Principles and Performance Characteristics of Algorithms for Live VM Migration

Petter Svärd
Dept. Computing Science
Umeå University
SE 90197 Umeå
Sweden
petters@cs.umu.se

Benoit Hudzia
SAP Research CEC Belfast
SAP (UK) Limited
BT37 0QB Newtownabbey
benoit.hudzia@sap.com

Steve Walsh
SAP Research CEC Belfast
SAP (UK) Limited
BT37 0QB Newtownabbey
steve.walsh@sap.com

Johan Tordsson
Dept. Computing Science
Umeå University
SE 90197 Umeå
Sweden
tordsson@cs.umu.se

Erik Elmroth
Dept. Computing Science
Umeå University
SE 90197 Umeå
Sweden
elmroth@cs.umu.se

ABSTRACT

Since first demonstrated by Clark et al. in 2005, live migration of virtual machines has both become a standard feature of hypervisors and created an active field of research. However, the rich ongoing research in live migration focus mainly on performance improvements to well-known techniques, most of them being variations of the Clark approach. In order to advance live migration beyond incremental performance improvements, it is important to gain a deeper understanding of the live migration problem itself and its underlying principles.

To address this issue, this contribution takes a step back and investigates the essential characteristics of live migration. The paper identifies five fundamental properties of live migration and uses these to investigate, categorize, and compare three approaches to live migration: precopy, postcopy and hybrid. The evaluated algorithms include well-known techniques derived from that of Clark as well as novel RDMA in-kernel approaches. Our analysis of the fundamental properties of the algorithms is validated by a set of experiments. In these, we migrate virtual machines with large memory sizes hosting workloads with high page dirtying rates to expose differences and limitations of the different approaches. Finally, we provide guidelines for which approach to use in different scenarios.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management; D.4.8 [**Operating Systems**]: Performance

General Terms

Design, Measurement, Performance

Keywords

Platform virtualization, Virtual machine monitors, Performance evaluation

1. INTRODUCTION

Although live Virtual Machine, *VM*, migration functionality is present in all mainstream hypervisors like Xen, KVM, Microsoft Hyper-V and VMware, the implementations are usually limited to the precopy approach, where the VM's runtime state is copied from source to destination host before the execution is resumed at the destination. While precopy live migration is a proven technology, it cannot in a transparent manner migrate demanding workloads, i.e. workloads that are CPU and/or memory intensive, because of issues with high downtime and unpredictable behavior [Liu et al. 2008, Svärd et al. 2011, Mohan and S 2013]. Many attempts to improve the performance of the precopy approach have been made [Svärd et al. 2011, Wood et al. 2011, Pan et al. 2012, Hudiza and Shribman 2012, Hu et al. 2012, Zhang et al. 2013, Song et al. 2013] but they all retain the same basic algorithm, derived from Clark [Clark et al. 2005].

The main alternative to the precopy algorithm is postcopy live migration. Postcopy algorithms move the execution to the destination host at the beginning of the migration process and then transfers the runtime state in an on-demand manner. Such approaches promise to solve the predictability and downtime issues with precopy migration [Hines and Gopalan 2009, Hirofuchi et al. 2011] but so far, postcopy migration has only been demonstrated for non-demanding workloads. The severe performance penalty for retrieving memory pages over the network means that there is a considerable risk of performance loss to the migrated VM and its applications when migrating demanding workloads. Novel postcopy algorithms that utilize Remote Direct Memory Access, *RDMA*, have less performance penalty for retrieving pages [Hudiza and Shribman 2012] but so far no in-depth studies have been performed as to their performance. Hybrid approaches where a precopy stage is included in a postcopy algorithm have also been proposed [Sahni and Varma 2012, Song et al. 2014] but such algorithms have been studied even less than the postcopy approach. A survey of precopy and postcopy approaches can be found in [Kapil et al. 2013].

To advance the research field beyond incremental algorithmic improvements and enable live VM migration also when migrating VMs with demanding workloads, it is therefore important to gain a deeper understanding of the characteristics of live migration and its underlying principles, both in the precopy, the postcopy and the hybrid case. In this contribution we define five common criteria for live migration, *Continuous service, Low resource usage, Robustness, Predictability* and *Transparency*, that provide a means to describe and compare live migration algorithms. We then perform a performance evaluation where we migrate VMs running demanding workloads on high-performance testbeds to highlight performance properties and other characteristics for precopy, postcopy and hybrid live migration. In the evaluation, VMs with up to 14 GB of RAM running workloads with page dirtying rates up to 10 GB/s are live migrated while measuring migration downtime, total migration time, transmitted data volume and performance degradation during migration. Based on the results from the evaluation we investigate, categorize and compare the three approaches using the five migration properties. We also study under what circumstances the approaches fail to uphold the desired properties and investigate the underlying cause of the failure.

The algorithms used in the evaluation are the RDMA in-kernel postcopy Hecatonchire [Hudiza and Shribman 2012] algorithm which also supports hybrid live migration and the Yabsume [Hirofuchi et al. 2011] userspace postcopy approach which are both open source additions to the KVM hypervisor [RedHat 2009]. We use the standard KVM live migration algorithm to represent the precopy approach in the evaluation. Finally, we provide guidelines for which live migration approach to use depending on the desired performance and operational requirements, as well as outline the future directions for the live migration research community.

2. BACKGROUND

To migrate a VM, its state is transferred from the source to the destination host. Live migration is the concept of performing this transfer and switch the VM's execution to the destination host without a perceived interruption in service. There are two main approaches to live migration. Either, the state is transferred before execution is switched from source to destination, *precopy migration*, or execution is first switched and the state is then transferred on demand, *postcopy migration*.

2.1 Precopy Live Migration

The precopy approach [Clark et al. 2005] transfers the whole content of the source VM's RAM to the destination host before the VM is resumed on the destination side. Figure 1 outlines this process. This figure also illustrates two commonly used performance criteria: *total migration time* and *migration downtime*. The total migration time is the time from when the migration process is initiated until it completes and the migration downtime is the period during which the VM is suspended and thus not responding to requests.

The exact implementation of precopy migration varies between different hypervisors but most share the same basic concept of three phases, commonly referred to as the *initial phase*, the *iterative phase* and the *stop-and-copy phase*. During the initial phase, each of the VM's memory pages are transferred from the source to the destination. Because the source VM continues to run during this process, already transmitted pages might be written to, *dirtied*, before the last page is transferred. These dirty pages need to be re-transferred since the destination would otherwise have an incorrect version of the page. To achieve this, the algorithm moves to the iterative phase, where pages dirtied in the previous iteration are re-transferred, until the remaining number of pages to transfer is below a certain threshold or a maximum number of iterations is reached [Clark et al. 2005]. At the beginning of the final phase, stop-and-copy, the source VM is suspended to enable transfer of the last remaining pages without any further pages being dirtied. Finally, the CPU state is transferred and VM execution is resumed on the destination host.

Because both the source and destination VMs are stopped during the stop-and-copy phase, this phase must be very short as there is otherwise a high risk of a perceived service interruption to users. If the migration downtime is extensive e.g. due to the iterative phase ending due to too many iterations, TCP connections might drop, but even shorter downtimes can lead to issues. For example, in transaction processing applications, missed timers, unscheduled or delayed events or clock drift can lead to data corruption and/or a crash [Svärd et al. 2011]. It is therefore desirable to minimize the amount of data transferred during the stop-and-copy phase.

This fundamental precopy algorithm can also be applied to other areas. For example, the Slacker MySQL live migration system [Barker et al. 2012] uses an application-level precopy algorithm to migrate databases between hosts without interrupting transactions.

2.2 Postcopy Live Migration

An alternative approach to live migration is to switch the VM execution to the destination host at the beginning of the migration process and then transfer the memory pages as they are requested by the VM. An outline of this algorithm is shown in Figure 2. In order to start the VM at the destination, the CPU state, BIOS data and the Video RAM contents needs to be transferred. The VM's RAM is next allocated at the destination host but no memory content is transferred from the source at this time. The VM is then resumed and starts to access its memory, which is transferred in a on-demand manner from the source. Several techniques, implemented in kernel and/or user space, can then be used to trap the memory reads and writes and transfer missing memory pages from the source to the destination. To shorten the total migration time, many postcopy implementations also include a background process that pulls memory pages from the source in a sequential order when it is not requesting any other pages [Hirofuchi et al. 2011]

[Hines and Gopalan 2009]. Without this addition, the total migration time might be extended since the migration process would otherwise not complete until the VM has accessed every page.

The main motivation for postcopy migration is a short and stable migration downtime and a predictable total migration time, but as the pages have to be pulled over the network before the VM can access them, there is a risk of performance degradation to the VM and its applications after the VM is resumed.

2.3 Hybrid Live Migration

From a technical point of view, hybrid migration may be viewed as a special case of postcopy migration as it is a postcopy algorithm preceded by a limited precopy stage [Sahni and Varma 2012]. The idea is that if a subset of the most frequently accessed memory pages are transferred before the VM execution is switched to the destination, the performance degradation after the VM is resumed can be reduced because fewer pages need to be retrieved from the source. However, the precopy phase can lead to a slightly longer total migration time than for pure postcopy algorithms and it is also challenging to choose the correct set of pages for transfer.

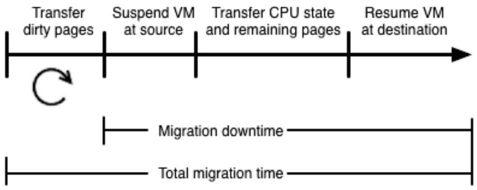

Figure 1: Precopy live migration.

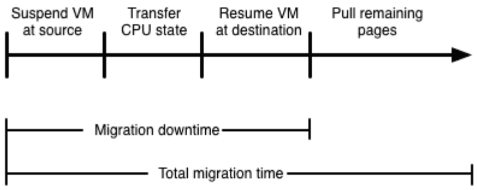

Figure 2: Postcopy live migration.

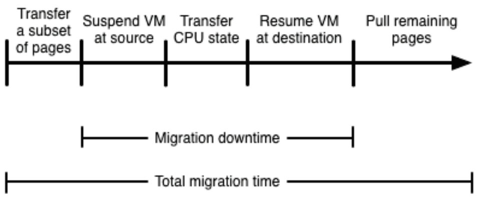

Figure 3: Hybrid live migration.

3. PROPERTIES AND CHALLENGES OF LIVE MIGRATION

However different in implementation, all approaches to live migration aim to fulfil the fundamental criteria of live migration, namely the ability to migrate the VM with no perceived interruption of service to the user [Clark et al. 2005]. In addition, the migration process should be invisible to the VM, its hosted applications and any connected peers both in terms of minimal performance degradation as well as any need to modify the software to support live migration. We summarize the fundamental aspects of live migration in a number of properties that are presented in this section. We also evaluate common challenges in live migration and how they relate to these properties.

3.1 Properties of Live Migration

In this section we propose the following five properties to be desired for live migration.

1. *Continuous service:* Service should be perceived as continuous by users of the VM's hosted applications despite migration downtime. In the case of interactive services, this means that the live migration process should not cause users to be disconnected or experience performance degradation to such a degree that it affects the normal execution of applications running on the VM. This means that the interruption in service must be short enough to not be perceivable by the users of services hosted in the migrating VM.

2. *Low resource usage:* The live migration process consumes resources on both the source and destination machines [Strunk 2012, Strunk and Dargie 2013] and this resource usage, sometimes referred to as *migration noise* [Koto et al. 2012], should be kept to a minimum. If an excessive amount of resources is consumed, performance and operation of applications in the VM as well as any co-hosted VMs might be affected, thus imposing performance penalties for the VM or any colocated VMs.

3. *Robustness:* The hypervisor, the VM or any hosted applications should not risk crashing or freezing due to the migration process.

4. *Predictability:* It should be possible to predict the duration of the migration process and how much resources it will consume. This includes predicting both the total migration time and the migration downtime as well as the amount of network and CPU resources required by the migration process on both the source and destination hosts. Postcopy migration is predictable in the sense that as the switch of execution always happens at the start of migration and the amount of transferred data is known.

5. *Transparency:* The VM's operating system and its hosted applications should not need to be migration aware. The migration process should be transparent to both the VM and any connected users and the performance of any hosted applications should not be affected.

Although an ideal live migration algorithm would fulfil all of these properties, this is not possible in practice. The continuous service property is central to live migration and unless met, the migration process cannot be considered live. However, the other criteria are more flexible and subject to compromises. For example, postcopy migration algorithms trade robustness for a shorter migration downtime and a more predictable migration process. Another example is hybrid algorithms that compromise transparency in order to make a better prediction as to which pages to transfer during the precopy phase [Lu et al. 2012].

3.2 Live Migration Challenges

The practical applicability of live migration is limited by three common problems that combined cause extended migration downtime, performance degradation and unpredictable behavior. In this section, we discuss three challenges that hamper live migration, the *transfer rate problem*, the *page re-send problem* and the *missing page problem* and how they affect the migration process with respect to the properties presented above.

3.2.1 The transfer rate problem

During the iterative phase of precopy live migration, the VM's pages are sent over the network from the source to the destination. As the source VM is running during this process, its memory contents is constantly updated. Because memory bandwidth is higher than network bandwidth, there is a high risk of memory pages being dirtied at a faster rate than they can be transferred over the network. In such cases, these pages are transferred repeatedly while the amount of dirty pages remaining to transfer does not decrease. This means that the migration process gets stuck in the iterative phase and as a result, the migration may have to be forced into the stop-and-copy phase with a large number of dirty pages remaining to transfer. As the VM is suspended during the stop-and-copy phase, this leads to extended migration downtime and a prolonged total migration time. Even in less severe cases, where the algorithm does not need to be forced to proceed to the stop-and-copy phase, downtime and total migration time are still extended to some degree.

The transfer rate problem poses a high risk to continuous service operation, as an extended migration downtime can lead to interruption of services and possibly disconnection of clients, lost database connections, or other issues. Even if the migration downtime is short enough for network connections not to drop (typically a few seconds for TCP connections over LANs or the internet), timing errors, missed triggers, etc, might occur and decrease the application's stability and performance. In our previous experiments with live migration of enterprise applications, downtimes as low as one second caused unrecoverable application problems [Hacking and Hudzia 2009, Svärd et al. 2011].

3.2.2 The page re-send problem

Live migration of a VM requires significant CPU and memory resources, although the heaviest load is put on the network. As a VM can easily have several gigabytes of RAM, a large amount of data is transferred during the live migration process. This problem is amplified in precopy migration as the source VM is running during the iterative phase and pages that have already been transferred are often being dirtied again. Since the state of the destination VM upon resume must be an exact copy of the source VMs state, these pages must be re-sent.

The page re-send problem was first discussed by Clark et al. Clark, Fraser, Hand, Hansen, Jul, Limpach, Pratt, and Warfield [2005] and can lead to excessive resource consumption as only the final version of a page is used and re-sending pages during migration consume both network and CPU resources. Furthermore, the page re-send problem is a challenge to the predictability criteria as it is not known beforehand the total number of pages that are to be re-transferred, making it difficult to estimate how long time a migration takes to complete.

Precopy migration is affected by both the page-resend and transfer rate problems. These problems are related as the transfer rate problem is a cause of the page-resend problem. However, factors like memory size, page dirtying rate and memory write patterns also affect the number of page resends [Clark et al. 2005, Svärd et al. 2011].

3.2.3 The missing page problem

Postcopy live migration algorithms resume the destination VM before its complete memory contents have been transferred to the destination host. After the execution is switched to the destination side, the missing pages are pulled over the network from the source. Because networks have lower bandwidth and higher latency than RAM, there is a performance penalty associated with accessing these missing pages. We refer to this phenomenon as *the missing page problem*. This residual dependency problem imposes a high risk of performance degradation for the hosted applications after the VM execution has switched to the destination host. If the performance degradation is severe, the transparency and continuous service objectives might not be met. The missing page problem also imposes a loss of robustness as it is not possible to fall-back to the source VM if the live migration fails, e.g. due to network disconnects that occur before the complete RAM content has been transferred. As the destination VM is not started until all memory pages are present in precopy methods, such algorithms are not affected by the missing page problem.

4. EXPERIMENTAL EVALUATION

To highlight the characteristics of the different live migration approaches a performance evaluation has been performed. The focus of this evaluation is to illustrate the conceptual differences between precopy, postcopy, and hybrid approaches in order to observe the characteristic behavior of the algorithms. Our experiments study various well-known metrics such as migration downtime, total migration time, and migration resource usage. We relate the impact of these metrics to the five live migration properties described in Section 3.1. The implementation of the evaluated algorithms are rather close to the definition given in Section 2 and the evaluation does not consider various modifications and improvements to the conceptual algorithms, including use of compression, caching, and page transfer reordering, topics that are further discussed in Section 6. We remark that although such techniques significantly improve the practical performance of a particular live migration algorithm, its main characteristics remains.

Two different workloads were used in the evaluation, one synthetic benchmark tailored specifically towards evaluation of live migration, *Appmembench* [A. Shribman 2012] and one real-world in-memory database application, SAP HANA [SAP 2013]. The chosen workloads are both demanding workloads as the difference in behavior between the migration approaches becomes more visible under demanding conditions. The Appmembench is a multi-threaded memory read/write application that repeatedly runs a defined memory access quote and every 100 ms reports a performance metric (a percentage). A complimentary mechanism runs an NTP client on the VM, enabling Appmembench to check whether the clock is advanced. Absence of the latter indicates VM downtime and 0 percent performance is reported for that time slot. To ensure soft real time performance and avoid potential locking problems, the SAP system used in the evaluation relies heavily on various timers and circuit breakers, which makes it very sensitive to variations in hardware performance (and thus also to VM migration). The tests were run on two testbeds, one with standard Gigabit Ethernet and one with IWARP [NetEffect 2012] networking to illustrate the algorithms behavior using both high-speed networking as well as during more typical conditions.

4.1 Algorithms

Three algorithms have been evaluated, one precopy algorithm and two postcopy algorithms. The two postcopy algorithms differ in implementation in that one does all transfer of memory pages in-kernel while the other one moves pages to userspace before they are transferred. The former algorithm includes an optional precopy stage, which means that it also supports hybrid migration. Both postcopy algorithms are modifications to the live migration algorithms in the KVM hypervisor assisted by custom kernel modules. The precopy algorithm evaluated in this contribution is the standard KVM live migration algorithm.

4.1.1 Precopy

The standard version of KVM 1.2.50 was used to demonstrate the precopy approach. The code was downloaded from the official git tree [Linux-KVM 2013] and built on the test machines.

4.1.2 Userspace postcopy

The open source *Yabusame* [Hirofuchi et al. 2011] implementation was chosen to represent the userspace postcopy approach (where pages are moved between kernelspace and userspace during migration). In Yabusame, the page fault handler is modified to pull the missing pages from the source where the source VM's RAM contents are exposed using a network block device server. In addition, an active pull optimization is used to ensure that pages are transferred continuously and not only moved upon page faults. The Yabusame implementation is outlined in Figure 4. In this figure, it can be seen that the migration process is controlled by the *qemu-kvm* processes on the source and destination hosts that communicate to coordinate the migration. The actual page transfer is handled by a userspace process on the destination side, *alloc_bg*. This process pulls the pages from an *xndb* server on the source host that reads from shared memory, where the VM's memory pages are stored. On the destination host, the *alloc_bg* process writes the pages to the VM using a custom kernel driver, *vmem*. This approach means that memory pages are being moved between kernel space and userspace during migration, which incurs a performance overhead.

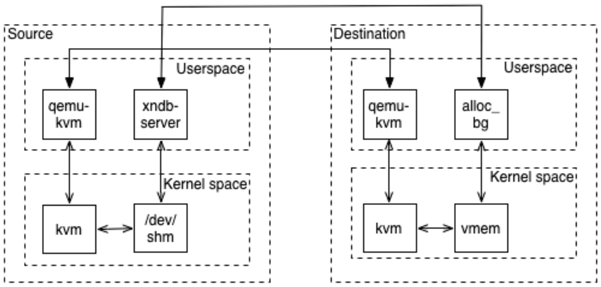

Figure 4: The Yabusame userspace postcopy approach.

4.1.3 In-kernel postcopy and hybrid

The Hecatonchire [Hecatonchire 2012], *Heca*, system represents the in-kernel postcopy approach where memory pages are transferred from the source to the destination host without moving them between kernel space and userspace. Since the Heca algorithm features an optional precopy phase it also supports hybrid live migration. Figure 5 shows the design of the system. As with Yabusame, migration is coordinated by the *qemu-kvm* processes on the source and destination hosts. If the VM tries to access a memory page that is not yet present, the VM traps a page fault and initializes a memory request, using the Heca kernel module. The page fault is transparently handled by the Heca kernel module on the virtual host that extracts the page from the destination over Remote Direct Memory Access, *RDMA*. When an application performs an RDMA read or write request, the application data is delivered directly to the network, reducing latency and thus enabling fast message transfers that run in parallel with other system operations. This means that pages are not transferred in and out of kernel space during migration. Similarly to Yabusame, the Heca approach use an active pull optimization.

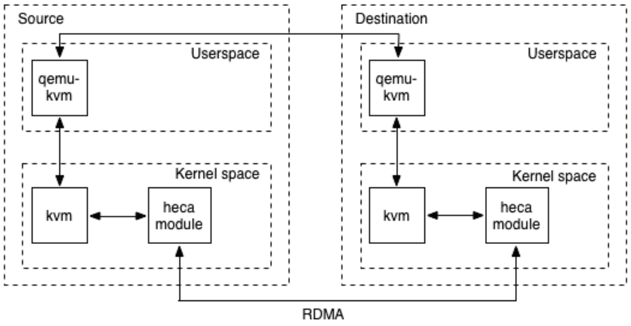

Figure 5: The Hecatonchire in-kernel postcopy approach.

4.2 Workloads

Two workloads were used in the performance evaluation. The first, Appmembench, is a benchmark for live migration algorithms while the second, SAP HANA, is a real-world in-memory database system with large memory requirements.

4.2.1 Appmembench

Appmembench is a benchmark for live migration that allocates and repeatedly writes blocks of memory while measuring the iteration time needed for the writes. It thus allows the rate at which memory pages are dirtied to be controlled accurately. In the evaluation, from 500 MB to 7 GB of memory was allocated to the benchmark. This benchmark is one variant of the *diabolical workload* [Clark et al. 2005], which is a workload that has a very high dirtying rate and is thus difficult to migrate. The source code for Appmembench is available online [A. Shribman 2012].

4.2.2 SAP HANA

SAP HANA is an in-memory database server. The software aggregates huge volumes of data and is typically very demanding in terms of CPU and memory usage. The SAP HANA benchmarking utility was used to put load on the servers by simulating users logging on and executing queries towards the server.

4.3 Testbeds

Two testbeds were used in the evaluation. In both testbeds, the VM disk images were kept on an NFS share accessible to both the source and destination hosts which means that storage migration was not needed. The hardware specifications of the testbeds are found in Table I. For the second testbed, the iWARP setup consist of 10GbE Chelsio T422CR network adapters connected to a Fujitsu 10GbE switch. Both testbeds used Linux stable 3.6.

Table 1: Hardware setup of the testbeds.

Testbed	CPU	RAM	Network
I	3.3 GHz Intel i5	16 GB	Gigabit Ethernet
II	3.3 GHz Intel i5	16 GB	iWARP

4.4 Scenarios

Three scenarios were included in the evaluation. The scenarios were chosen to study the characteristics of the different approaches while migrating demanding workloads and as such expose any shortcomings such as interruptions in service, unpredictable behavior, high resource usage and performance degradation. Table II contains an overview of the three scenarios. For policy reasons, the Yabusame postcopy algorithm could not be run on testbed II but the results from testbed I are sufficient to draw conclusion about its characteristics.

Table 2: Overview of the three evaluation scenarios.

	Evaluated Algorithms	Workload	Testbed
I	Postcopy (userspace), Hybrid, Precopy	Appmembench	I
II	Postcopy (kernel), Hybrid, Precopy	Appmembench	II
III	Postcopy (kernel), Hybrid, Precopy	SAP HANA	II

In all scenarios, the live migration process was performed as follows:

1. $\mathbf{t = 0}$ start the application inside the VM.
2. $\mathbf{t = 10\ s}$ initiate live migration of the VM.
3. $\mathbf{t = 70\ s}$ force the live migration to stop-and-copy if it has not completed.
4. $\mathbf{t = 120\ s}$ end scenario.

Notably, Step 3 is applied only when the live migration process gets stuck in the iterative phase which can happen due to the effects of the transfer rate problem, discussed in Section 3.2.1. The 70 s threshold was selected empirically, as the remaining working set size was stabilized after that time.

4.4.1 Appmembench on Gigabit Ethernet testbed

Scenario I compares traditional precopy migration with postcopy and hybrid migration. The Yabusame algorithm was used to demonstrate postcopy in this scenario and for the hybrid approach, the Heca algorithm was configured with a 5 s precopy phase. A 1 GB VM running Appmembench with 500 MB of allocated memory was migrated and the page dirtying rate was increased by a factor of 10 between runs, from 10 MB/s to 1 GB/s.

Results: Figure 6 shows the migration downtime in seconds. As seen, the migration downtime for the precopy algorithm increases as the page dirtying rate rises while the Yabusame userspace postcopy algorithm maintains a much shorter and more stable downtime even though a small increase is seen for Yabusame when the page dirtying rate is increased from 10 to 100 MB/s. The migration downtime for the Heca hybrid algorithm is short and almost the same in all runs.

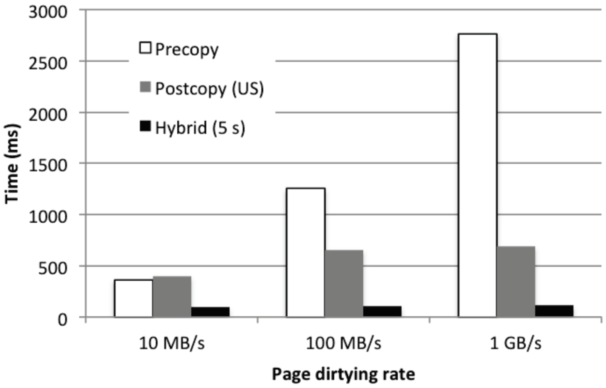

Figure 6: Migration downtime for Appmembench with 500 MB allocated memory on a 1 GB VM for varying page dirtying rates. This scenario uses Gigabit Ethernet.

In Figure 11, which shows the performance degradation measured by Appmembench during the migration process, it can be seen that while it takes longer for the precopy migration algorithm to finish, performance is close to 100% except for a brief interruption of services in the 1 GB/s case and a minor performance degradation in the 100 MB/s case. These degradations were both caused by the high page dirtying rates, which forced the algorithm to proceed to the stop-and-copy phase with too many pages left to be transferred. The result for the Yabusame postcopy algorithm is good with a dirtying rate of 10 MB/s (not shown in the figure) but in the 100 MB/s and 1 GB/s cases, severe performance degradation is measured for up to 23 seconds (from t=19 to t=42) after migration is initiated. This degradation is caused by the extensive copying of memory pages between user and kernel space in Yabusame. Finally, the Heca hybrid algorithm with a 5 second precopy stage shows no degradation of service in any of the runs.

Figure 7 shows the number of dirty pages that remains to be transferred when the source VM is suspended for a page dirtying rate of 1 GB/s. The page size is 4 KB in this, and all other, scenarios. As the Yabusame algorithm is a pure postcopy approach, no pages have been transferred before the suspend (with the exception of CPU-state, BIOS information and VRAM contents) so it is expected to have the highest number. What is interesting is that even though the precopy migration ran for 60 seconds before forced into the stop-and-copy phase, it still has more dirty pages remaining to transfer than the Heca hybrid algorithm that only had a 5 second precopy phase in this case.

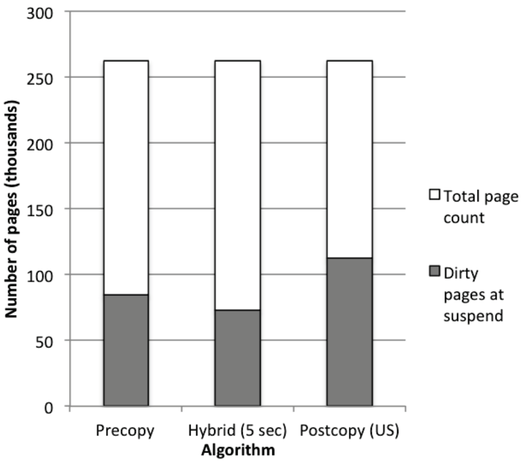

Figure 7: Number of dirty pages remaining to transfer when the source VM is suspended. The application is Appmembench with a page dirtying rate of 1 GB/s.

4.4.2 Appmembench on iWARP testbed

Scenario II compares precopy, postcopy and hybrid migration with respect to migration downtime and performance degradation during migration. The scenario investigates the effect of varying the working set size and page dirtying rate during migration as well as the duration of the precopy phase in hybrid migration. The Heca algorithm was used to demonstrate both the postcopy and hybrid approaches in this scenario.

Two tests were performed for each algorithm. First, Appmembench was run with a fixed page dirtying rate of 1 GB/s while the amount of memory allocated to the benchmark was varied from 500 MB up to 7 GB. The migration was run once for precopy and postcopy live migration and 3 times for hybrid migration, varying the precopy phase between 3, 5 and 10 seconds. The VM size ranged from 1 GB to 14 GB, i.e., twice the size of the amount of memory allocated to Appmembench.

In the second test, Appmembench was run with 5 GB of allocated memory and with page dirtying rates varying from 10 MB/s to 10 GB/s. The precopy algorithm was compared to hybrid migration with a 5 second precopy phase. A 10 GB VM was used in this test. The second test is basically the same test as Scenario I but performed on Testbed II which means that instead of Gigabit Ethernet, iWARP accelerated networking was used.

Results: The result from the first test is shown in Figure 8 that illustrates the downtime in milliseconds. In the first migration, with 500 MB allocated to Appmembench, all algorithms show roughly the same downtime. When the amount of allocated memory increases, they start to behave differently. Using the precopy algorithm, the migration downtime increases superlinearly with the amount of memory allocated to the Appmembench benchmark. The migration downtime for the postcopy and hybrid algorithms also increases although at a much slower rate, and it stays well below 500 ms in all runs. The shortest downtime in this test is achieved by the hybrid algorithm but notably, for small VM sizes (1 GB and 4 GB), the length of the precopy phase used by hybrid migration has no effect on the downtime. When the VM size increases to 10 GB and 14 GB, the migration downtime becomes shorter as the hybrid precopy phase is extended, with 10 seconds precopy giving the best performance for the largest VM size.

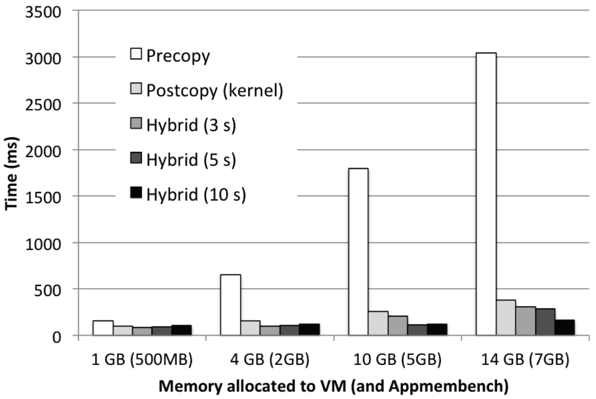

Figure 8: Migration downtime for precopy, postcopy and hybrid approaches running Appmembench, 1 GB/s dirtying rate. The amount of memory allocated to Appmembench ranges from 500 MB to 7 GB.

Figure 9 shows the migration downtime in milliseconds for the second test. As seen, the downtime for the precopy algorithm increases superlinearly with the dirtying rate, which is the same behaviour as in Scenario I, while the hybrid algorithm with 5 s precopy stage maintains a stable downtime, below 250 ms in all runs. However, in the 10 MB/s case, the precopy algorithm achieves a shorter downtime.

Figure 10 shows the total migration time for the precopy algorithm in Scenario II. With a page dirtying rate of 10 MB/s, the precopy algorithm finished after 6 and 24 seconds with 500 MB and 5 GB allocated to Appmembench, respectively. In all other cases, the migration was forced into the stop-and-copy phase after 60 seconds as the number of dirty pages did not decrease.

4.4.3 SAP HANA on iWARP testbed

The purpose of Scenario III is to evaluate how precopy, postcopy, and hybrid migration behave in terms of migration

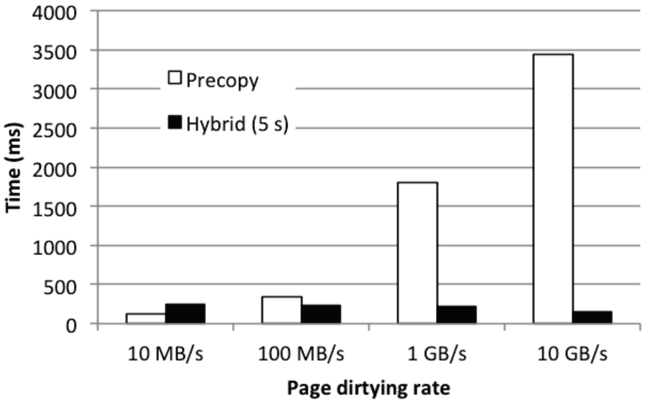

Figure 9: Migration downtime for precopy and hybrid running Appmembench with 5 GB allocated memory. The page dirtying rate ranges from 10 MB/s to 10 GB/s.

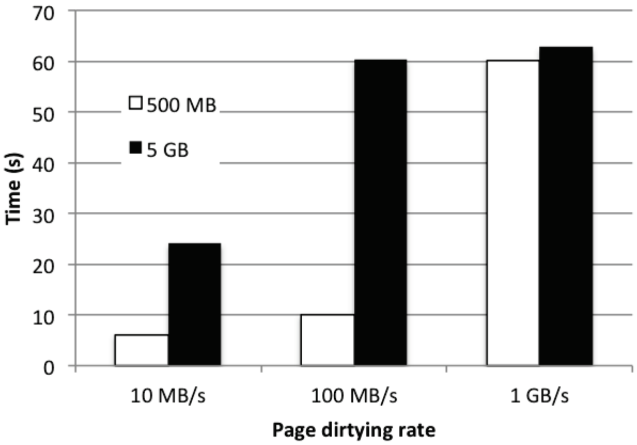

Figure 10: Total migration time for the precopy algorithm running Appmembench with 500 MB and 5 GB allocated memory. Page dirtying rate varies from 10 MB/s to 1 GB/s.

downtime when live migrating a database-driven, memory intensive business application. A 14 GB VM running a SAP HANA instance was used for this purpose. The SAP HANA benchmarking utility was used to put load on the servers by simulating users logging on and sending queries to the server. In total 5 runs were made, one run each using precopy and postcopy migration and 3 runs with hybrid migration, varying the duration of the precopy phase between 3, 5 and 10 seconds. The Heca algorithm represents both postcopy and hybrid migration in this scenario.

Results: The results for Scenario III are shown in Figure 12. As seen, the behavior of the algorithms is similar to the earlier scenarios in that the precopy algorithm shows a much longer migration downtime than both hybrid and postcopy. However, it is interesting that there is no difference between hybrid and postcopy in terms of migration downtime which means that the duration of the precopy phase had little effect in this scenario.

5. COMPARISON OF THE APPROACHES

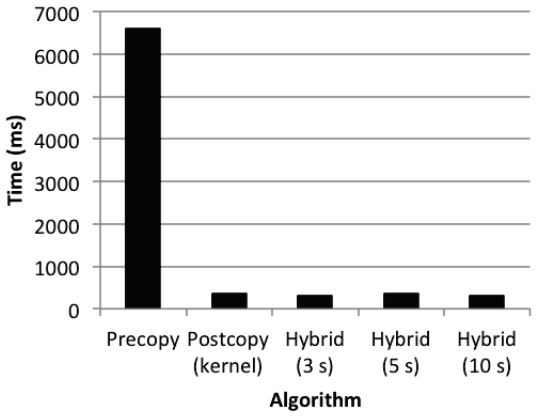

Figure 12: Migration downtime for precopy, postcopy and hybrid, running SAP HANA. The VM size is 14 GB.

The algorithms in the evaluation are all designed to perform efficient live migrations with minimal downtime but differences in implementation and underlying approach mean that their results in the evaluation differed greatly. Based on these results, we examine how the different live migration algorithms meet the live migration properties presented in Section 3.1 while migrating demanding workloads and how they are affected by the three live migration challenges outlined in Section 3.2.

5.1 Continuous service

All algorithms evaluated in this contribution fulfill the continuous service objective while migrating non-demanding workloads, however, as seen in the evaluation once the algorithms are stressed, they start to behave differently. While migrating demanding workloads the precopy algorithm can no longer uphold the continuous service criteria. As illustrated in Figures 6, 8 and 9, the migration downtime for the precopy algorithm increases with both the page dirtying rate and the size of the working set which leads to a perceived interruption in service. The cause of this is the transfer rate problem, which is more likely to arise when migrating VMs running heavy workloads.

Considering the postcopy approaches, the Yabusame algorithm also fails to meet the continuous service property when the dirtying rate is high. This is visible in Figure 11 which shows mild to severe performance degradation for Yabusame migration after the execution is switched to the destination host. The reason for this performance degradation is the missing page problem and to rectify this to some extent, most postcopy implementations, including Yabusame and Heca, use a pre-caching mechanism to predict which pages are needed by the VM in the near future based previous requests. Those pages are pulled from the source even if they are not requested by the VM. However, such a modification does not completely solve the problem as firstly, no pages are present directly after the VM is resumed, and secondly, the prediction of which pages are needed right after resume tend to be difficult and error-prone.

Hybrid live migration can in theory reduce the effect of the missing page problem as a subset pages are transferred dur-

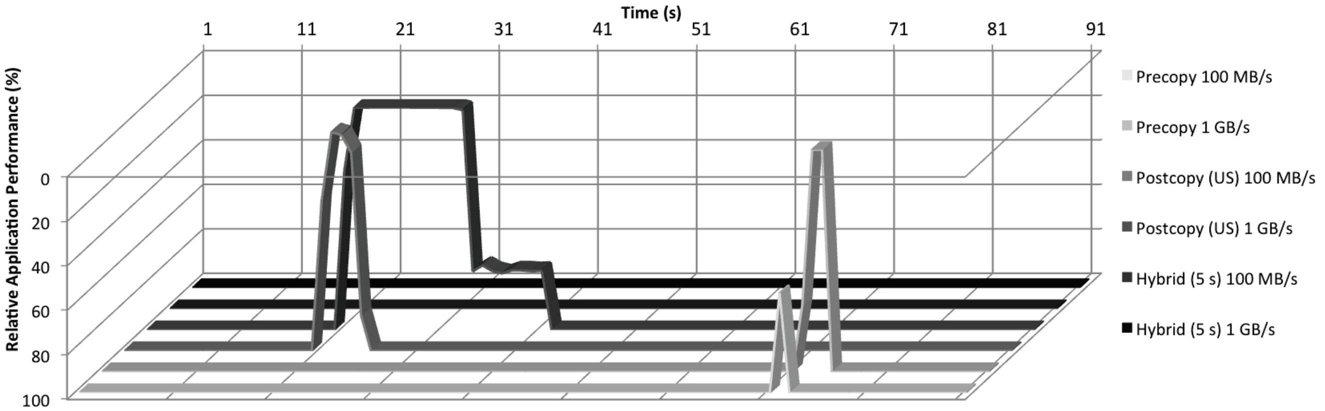

Figure 11: Relative application performance degradation during migration for precopy, postcopy and hybrid running Appmembench with page dirtying rates 100 MB/s and 1 GB/s.

ing the precopy phase before the execution is switched to the destination host. If this subset is chosen so that the most frequently accessed pages are transferred, the risk of a page not being present when it is requested by the destination VM after the switch is reduced compared to a pure postcopy algorithm. As the performance penalty for retrieving a page from the source host over the network is high, the risk of post-migration performance degradation is reduced for hybrid migration compared to pure postcopy migration. This effect is seen in Figure 11 which demonstrates that the in-kernel Heca hybrid algorithm caused no performance degradation during migration in any of the runs.

Notably, the duration of the precopy phase needs to be tuned for optimal downtime. Even though the hybrid algorithm obtained the shortest downtimes in the benchmark migration tests throughout the evaluation, increasing the precopy phase from 5 to 10 seconds actually resulted in a longer downtime in Scenario II as illustrated in Figure 8, where a smaller amount of memory was allocated to Appmembench. The opposite was true for the runs with more memory allocated. This is because even though there is less memory unmapped for smaller VMs, it is more fragmented and therefore requires more *ioctl* requests which increases the downtime.

The drawback with the hybrid approach is that it is difficult to choose the correct set of pages to transfer during the precopy phase. In the real world application case, there is very little difference between the pure postcopy and the hybrid algorithms as seen in Figure 12. This is most likely due to the more unpredictable nature of the SAP HANA application compared to the synthetic benchmark which makes it difficult to select suitable pages to transfer during the precopy phase. Transferring the pages that are most frequently written is not sufficient as pages that are frequently read must also be considered. Tracking page reads is more expensive performance-wise than tracking writes [Lu and Shen 2007] and would thus risk reducing the performance of the VM, its applications and any co-located VMs.

Finally, hybrid migration is still affected by the missing page problem and if stressed beyond its limits it might be subject to performance degradation after the execution switch, even though this was not seen in this evaluation. However, since the evaluation used heavy workloads with a high page dirtying rate, it is not likely that such performance degradation would occur under more normal circumstances.

5.2 Low resource usage

Postcopy approaches are usually lean on resource usage since they transfer each memory page only once. Even though the hybrid approach includes a time-limited precopy stage, only a small subset of the VM's pages are transferred during this stage which means that the overhead in resource usage compared to pure postcopy is usually small.

In contrast, precopy algorithms often use significant resources. As illustrated by Figure 7, the precopy algorithm, which was forced into the stop-and-copy phase after 60 s in this scenario, has more dirty pages remaning to transfer than the hybrid algorithm, which only ran for 5 seconds. Notably, the postcopy algorithm which does not transfer any pages before the VM is suspended only a had a slightly larger amount of dirty pages remaning to transfer. As the network bandwidth is 1 GB/s, i.e. the same as the page dirtying rate, this scenario is an example of when the transfer rate problem takes effect. The precopy migration algorithm is stuck in the iterative phase where pages are being transmitted over and over again, but the amount of dirty pages does not decrease between iterations which means that the migration process is wasting resources.

5.3 Robustness

Precopy approaches like the KVM algorithm used in the evaluation are robust since if migration fails, it is possible to fall-back to the source host. However, for postcopy and hybrid approaches, the situation is different as they sacrifice robustness to minimize downtime and gain predictability. Unless the source host remains available during the whole process, the destination VM suffers an unrecoverable crash as there is no way to transfer the remaining memory pages. This is the case with both the Heca (hybrid and postcopy) and the Yabusame algorithms. As seen in the evaluation, this leads to a short and stable migration downtime and a predictable behavior, but at the expense of robustness.

5.4 Predictability

It is much easier to predict the behavior of a postcopy than a precopy algorithm since postcopy algorithms transfer each memory page once only. Additionally, the postcopy approach has the benefit of an almost immediate switch of execution as only the BIOS information, CPU state and the Video RAM contents are transferred before the context switch. This information makes up only a small part of the total RAM and the transfer is usually very fast. Postcopy migration is therefore predictable as the switch of execution always happens at the start of migration and the amount of data that is transferred is known. In the hybrid case where a precopy stage is included in postcopy migration, the duration of this precopy phase is known. This means that the hybrid algorithms behavior is also predictable. Note that predictability refers to migration downtime, resource usage and total migration time. Post-migration service interruption caused by the missing page problem is not considered as a part of predictability but belong to the continuous service criteria. In the evaluation, both the postcopy and hybrid approaches showed consistently predictable behavior in all scenarios.

Precopy algorithms are much more un-predictable since they might transfer each memory page more than once, making it very hard to predict migration time and resource usage. In addition to the greatly varying migration downtime of the precopy approach in the evaluation, Figure 10 illustrates that the total migration time is also hard to predict since there is no obvious pattern in its duration.

5.5 Transparency

The transparency criteria is fulfilled by all of the algorithms studied in this contribution, precopy, postcopy and hybrid, since none of them require any modifications to be made to the guest OS or any hosted applications. Benefits and drawback of alternative, non-transparent, approaches are discussed in Section 6.

5.6 Comparison summary

Table 3, summarizes the criteria that are met by the algorithms in the evaluation. The main benefit of precopy migration is that it is robust. If the migration fails, it is possible to fall back to the source host, provided failure of this host was not the cause of the problem. If the VM to be migrated is not too heavy in terms of CPU and/or memory usage, and enough network bandwidth is available, the standard precopy live migration algorithms of modern hypervisors work well. The precopy algorithm is also mature and migrates non-demanding workloads with acceptable downtime [Clark et al. 2005] over LANs. The downside with precopy migration is that it is subject to both the transfer rate and the page re-send problem. This means that continuous service cannot be guaranteed and when migrating VMs running demanding workloads interruption of service is quite likely. This also means that there is a risk of excessive resource consumption, longer total migration time and less predictable behavior compared to postcopy approaches.

Several attempts have been made to improve the precopy algorithm. The transfer rate problem has been tackled by increasing migration throughput with compression [Svärd et al. 2011, Wood et al. 2011], data de-duplication [Wood et al. 2011, Zhang et al. 2013], use of pass-through network devices [Pan et al. 2012] and parallelizing primitive operations [Song et al. 2013] while the page re-send problem has been addressed by non-sequential transfer order [Hudiza and Shribman 2012] and log-based migration [Hu et al. 2012]. However, none of these improvements completely remedy the transfer rate or page-resend problem. Thus, the risk of extended migration and excessive resource usage remains also for an improved precopy migration algorithm.

Table 3: Summary of live migration algorithms and criteria.

	Precopy	Postcopy	Hybrid
Continuous service		(✓)	✓
Low resource usage		✓	✓
Robustness	✓		
Predictability		✓	✓
Transparency	✓	✓	✓

In contrast to precopy migration, the postcopy approach is not robust as it is not possible to fall back to the source VM as this one is suspended at the beginning of migration and its state thus is inconsistent with the destination copy. The benefits with postcopy live migration is the short and stable migration downtime and that postcopy algorithms are not subject to the page-resend problem. However, because the postcopy approach is affected by the missing page problem, there is a risk of performance degradation after the execution has been switched to the destination host. If this performance degradation is large enough, the continuous service criteria may not be met. This can be observed in the evaluation where the Yabusame postcopy algorithm had issues with post-migration performance degradation for page dirtying rates higher than 100 MB/s. This is probably caused by the implementation of that algorithm where unnecessary overhead is added by copying memory pages between kernelspace and userspace.

The approach that showed the overall best results in the evaluation was the hybrid approach. The Heca hybrid algorithm with a 5 s precopy phase had a short, stable migration downtime in all scenarios and showed no performance degradation after the VM was resumed on the destination. However, in Scenario III, there was no difference between the postcopy and hybrid versions of the Heca algorithm so it is thus possible that the biggest performance gain in Heca comes from using the in-kernel zero-copy RDMA approach that enables the network adapter to transfer data directly to or from application memory.

To summarize, precopy should be used only if robustness cannot be sacrificed. In all other cases it is better to use postcopy or hybrid migration. The predictable nature and low resource usage of such approaches means that reliable migrations can be performed without wasting resources and the short, stable migration downtime makes these approaches better suited for migrating demanding workloads.

6. RELATED WORK

Since live migration was first demonstrated [Clark et al. 2005], there have been many attempts to improve the performance of the precopy migration algorithm and several algorithms that tackle the transfer rate problem by increasing migration throughput have been suggested. The *XBRLE* [Svärd et al. 2011] and *XBZRLE* [Hudiza and Shribman 2012] delta compression algorithms as well as CloudNet [Wood et al. 2011] are examples of such approaches. Delta compression is the idea of transferring changes to memory pages, *deltas*, instead of the full page contents. When a page is transferred during the iterative phase of precopy migration, it is stored in a cache. If the same page is dirtied again, a compressed delta page can be computed and transferred instead of the full page. Delta compression has also been used for cold migration of HPC clusters [Anedda et al. 2010].

Another way to increase throughput is by using data deduplication techniques. CloudNet [Wood et al. 2011], is a migration solution for cloud environments and routes the migration traffic through a proxy. The proxy calculates a checksum for each page and keeps track of duplicate pages, which are sent as references, thus reducing the amount of transferred data. Using this technique, throughput can be increased when migrating multiple instances of VMs with similar memory contents. A similar approach is used by Jo, Gustafsson, Son, and Egger [2013]. Their algorithm tracks the VM's I/O to the network storage device and maintains an updated mapping of memory pages that reside in identical form on the storage device. During the iterative phase of precopy migration, the destination host fetches those memory pages directly from the storage device instead of transferring those pages over the network from the source. Another data de-duplication approach is MiyakoDori [Akiyama et al. 2012], geared towards data centers where VMs are sometimes migrated back and forth between hosts for server consolidation purposes. The MiyakoDori system takes advantage of this by keeping a memory image of a migrated VM on the source host. If the VM is then migrated to a host where it has been provisioned before, pages that are identical with those in the saved image are not transferred. Similar ideas are used by Cui, Li, Li, Huai, Hu, Wo, Al-Aqrabi, and Liu [2013] in the VMScatter system which is an one-to-many migration method to migrate VMs from a single source to multiple destinations simultaneously. Deshpande, Kulkarni, and Gopalan [2012] also utilize data deduplication techniques to reduce the traffic load on the core network links. They suggest a distributed system for inter-rack live migration, *IRLM*, that aims to reduce the traffic load on the core network links during mass VM migration through distributed deduplication of the VMs memory images.

A somewhat different approach to increase migration throughput is an approach where the state between the source and the destination is synchronized with the help of log files [Hu et al. 2012]. Memory writes are traced on the source and replayed on the destination. In their experiments, resource usage was reduced compared with the precopy algorithm which indicate that their approach to some extent remedy the page-resend problem. Migration throughput can also be increased by applying data and pipeline parallelism to the precopy algorithm. Song, Shi, Liu, Yang, and Chen [2013] propose the PMigrate algorithm where several threads are spawned to handle the most time-consuming primitive operations during live migration.

To tackle the page re-send problem, Hudiza and Shribman [2012] propose a Least Recently Used, *LRU*, page reordering technique. The algorithm is a development of the page transfer reordering algorithm presented by Svärd, Tordsson, Hudzia, and Elmroth [2011] and aims to reduce the number of page re-sends by first sending pages which have a smaller likelihood of changing again in the near future. The number of page re-sends can be significantly reduced when pages are updated with no regular pattern. LRU page reordering is designed to give no degradation in performance even when memory is updated in a sequential sweep. A similar approach geared towards storage migration is proposed by Zheng, Ng, and Sripanidkulchai [2011] who use an improved VM disk storage block live migration algorithm that includes an analysis of the write history to storage blocks. They use this information to schedule the transfer order of the blocks, thereby reducing the amount of storage data being transferred during live migration. The SonicMigration [Koto et al. 2012] algorithm uses a selective approach when transferring memory pages. Free pages and soft-state pages, i.e. caches for disk blocks etc are not transferred since they can be re-created by the destination hypervisor. If a significant part of the VMs memory consists of such pages, total migration time can be significantly reduced.

In an effort to reduce resource usage for precopy migration Xu, Yao, Wang, and Zhou [2012] propose a network bandwidth cost function that capture the tradeoff between migration overhead and service degradation based on queuing theory. The idea is to minimize the resource usage during migration while maintaining a short migration downtime thereby reaching a tradeoff between migration performance and delivering an acceptable quality of service. In related effort by Liu, Jin, Xu, and Liao [2013], linear regression is used to model performance and energy consumption for pre-copy algorithms. Hu et al. [Hu et al. 2013] develop an automated testing framework for live migration and use this to evaluate memory and storage migration for virtualization systems such as KVM, Xen, VMWare, and Hyper-V.

HSG-LM [Lu et al. 2012] is a hybrid migration technique implemented in the guest OS kernel. They reason that guest OS can make a more accurate prediction as to what pages will be accessed in the near future than the hypervisor as the hypervisor does not have access to data of the same granularity. Because the guest OS is modified and aware of the migration, the HSG-LM thus sacrifices the transparency with the goal of delivering better performance.

In most cases, migration is performed within the same subnet, *LAN* migration. In these cases, the VM's disk image(s) can be kept on a network storage device accessible from both the source and destination hosts which means that only the VM's memory contents need to transferred during migration. However, VMs can also be migrated over WAN links. As it is impractical to use shared storage devices in such cases, this makes WAN-migration more cumbersome than LAN migration because the VM's storage has to be migrated [Hirofuchi et al. 2009, Zheng et al. 2011, Nicolae and

Cappello 2012], as well as the VM's IP traffic that must be rerouted [Wood et al. 2011] as it no longer resides on the same subnet. WAN migration has been successfully demonstrated in several cases [Ramakrishnan et al. 2007, Wood et al. 2011] and is implemented in Microsoft Hyper-V [Microsoft 2012]. The VMFLock system achieves migration of groups of VMs across data centers through a combination of data deduplication and accelerated instantiation, where VMs are booted before all data is present at the destination site [Al-Kiswany et al. 2011].

Live migration can also be used as a tool to improve datacenter performance. The sandpiper [Wood et al. 2007] system uses live migration to resolve overloaded servers in datacenters. Sandpiper monitors server performance using a combination of black box as well as grey box techniques and uses live migration to re-arrange the physical mapping if necessary. Similar ideas are utilized in the Reactive Cloud [Hirofuchi et al. 2012] system in which VMs are migrated using postcopy migration to achieve a dynamic consolidation of VMs in a datacenter as well as by Chuan, Huaxiang, Zhilou, Ying, and Lina [2012] in a study of virtual machine migration strategies.

A similar use case is applying live migration for high availability, where state updates asynchronously are propagated from a primary to secondary VM at millisecond frequency. By combining this synchronization with speculative execution, fast and transparent fail-over can be achieved [Cully et al. 2008]. Similar methods have been used also to achieve high-availability of database systems by workload-aware memory synchronization methods [Minhas et al. 2013].

7. FUTURE DIRECTIONS

In this section, we outline a few future directions for the live migration research community.

7.1 Dynamic context switching

Dynamic context switching is the concept of allowing execution to move dynamically between the source and destination hosts instead of performing a single context switch at a certain point in the migration process. As the CPU state is small and can easily be migrated, these dynamic context switches can be performed almost instantaneously. For example, after a certain amount of pages have been transferred using a pre-copy stage, the execution is switched over to the destination side while pages continue to be migrated in the background. If performance is degraded because the VM requests a lot of pages that are not present, execution is switched back to the source. The execution then moves back and forth as needed until the migration process is complete. The dynamic context switching algorithm can be considered a seamless hybrid of precopy and postcopy migration.

7.2 VM phase detection

It has been demonstrated that applications normally run in cycles, repeating a number of tasks following a number of normal patterns. It is also possible to detect which phases an application consists of and to determine which phase it is currently in [Sembrant et al. 2012]. If this technique can be modified to detect the phases of a VM's execution, this information can be used to improve the performance of live migration. For example, a postcopy migration could be initiated when the VM is at the beginning of the least CPU and/or memory intensive phase. As the dirtying rate is at its lowest at that time, the impact of the missing page problem is reduced. This approach has been studied by Zhihong et al. who examine the average utilization of the CPU, memory, I/O and network bandwidth of a VM to make a series of decisions about migration, such as whether a migration is triggered, what virtual machine should be migrated, and to which host [Li et al. 2012]. The idea is also is somewhat related to the HSG-LM technique, mentioned above, but the goal is to address the transfer rate problem without breaching the transparency criteria.

8. CONCLUSION

So far, studies of live migration typically focus on implementation details and performance optimization, most commonly for precopy migration. In contrast, this paper studies the underlying concepts and fundamental characteristics, as well as introduces desired properties of live migration algorithms that can be used to gain a deeper understanding about the principles and performance of live migration. With a starting-point in these properties we analyze the benefits and drawbacks of three live migration algorithms: precopy, postcopy and hybrid.

In the study of migration approaches performed in the paper, it is seen that although precopy approaches are robust, widely available and proven, they fail in delivering continuous service throughout the migration process when migrating demanding workloads. As validated by the evaluation, precopy migration is also unpredictable and consume more resources than necessary. Although many attempts of improving the precopy approach have been made, none of these completely solve the underlying issues associated with this technique. On the other hand postcopy approaches, including hybrid, are lean in terms of resource usage and deliver a short and predictable migration downtime. The evaluation in this contribution indicate that novel in-kernel postcopy algorithms are able to migrate demanding workloads with no interruption in service or performance degradation. The downside of postcopy migration, including hybrid, is the lack of robustness as it is not possible to fall back to the source host if migration fails.

To summarize, due to its limitations, precopy live migration should be used only when robustness cannot be sacrificed. In all other cases postcopy or hybrid migration provide live migration with a short downtime, minimal resource usage and a predictable total migration time. We foresee that support for postcopy migration will be included in future versions of mainstream hypervisors.

Acknowledgments

We acknowledge Eliezer Levy, Aidan Shribman and Tomas Ögren for their contributions to this work. Financial support has been provided by the EC FP7 under grant agreements no. 215605 (RESERVOIR) and 257115 (OPTIMIS) as well as by UMIT research lab and the Swedish Government's strategic research project eSSENCE.

9. REFERENCES

[A. Shribman 2012] A. Shribman. 2012. Appmembench source code. https://github.com/hecatonchire/heca-misc/tree/master/bench/appmembench, Visited on 2013-01-24.

[Akiyama et al. 2012] Akiyama, S., Hirofuchi, T., Takano, R., and Honiden, S. 2012. MiyakoDori: a memory reusing mechanism for dynamic VM consolidation. In *CLOUD '12 IEEE 5th International Conference on Cloud Computing*. 606 –613.

[Al-Kiswany et al. 2011] Al-Kiswany, S., Subhraveti, D., Sarkar, P., and Ripeanu, M. 2011. Vmflock: virtual machine co-migration for the cloud. In *20th international symposium on High Performance Distributed Computing (HPDC)*. ACM, 159–170.

[Anedda et al. 2010] Anedda, P., Leo, S., Manca, S., Gaggero, M., and Zanetti, G. 2010. Suspending, migrating and resuming hpc virtual clusters. *Future Generation Computer Systems 26,* 8, 1063–1072.

[Barker et al. 2012] Barker, S., Chi, Y., Moon, H. J., Hacigümüş, H., and Shenoy, P. 2012. "Cut me some slack": latency-aware live migration for databases. In *EDBT '12: 15th International Conference on Extending Database Technology*. ACM, 432–443.

[Chuan et al. 2012] Chuan, C., Huaxiang, Z., Zhilou, Y., Ying, F., and Lina, L. 2012. A new live virtual machine migration strategy. In *ITME '12: International Symposium on Information Technology in Medicine and Education*. 173–176.

[Clark et al. 2005] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. 2005. Live migration of virtual machines. In *NSDI '05: 2nd Symposium on Networked Systems Design and Implementation*. ACM, 273–286.

[Cui et al. 2013] Cui, L., Li, J., Li, B., Huai, J., Hu, C., Wo, T., Al-Aqrabi, H., and Liu, L. 2013. VMScatter: migrate virtual machines to many hosts. In *VEE '13 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 63–72.

[Cully et al. 2008] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. 2008. Remus: High availability via asynchronous virtual machine replication. In *5th USENIX Symposium on Networked Systems Design and Implementation*. 161–174.

[Deshpande et al. 2012] Deshpande, U., Kulkarni, U., and Gopalan, K. 2012. Inter-rack live migration of multiple virtual machines. In *VTDC '12: 6th international workshop on Virtualization Technologies in Distributed Computing Date*. ACM, 19–26.

[Hacking and Hudzia 2009] Hacking, S. and Hudzia, B. 2009. Improving the live migration process of large enterprise applications. In *VTDC '09: 3rd International Workshop on Virtualization Technologies in Distributed Computing*. ACM, 51–58.

[Hecatonchire 2012] Hecatonchire. 2012. The Hecatonchire Project. http://www.hecatonchire.com, Visited on 2013-01-24.

[Hines and Gopalan 2009] Hines, M. R. and Gopalan, K. 2009. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE '09: The 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, 51–60.

[Hirofuchi et al. 2011] Hirofuchi, T., Nakada, H., Itoh, S., and Sekiguchi, S. 2011. Reactive consolidation of virtual machines enabled by postcopy live migration. In *VTDC '11: Workshop on Virtualization Technologies in Distributed Computing*. IEEE, 11–18.

[Hirofuchi et al. 2012] Hirofuchi, T., Nakada, H., Itoh, S., and Sekiguchi, S. 2012. Reactive cloud: Consolidating virtual machines with postcopy live migration. *IPSJ Transactions on Advanced Computing Systems 5*, 34–46.

[Hirofuchi et al. 2009] Hirofuchi, T., Ogawa, H., Nakada, H., Itoh, S., and Sekiguchi, S. 2009. A live storage migration mechanism over wan for relocatable virtual machine services on clouds. In *CCGRID '09: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 460–465.

[Hu et al. 2012] Hu, L., Zhao, J., Xu, G., Chang, D., and Chu, J. 2012. A fast convergent live migration of virtual machines. *Journal of Information and Computational Science 9*, 4405?–4412.

[Hu et al. 2013] Hu, W., Hicks, A., Zhang, L., Dow, E. M., Soni, V., Jiang, H., Bull, R., and Matthews, J. N. 2013. A quantitative study of virtual machine live migration. In *ACM Cloud and Autonomic Computing Conference*. ACM.

[Hudiza and Shribman 2012] Hudiza, B. and Shribman, A. 2012. Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications. In *VHPC '12: 7th Workshop on Virtualization in High-Performance Cloud Computing*.

[Jo et al. 2013] Jo, C., Gustafsson, E., Son, J., and Egger, B. 2013. Efficient live migration of virtual machines using shared storage. In *VEE '13 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 41–50.

[Kapil et al. 2013] Kapil, D., Pilli, E. S., and Joshi, R. C. 2013. Live virtual machine migration techniques: Survey and research challenges. In *2013 IEEE 3rd International Advance Computing Conference (IACC)*. IEEE, 963–969.

[Kim et al. 2013] Kim, J., Chae, D., Kim, J., and Kim, J. 2013. Guide-copy: fast and silent migration of virtual machine for datacenters. In *SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM.

[Koto et al. 2012] Koto, A., Yamada, H., Ohmura, K., and Kono, K. 2012. Towards unobtrusive vm live migration for cloud computing platforms. In *APSYS '12: Proceedings of the Asia-Pacific Workshop on Systems*. ACM, 7:1–7:6.

[Li et al. 2012] Li, Z., Luo, W., Lu, X., and Yin, J. 2012. A live migration strategy for virtual machine based on performance predicting. In *Computer Science Service System (CSSS), 2012 International Conference on*. 72–76.

[Linux-KVM 2013] Linux-KVM. 2013. Userspace git tree. git://git.kernel.org/pub/scm/virt/kvm/qemu-kvm.git, Visited on 2013-01-24.

[Liu et al. 2013] Liu, H., Jin, H., Xu, C.-Z., and Liao, X. 2013. Performance and energy modeling for live migration of virtual machines. *Cluster computing 16,* 2, 249–264.

[Liu et al. 2008] Liu, P., Yang, Z., Song, X., Zhou, Y., Chen, H., and Zang, B. 2008. Heterogeneous live migration of virtual machines. In *IWVT '08: International Workshop on Virtualization Technology*.

[Lu et al. 2012] Lu, P., Barbalace, A., and Ravindran, B. 2012. Hybrid and Speculative Guest OS Live Migration without Hypervisor. Tech. rep., ECE Dept., Virginia Tech.

[Lu and Shen 2007] Lu, P. and Shen, K. 2007. Virtual machine memory access tracing with hypervisor exclusive cache. In *ATC '07: The 4th International Conference on Autonomic and Trusted Computing*. USENIX Association, 3:1–3:15.

[Microsoft 2012] Microsoft. 2012. Windows Server 2012 Server Virtualization. `bit.ly/Xz8pFE`, Visited on 2013-01-24.

[Minhas et al. 2013] Minhas, U. F., Rajagopalan, S., Cully, B., Aboulnaga, A., Salem, K., and Warfield, A. 2013. Remusdb: Transparent high availability for database systems. *The VLDB Journal 22,* 1, 29–45.

[Mohan and S 2013] Mohan, A. and S, S. 2013. Survey on live vm migration techniques. *International Journal of Advanced Research in Computer Engineering & Technology*.

[NetEffect 2012] NetEffect. 2012. Understanding iWARP. `http://www.cse.ohio-state.edu/~panda/788/papers/1k_understanding_iwarp.pdf`, Visited on 2013-01-24.

[Nicolae and Cappello 2012] Nicolae, B. and Cappello, F. 2012. A hybrid local storage transfer scheme for live migration of i/o intensive workloads. In *HPDC '12: 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 85–96.

[Pan et al. 2012] Pan, Z., Dong, Y., Chen, Y., Zhang, L., and Zhang, Z. 2012. CompSC: live migration with pass-through devices. In *VEE '12: Eighth Annual International Conference on Virtual Execution Environments*. ACM.

[Ramakrishnan et al. 2007] Ramakrishnan, K. K., Shenoy, P., and Van der Merwe, J. 2007. Live data center migration across WANs: a robust cooperative context aware approach. In *INM '07: The ACM SIGCOMM Workshop on Internet Network Management 2007*. ACM, 262–267.

[RedHat 2009] RedHat. 2009. KVM - kernel-based virtualization machine white paper. `http://red.ht/IqDYvb`, Visited on 2013-01-24.

[Sahni and Varma 2012] Sahni, S. and Varma, V. 2012. A hybrid approach to live migration of virtual machines. In *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 1–5.

[SAP 2013] SAP. 2013. SAP HANA. `http://bit.ly/GKZkDy`, Visited on 2013-02-13.

[Sembrant et al. 2012] Sembrant, A., Black-Schaffer, D., and Hagersten, E. 2012. Phase guided profiling for fast cache modeling. In *CGO '12: The 2012 International Symposium on Code Generation and Optimization*. ACM, 175–185.

[Song et al. 2013] Song, C., Shi, J., Liu, R., Yang, J., and Chen, H. 2013. Parallelizing live migration of virtual machines. In *VEE '13 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 85–95.

[Song et al. 2014] Song, J., Liu, W., Yin, F., and Gao, C. 2014. Tsmc: A novel approach for live virtual machine migration. *Journal of Applied Mathematics*.

[Strunk 2012] Strunk, A. 2012. Costs of virtual machine live migration: A survey. In *SERVICES '12: IEEE Eighth World Congress on Services*. 323–329.

[Strunk and Dargie 2013] Strunk, A. and Dargie, W. 2013. Does live migration of virtual machines cost energy? In *AINA '13: The 27th IEEE International Conference on Advanced Information Networking and Application*. IEEE.

[Svärd et al. 2011] Svärd, P., Hudzia, B., Tordsson, J., and Elmroth, E. 2011. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *VEE '11: The 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, 111–120.

[Svärd et al. 2011] Svärd, P., Tordsson, J., Hudzia, B., and Elmroth, E. 2011. High performance live migration through dynamic page transfer reordering and compression. In *CloudCom '11: 3rd IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 542–548.

[Wood et al. 2011] Wood, T., Ramakrishnan, K. K., Shenoy, P., and van der Merwe, J. 2011. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. In *VEE '11: The 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, 121–132.

[Wood et al. 2007] Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. 2007. Black-box and gray-box strategies for virtual machine migration. In *NSDI '07: 4th USENIX Symposium on Networked Systems Design & Implementation*. 229–242.

[Xu et al. 2012] Xu, X., Yao, K., Wang, S., and Zhou, X. 2012. A vm migration and service network bandwidth analysis model in iaas. In *CECNet '12: 2nd International Conference on Consumer Electronics, Communications and Networks*. 123–125.

[Zhang et al. 2013] Zhang, Z., Xiao, L., Zhu, M., and Ruan, L. 2013. Mvmotion: a metadata based virtual machine migration in cloud. *Cluster Computing*, 1–12.

[Zheng et al. 2011] Zheng, J., Ng, T. S. E., and Sripanidkulchai, K. 2011. Workload-aware live storage migration for clouds. In *VEE '11: The 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, 133–144.