge base*, which contains memory access data of all VM-siblings within each VM-cluster. The main challenge of the collector is to determine the size and length of historical data considered by the predictor. Accordingly, the collector splits the received data into hot and cold parts. The predictor uses the hot part to determine the memory pages to be prefetched at VM-siblings, while the cold part helps strategic planning of future data center design and maintenance through offline analysis. Extending the iBalloon, one can run a daemon that collects all data from each host's memory profiler. Although the concept of cold and hot memory pages[10] provides performance overhead, we can still use this concept to reduce the memory access interception.

Predictor

The predictor is the brain of the orchestrator that exploits the collected data of the knowledge-base. It possibly uses machine learning-based techniques (beyond the scope for this article) to predict the memory page accesses for each VM-sibling within a VM-cluster. Since the cloud environment is a heterogeneous one, both the VM type and the underlying host computation resources (CPU, RAM) must be considered by the predictor to further improve the prediction accuracy. The centralized predictor can exploit data for memory access patterns of all VM-siblings within a

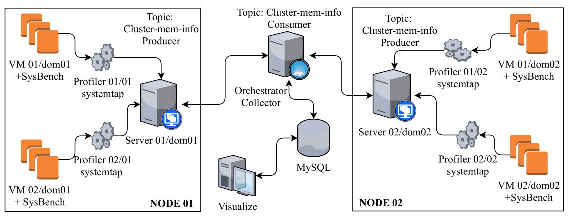

FIGURE 3. Platform prototype implementation

VM-cluster, regardless whether they are hosted on a single host or scattered across several hosts. With this knowledge, the predictor can estimate more accurately which memory pages will be accessed by slack-VMs and inform the prefetchers and hypervisors accordingly to prefetch those memory pages into guest main memory.

Prefetcher

The predictor submits the information to the prefetcher on each host to initiate prefetching of memory pages from local disks to all locally hosted VMs. The prefetcher issues a command to the *Hypervisor* to prefetch (swap-in) a memory page into each VM to reduce the memory page fault.

PLATFORM PROTOTYPE IMPLEMENTATION AND EVALUATION

We implemented three modules (the classifier, profiler, and prefetcher) to investigate the effectiveness of the orchestration platform for data collection. The goal of the initial platform prototype was to evaluate how many memory page faults can be generated, profiled, and transferred to the collector, both for traditional and virtual environment.

Platform Prototype Implementation

Figure 3 presents the implementation of our platform prototype. We deploy the profiler on two hosts, each installed with XenServer (v7.6.0) and Ubuntu 16.04.6 amd64. The hosts (quad-core CPU, 4 GB RAM, SSD) are connected using NAT and 1 Gb/s network. Each host administers a VM-pool Host 0*x*/*dom*0*x* and each VM gets a VMID, which is used to specify it as a source of transferred data in the profiler. The collector and the classifier are deployed on the orchestrator, which has 6 GiB RAM and is also installed with the same Ubuntu.

We used SysBench benchmark tool to generate intensive memory allocation and page faults in Ubuntu with a single thread, which generated a total of 100GiB memory blocked in blocks of 1MB each:

```
sysbench --num-threads=1 --test=memory
    --memory-block-size = 1M --memory-total-
    size=100G run
```

Generated page faults were profiled by systemtap, which was extended to submit the VMID. The following listing presents an entry from the page fault, and shows when (including microseconds) and on which host a process generated a page fault, which could be either write (w) or read (r). Additionally, we submit the type of the page fault, i.e., minor or major. The size of each entry was always the same (51B).

```
ID:Timestamp:PID:fault_address:fault_access:kind:fault_time
1:1591116972164574:30134:140013325101104d:w:minor:1
```

For sending the profiled data, we used Apache Kafka (v2.2.0) and Zookeeper configured with default ports and JDK (1.8.0) on all brokers in the system. Systemtap sends data to the Kafka producer at Host 0*x*/*dom*0*x*, which is collected by the Kafka consumer at the collector. Finally, the collected stream data was stored in a file and the collector (written in C) writes it in MySQL. The classifier specifies VMIDs and groups them in a VM-cluster. Our current classifier prototype uses only one VM-cluster as we measure the data transfer rate.

Evaluation

We conducted two groups of experiments. The first group of experiments was intended to investigate the cap varying the number of sources (VMs and hosts), i.e., how many entries the platform prototype is able to generate and send them to the collector without using the profiler systemtap. The second group of experiments determined the overhead of introducing the profiler, which resulted in lower number of records that were collected in real-time. We run each experiment in two different environments (bare metal and virtual). We denote experiments as B1, B1P, B2, B2P, V1, V2, and V1P, where B and V denote the environment (bare metal or virtual), 1 or 2 sources (VMs or hosts), and P denotes experiments with the profiler.

Table 1 presents the achieved throughput of each experiment. We observe that sending data without the profiler achieved 17 million entries/s for one node as a source, which is the maximum from all experiments since we used only one node. Introducing another node (B2) reduced the throughput to 12 million entries/s. On the other side, the profiler (B1P) reduces the throughput to 250000 entries/s with one node and is stable even with two nodes because the streaming cap of 17 million entries/s is not reached.

Virtual environment reduced the throughput by half for the experiment with one VM compared to the

TABLE 1. Results of the evaluation. Presented number for the throughput shows the million of entries received by the collector per second.

Experiment	Average throughput ($\cdot 10^6$/s)
B1	17
B1P	0.25
B2	12
B2P	0.25
V1	8.5
V1P	0.078
V2	0.25

equivalent B1 and 3.2 times with the profiler. An interesting observation is the higher deviance in the virtual environment.

Discussion

Since the memory access time at the host is around 50 ns,[11] we estimate around 20 million memory accesses/s if all accesses are page hits, without any page misses and swaps. However, we are mostly interested in TLB misses and page faults, which happen in a range between $0.1-1\%$.[11] This leads to a maximum number of 200000 records/s that need to be stored and processed. We assume the worst case where all TLB misses are also page table misses (page faults), which is opposite to the write count disparity feature. On the other side, the total number of memory pages is usually smaller than 200000, as the legacy page size of 4 kB is nowadays abandoned to reduce the TLB misses. More precisely, all modern CPU architectures and operating systems support memory page size of 2 MB, while some even in the range of GiB. In a virtual environment, extended page table (EPT) faults are handled within 2.4 μs.

We selected the current state-of-the-art streaming platforms to evaluate the feasibility of the new proposed platform, such as Apache Kafka, which can handle at each profiler up to 800000 of 100B-long messages per second, regardless of the data size (even up to 1.4 TB). Our platform achieved the maximal throughput of 17 million entries/s, 51B each.

Although the profiler can collect and update the page faults at each host, Zhang *et al.*[12] specify a hybrid hardware and software tracing mechanism to collect and profile last-level TLB misses, up to cache line granularity of 64B. Moreover, another challenge is to collect data from all profilers to the central orchestrator. For example, for a network overhead of only 1% in 1s^{-1}, we can submit 1300 records, 100B each. Although profilers can group several messages into a few larger ones to reduce the packet header overhead, the total bandwidth remains in a similar range.

Let us analyze the price to be paid to achieve increased performance. At each host, the platform runs both Kafka to utilize a portion of computing resources. While the CPU is usually underutilized in data centers, the memory is often a bottleneck. Additionally, there is a small network overhead depending on the number of pages transmitted. This bottleneck is visible when more nodes or VMs are used (see Table 1).

FURTHER IMPLEMENTATION CHALLENGES

Network Overheads

The designers and programmers must consider the network latency and bandwidth that also impact the quality of the gathered data. Another challenge is the network heterogeneity, especially its latency, as hosts can be connected through a single physical switch, while others through several with higher latency. Recent high-speed and high-throughput memory and network, such as byte-addressable non-volatile memory express (NVMe) over fabrics (NVMf) reported negligible application performance degradation.[13] For example, the latest networking generates very low latency of only 1 μs and a very high bandwidth up to 200s^{-1}.[14] These trends in the networking allow possibilities for broader dynamic memory page management through the network and make our orchestration platform feasible.

Prediction Implementation Challenges

The amount of data analyzed by the prediction process impacts its performance. For example, a large history of records has less impact over the current memory accesses due to many context switches that may happen in the meantime. On the other side, considering a small amount of historical records may not be enough as a slack-VM may perform much worse than the VM-leader and, thus, a memory access pattern cannot be detected or valid for this particular case.

The predictor must propose the pages to swap to the disk and avoid being prefetched to the other VM-siblings. Various application types can also show different behavior.

The behavior of each VM fluctuates due to cloud performance instability[15] and therefore, there is no simple and appropriate function for modelling variations in memory page accesses of a VM. Nevertheless, we exploit the fact that caches and memory paging are not directly mapped but associative, which means that even a relaxed prediction performance still diminishes the memory page fault rate.

The prediction accuracy is affected by other VMs of other VM-clusters running other jobs on the same host. For example, VM_{13}, VM_{14}, and VM_{15}, which share the same Host 1 memory with VM_{11} and VM_{12}, will affect their memory access and page faults (see Fig 1). This may make the prediction of VM_{31} page access less accurate based on VM_{11}'s access pattern. This problem is analyzed by Nemati *et al.*.[16] They introduced inter- and intracluster similarity metrics to discover distinct groups of workloads with negligible CPU and memory overhead.

The interhypervisor orchestrator needs to predict a vector of memory page accesses for each VM-sibling, as follows.

- Characterize the VM-cluster using a set of parameters that reflect the memory pages accesses for each VM-sibling.[16]
- Estimate the memory access of each VM-leader and use it for VM-siblings. Additionally, considering the heuristics with the write count disparity, only a few and frequently updated memory pages could reduce page faults even more.[17]
- Experimentally evaluate the various machine learning methods (including random forest or similar) considering the tradeoff among latency and resource utilization overhead versus performance.

Prefetching Challenges

Modern operating systems and hypervisors support memory pages of various size, such as small (4 KiB), medium (2 MiB), and even large of up to 1GiB. While such plethora of heterogeneous memory page sizes may improve the performance,[18] it may convolute both prediction and prefetching. For example, memory access pattern of a VM-leader that uses memory pages of medium size differs from the access pattern of slack-VMs that use small sized ones.

On the other side, write memory accesses on VM-leaders may be logged by the hardware using the Page-Modification Logging[19] on commodity Intel processors. This hardware-assisted enhancement allows the hypervisor to monitor the modified (dirty) memory pages directly while running VMs, thereby distinguishing the "write-hot" and "write-cold" memory pages in real-time.

CONCLUSION AND FUTURE WORK

We have introduced a platform that enhances the memory page management techniques to reduce the page faults and increase the performance of virtualized data centers based on an interhypervisor (interhost) approach. State-of-the-art techniques implemented in today's virtualized environments include host-based prefetching and memory swapping concepts. Currently, these approaches are implemented for a single host and cannot be exploited for VMs spread over different hosts.

The interhost orchestration platform has a potential to open up new research directions in cloud data center memory management. Our approach enhances the memory page management implemented for an intrahypervisor solution by an interhypervisor platform, as a more efficient dynamic technique intended for cloud data centers. It can be efficiently implemented for VMs running an application that implements large horizontal scaling among different hosts.

The basic principle of the new proposed approach for the interhypervisor orchestration platform is to detect memory access patterns that generate page faults within VMs hosted on different hosts. Exploiting the patterns in gathered data, the orchestrator may predict which memory pages will be accessed in the near future and, therefore, may avoid generating page faults at the other VMs (slack-VMs).

The platform prototype was developed including the three modules classifier, profiler, and collector. The initial evaluation showed the effectiveness of the platform to collect generated entries about page-faults with a maximal throughput of 17 million entries/s. Although the initial prototype of the profiler reduced the throughput to 250000 entries/s of a host, still the platform prototype was able to reach the estimated 200000 page faults/s,[11] even with a low power hosts and 1 s^{-1} network.

We are currently working in the PRE-FETCH project on implementation of the other two modules the predictor and prefetcher and will investigate the effectiveness of the overall interhost orchestration platform. The initial experiments with the random forest predictor showed a promising accuracy.

ACKNOWLEDGMENTS

This work was supported in part by bilateral AUT-MKD project PRE-FETCH: Pro-active Memory Management with Page Faults Prediction in Clouds, financed by Austrian Federal ministry of Science, Research and Economy (BMWFW), and the Ministry of Education and Science of Republic of North Macedonia, and in part by the ASPIDE project funded European Union's Horizon 2020 research and innovation program under Grant Agreement 801091.

REFERENCES

1. H. Liu *et al.*, "Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1350–1363, May 2015.
2. W. Z. Zhang, H. C. Xie, and C. H. Hsu, "Automatic memory control of multiple virtual machines on a consolidated server," *IEEE Trans. Cloud Comput,*, vol. 5, no. 1, pp. 2–14, Jan.–Mar. 2017.
3. Q. Zhang *et al.*, "MemFlex: A shared memory swapper for high performance VM execution," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1645–1652, Sep. 2017.
4. M. Hines *et al.*, "Applications know best: Performance-driven memory overcommit with Ginkgo," in *Proc. IEEE CLOUDCOM*, 2011, pp. 130–137.
5. E. Tasoulas, H. Haugerund, and K. Begnum, "Bayllocator: A proactive system to predict server utilization and dynamically allocate memory resources using Bayesian networks and ballooning," in *Proc. 26th Large Installation Syst. Admin. Conf.*, 2012, pp. 111–122.
6. K. Gopalan, R. Kugve, H. Bagdi, Y. Hu, D. Williams, and N. Bila, "Multi-hypervisorvirtual machines: Enabling an ecosystem ofhypervisor-level services," in *Proc. Conf. Usenix Annu. Tech. Conf.*, 2017, pp. 235–249.
7. B. Fekade, T. Maksymyuk, and M. Jo, "Clustering hypervisors tominimize failures inmobile cloud computing," *Wireless Commun. Mobile Comput.*, vol. 16, no. 18, pp. 3455–3465, 2016.
8. S. Ren, Y. Zhang, L. Pan, and Z. Xiao, "Phantasy: Low-latency virtualization-based fault tolerance via asynchronous prefetching," *IEEE Trans. Comput.*, vol. 68, no. 2, pp. 225–238, Feb. 2019.
9. Q. Zhang *et al.*, "iBalloon: Efficient VM memory balancing as a service," in *Proc. IEEE Int. Conf. Web Serv.*, Jun. 2016, pp. 33–40.
10. W. Zhao, Z. Wang, and Y. Luo, "Dynamic memory balancing for virtual machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 37–47, Jul. 2009.
11. D. Patterson and J. Hennessy, *Computer Organization and Design RISC-V Edition*, 1st ed. San Mateo, CA, USA: Morgan Kaufmann, 2017.
12. J. Zhang, Y. Liu, H. Li, X. Zhu, and M. Chen, "PTAT: An efficient and precise tool for tracing and profiling detailed TLB misses," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 3, pp. 62:1–62:17, May 2018.
13. Z. Guz *et al.*, "NVMe-over-fabrics performance characterization and the path to low-overhead flash disaggregation," in *Proc. ACM Int. Syst. Storage Conf.*, 2017, pp. 16:1–16:9.
14. S. Gugnani, X. Lu, and D. K. D. Panda, "Swift-X: Accelerating OpenStack swift with RDMA for building an efficient HPC cloud," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2017, pp. 238–247.
15. R. Mathá, S. Ristov, T. Fahringer, and R. Prodan, "Simplified workflow simulation on clouds based on computation and communication noisiness," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1559–1574, Jul. 2020.
16. H. Nemati, S. V. Azhari, and M. R. Dagenais, "Host hypervisor trace mining for virtual machine workload characterization," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2019, pp. 102–112.
17. X. Chen *et al.*, "Refinery swap: An efficient swap mechanism for hybrid DRAM-NVM systems," *Future Gener. Comput. Syst.*, vol. 77, no. C, pp. 52–64, 2017.
18. T. Mason, T. D. Doudali, M. Seltzer, and A. Gavrilovska, "Unexpected performance of Intel Optane dc persistent memory," *IEEE Comp. Architecture Lett.*, vol. 19, no. 1, pp. 55–58, Jan.–Jun. 2020.
19. Intel Corporation, "Page modification logging for virtual machine monitor white paper," 2015, Accessed on: Sep. 20, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/processors/page-modification-lo gging-vmm-white-paper.html

SASKO RISTOV is currently a Postdoctoral University Assistant with the University of Innsbruck, Innsbruck, Austria. His research interests include performance modeling, optimization, scheduling, and resource management in distributed and parallel systems. He received the Ph.D. degree from Sts. Cyril and Methodius University, Skopje, North Macedonia, in 2012. He is the corresponding author of this article. Contact him at sashko@dps.uibk.ac.at.

THOMAS FAHRINGER is currently a Full Professor of computer science heading the Distributed and Parallel Systems Group, University of Innsbruck, Innsbruck, Austria. His research interests include software architectures, programming paradigms, compiler technology, performance analysis, and prediction for parallel and distributed systems. He received the Ph.D. degree in 1993 from the Vienna University of Technology, Vienna, Austria. Contact him at tf@dps.uibk.ac.at.

RADU PRODAN is currently a Full Professor of distributed systems with the Institute of Software Technology, University of Klagenfurt, Klagenfurt, Austria. His research interests include performance, optimization, and resource management tools for parallel and distributed applications. He received his the Ph.D. degree in 2004 from the Vienna University of Technology, Vienna, Austria. He is a member of IEEE. Contact him at radu@itec.aau.at.

MAGDALENA KOSTOSKA is currently an Associate Professor with the Sts. Cyril and Methodius University, Skopje, North Macedonia. She received the Ph.D. degree from the Ss. Cyril and Methodius University in 2014. Her research interests include cloud computing and Internet of Things. Contact her at magdalena.kostoska@finki.ukim.mk.

MARJAN GUSEV is currently a Full Professor with the University Sts. Cyril and Methodius, Skopje, North Macedonia. He received the Ph.D. degree from University of Ljubljana, Ljubljana, Slovenia, in 1992. His research interests include Internet of Things, cloud computing, and eHealth solutions. Contact him at marjan.gushev@finki.ukim.mk.

SCHAHRAM DUSTDAR is currently a Full Professor of Computer Science heading the Distributed Systems Group, TU Wien, Vienna, Austria. His work focuses on Internet technologies. He is an IEEE Fellow, a member of the Academia Europaea, and an ACM Distinguished Scientist. Contact him at dustdar@dsg.tuwien.ac.at.