Urgent Virtual Machine Eviction with Enlightened Post-Copy

Yoshihisa Abe[†], Roxana Geambasu[‡], Kaustubh Joshi[•], Mahadev Satyanarayanan[†]
[†]Carnegie Mellon University, [‡]Columbia University, [•]AT&T Research
{yoshiabe, satya}@cs.cmu.edu, roxana@cs.columbia.edu, kaustubh@research.att.com

Abstract

Virtual machine (VM) migration demands distinct properties under resource oversubscription and workload surges. We present *enlightened post-copy*, a new mechanism for VMs under contention that evicts the target VM with fast execution transfer and short total duration. This design contrasts with common live migration, which uses the down time of the migrated VM as its primary metric; it instead focuses on recovering the aggregate performance of the VMs being affected. In enlightened post-copy, the guest OS identifies memory state that is expected to encompass the VM's working set. The hypervisor accordingly transfers its state, mitigating the performance impact on the migrated VM resulting from post-copy transfer. We show that our implementation, with modest instrumentation in guest Linux, resolves VM contention up to several times faster than live migration.

1. Introduction

As a means of load balancing, VM migration plays a crucial role in cloud resource efficiency. In particular, it affects the feasibility of oversubscription. Oversubscription is co-location of VMs on the same host in which their allocated resources can collectively exceed the host's capacity. While it allows the VMs to share the physical resources efficiently, at peak times they can interfere with each other and suffer performance degradation. Migration, in this situation, offers a way of dynamically re-allocating a new machine to these VMs. The faster the operation is in resolving the contention, the more aggressive vendors can be in deploying VMs. Without a good solution, on the other hand, those with performance emphasis must relinquish resource efficiency and use more static resource assignments, such as Placements on Amazon EC2 [1]. Although such strategies can guarantee VM performance, they lead to wasted resources due to conservative allocation. Migration for contending VMs thus has its own value, which can impact resource allocation policies.

However, migration of a VM under contention poses challenges to be solved because of its distinct requirements. It needs to salvage the performance of all the VMs being affected, namely their *aggregate performance*. The primary goal, therefore, is to evict the target VM from its host rapidly, allowing the other VMs to claim the resources it is currently consuming. This objective is achieved by transferring the execution of the migrated VM to a new host, so that its computational cycles are made available to the other VMs. Additionally, the duration of migration decides when the VM's state can be freed on the source host; reclaiming the space of memory state, which can be tens or hundreds of gigabytes, can be particularly important. While adhering to these priorities, the impact on the migrated VM should be mitigated to the extent possible, for its service disruption to be limited and the aggregate performance to recover fast.

These properties are especially desirable when the VMs provide services that need to sustain high overall throughput. Example cases include back-end servers or batch-processing applications, such as big data analytics. With these types of workloads, resolving the contention between VMs directly translates to optimizing their performance as a whole. Migration can also be more for saving the contending VMs than the target VM itself. The contending VMs can be running more latency-sensitive services, such as web servers, than those of the target VM. Or, the target VM may be malfunctioning, for example under a DoS attack, needing diagnosis in segregation. In all these situations, as illustrated in Figure 1, migration with appropriate characteristics would allow the VMs to be co-located under normal operation, and to be allocated new resources when experiencing a surge of loads.

Unfortunately, the requirements described above defy the trends in VM migration approaches. In particular, the current standard of live migration [12, 30] often results in elongated duration due to its design principles [19, 38]. This behavior leaves VMs under contention, delaying their performance recovery. In this paper, we present a design point that is fundamentally different and driven by *enlightenment* [28]. Enlightenment is a type of knowledge the guest passes to the hypervisor for improving the efficiency of its operation. Applying it to migration, we develop an approach called *en-*

VEE '16 April 2–3, 2016, Atlanta, Georgia, USA.

DOI: http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2892242.2892252

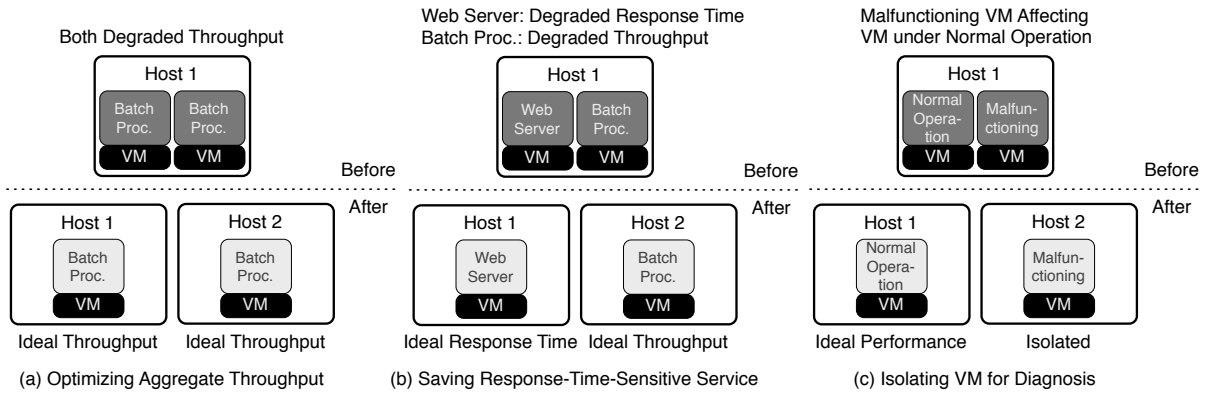

Figure 1. Scenarios for Urgent VM Eviction. The top and bottom rows illustrate VMs before and after migration, respectively.

lightened post-copy. It employs a post-copy style, in which the VM is resumed on the destination before its state has been migrated, and mitigates the resulting performance impact through enlightenment. Upon migration, the guest OS informs the hypervisor of memory regions that require high transfer priority for sustaining the guest performance. The hypervisor then transfers the execution of the VM immediately to a new host, while continuously pushing its state as instructed and also serving demand-fetch requests by the destination VM. Our implementation, with modest changes to guest Linux, shows the effectiveness of the guest's initiative in performing migration with the desired properties.

The main contributions of this paper are as follows:

- Explaining common behaviors of migration and deriving properties desired for VMs under contention (Section 2).
- Presenting the design and implementation of enlightened post-copy, an approach that exploits native guest OS support (Sections 3 and 4).
- Evaluating the performance and trade-offs of enlightened post-copy against live migration and other basic approaches (Section 5).

2. Analysis of VM Migration

VM migration has historically focused on the liveness, namely minimal suspension, of the target VM. Specifically, live migration is the current standard widely adopted by common hypervisors [12, 30]. The characteristics of live migration, however, deviate from those desired in the context of VMs under contention; it leads to extended duration of migration and thus continued interference of the VMs. Figure 2 illustrates this problem, with live migration by qemu-kvm 2.3.0 under the Memcached workload and experimental setup described in Section 5. One of the two contending VMs is migrated, with approximately 24 GB of memory state to be transferred, under varied bandwidth. Migration takes 39.9 seconds at 10 Gbps, and fails to complete at 2.5 Gbps. For the duration of migration, the VMs suffer degraded performance due to their mutual interference. In this section, we describe the algorithm and behavior of live migration causing this problem, and derive the properties desired in our solution.

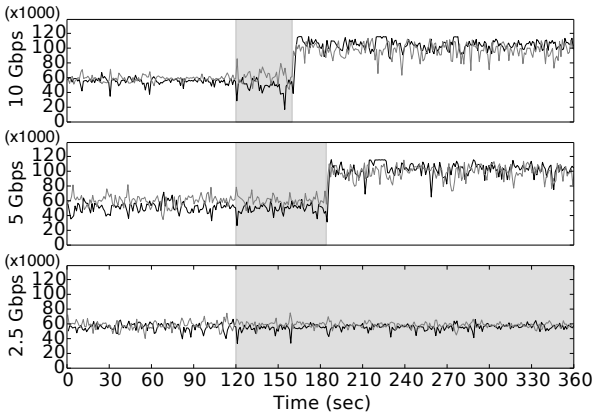

Figure 2. Live Migration Behavior of qemu-kvm under VM Contention. The y-axis indicates throughput in operations per second. The black and gray lines represent the migrated and contending VMs, respectively, and the shaded areas the duration of migration.

2.1 Mechanics of Migration

Figure 3 illustrates aspects of VM migration including execution, state transfer, and performance. Migration is initiated on the source, on which the VM originally executes. The VM is suspended at one point, and then resumed on the destination. State transfer during the course of migration is categorized into two types: pre-copy and post-copy. Pre-copy and post-copy are phases performed before and after the VM resumes on the destination, respectively. Associated with the duration of these state transfer modes, there are three key time metrics:

- **Down time**: time between the suspension and resume of the VM, during which its execution is stopped.

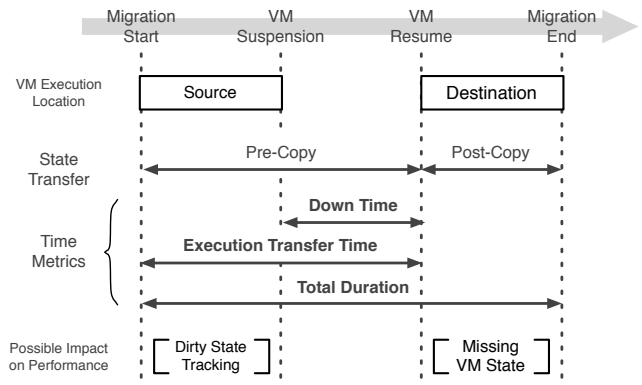

Figure 3. Overview of Migration Mechanics

- **Execution transfer time**: time since the start of migration until the VM resumes on the destination.
- **Total duration**: time since the start and until the end of migration.

Until execution transfer completes, contending VMs on the source continue to experience degraded performance. Thus, the faster execution transfer is, the more effective the migration operation is in salvaging the performance of both the migrated and other VMs. Reasonable down time also is important for mitigating the service disruption of the migrated VM. The VM execution halts during this time, rather than continuing with performance degradation. Finally, the hypervisor on the source needs to maintain the migrated VM's state until the end of total duration. Shorter total duration, therefore, means that allocated resources such as guest memory can be freed and made available to other VMs sooner.

Pre-copy and post-copy phases have associated performance costs. Pre-copy, when overlapped with VM execution, requires tracking state changes to synchronize the destination hypervisor with the latest VM state, a computational overhead known as migration noise [20]. Post-copy, on the other hand, can stall VM execution when the running guest accesses memory contents that have not arrived on the destination.

2.2 Live Migration

Live migration commonly employs pre-copy, and its algorithm works as shown in Figure 4. Upon initiation, live migration starts sending memory page contents while continuing the VM execution and keeping track of memory content changes. It then iteratively retransmits the pages whose content has been dirtied since its last transfer. The purpose of the iteration phase is to minimize down time, thereby optimizing for the liveness of the migrated VM. While iterating, the algorithm uses the current rate of state transfer to estimate down time, during which the last round of retransmission is performed. If the expected down time is short enough (e.g., 300 ms in qemu-kvm), the iteration phase completes and the VM is resumed on the destination. Implementations

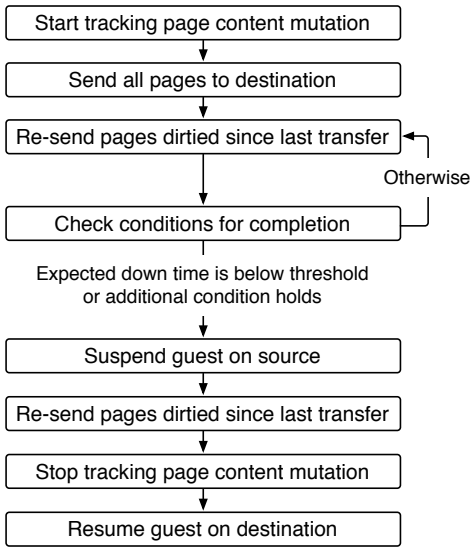

Figure 4. Live Migration Algorithm

can also have additional conditions for preventing migration from taking an excessive amount of time. Common examples include a limit on the number of iterations, and high expected down time that steadily exceeds a threshold. Regardless of the exact form in which these conditions are expressed, common to these parameters of live migration is that they aim to control the maximum duration of the iteration phase. Note that Figure 4 illustrates the migration of memory state; in this paper, we assume the availability of disk state through shared storage.

2.2.1 Impact of Workloads

Being pre-copy and optimized for down time, live migration handles state transfer while dealing with the guest state changes. Consequently, its behavior depends on the guest workload and the bandwidth available for migration traffic. Figure 5 shows the throughput of a Memcached server, hosted in a VM, during live migration by qemu-kvm. The memslap benchmark generates a load for the server, and its set-get ratio and the bandwidth available for migration traffic are varied. The other configurations for these measurements are the same as those described in Section 5, with the guest allocated 30 GB of memory and the server using 24 GB as its cache. Note that qemu-kvm zeroes out guest memory when setting it up, and live migration compresses each page whose bits are all zeros to one byte accompanied by a header; thus, it avoids sending the unused 6 GB in these measurements.

As the available bandwidth for migration traffic decreases, live migration takes more time to complete. This increase is non-linear; with set-get ratio of 1:9, migration finishes in approximately 40 and 90 seconds at 10 and 5 Gbps, respectively. At 2.5 Gbps, it fails to complete in a timely manner. With set-get ratio of 5:5, migration does not complete even at 5 Gbps. This is because expected down time never becomes short enough with the guest workload, and

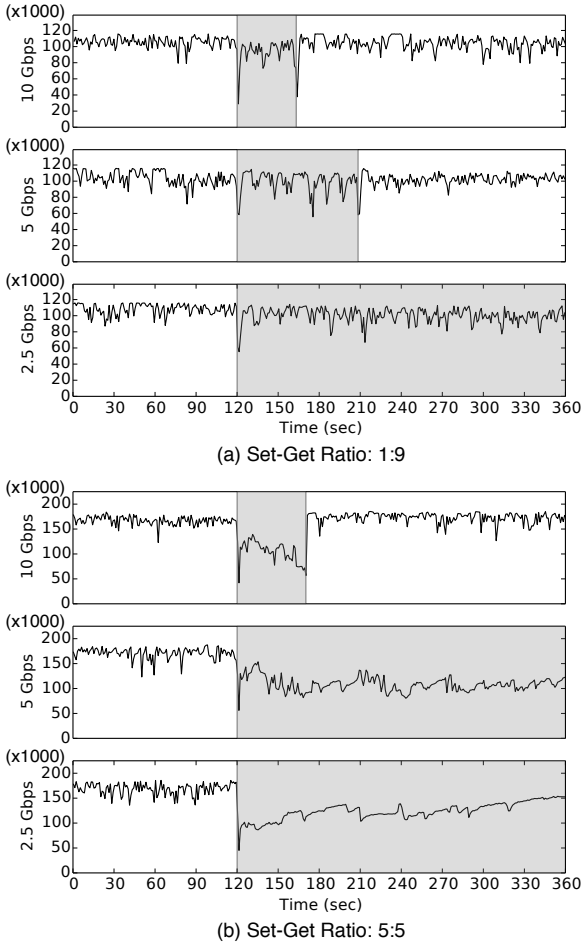

Figure 5. Behavior of Migrating a Memcached VM with qemu-kvm. The y-axis indicates throughput in operations per second, and the shaded areas the duration of migration.

qemu-kvm does not use a hard limit on the number of iterations. In addition, we can observe more throughput degradation during migration with set-get ratio of 5:5 than with 1:9. As the workload generates more memory content changes, dirty state tracking interferes more with it because of trapping memory writes, which are caught more frequently. Finally, even when live migration performs fairly well with set-get ratio of 1:9 and at 10 Gbps, it takes considerably longer than transferring 24 GB over that bandwidth (which takes less than 20 seconds). qemu-kvm's migration code is single-threaded, and it saturates a CPU core to transmit state at gigabytes speed while tracking dirty pages. Migration can thus easily be a CPU-bound procedure unless special care is taken, for example by parallelization of the code [39].

2.2.2 Commonality among Implementations

The characteristics of live migration explained above are inherent to its algorithm, and therefore shared by major implementations. Figure 6 shows the behavior of migrating a Memcached VM with common hypervisors, qemu-

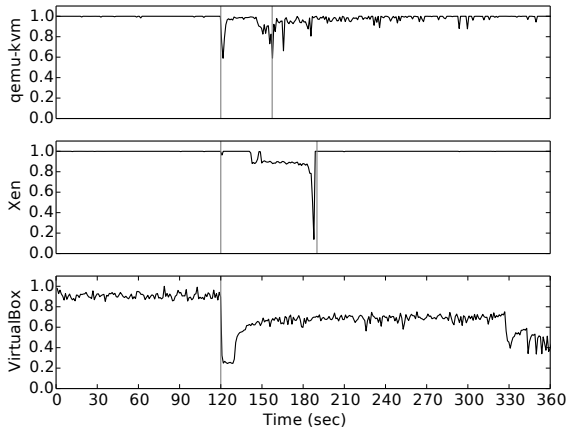

Figure 6. Behavior of Migrating a Memcached VM with Major Hypervisors at 10 Gbps. The y-axis indicates throughput normalized against the maximum in each measurement, and the shaded areas represent the duration of migration.

kvm 2.3.0, Xen 4.1.6, and VirtualBox 4.3.16[1]. The memslap benchmark is run with set-get ratio of 1:9. The VM memory size and server cache size are the same as explained previously: 30 GB and 24 GB. The VM is assigned 4 cores, and migrated over a 10 Gbps link. Note that we used machines different from those for the rest of our measurements, due to hardware accessibility reasons. They were equipped with an Intel Core i7-3770 CPU at 3.4 GHz and 32 GB of memory, running Ubuntu 12.04 with Linux kernel version 3.5.0.

As each implementation differs from one another, the performance cannot be directly compared between the hypervisors. In particular, the duration of the iteration phase is determined by parameters, and additional factors such as page content compression also lead to varying performance. For example, Xen takes longer than qemu-kvm, with its nature of throughput degradation during migration differing from that of qemu-kvm. Also, unlike the other hypervisors, VirtualBox does not complete migration under this workload. The key point of these results, however, is not the absolute performance differences, but the common behavior that the VM lingers on the source for tens of seconds or longer. This total duration exemplifies the cost paid in an effort to minimize down time.

2.3 Desired Properties of Migration under Contention

Live migration thus exhibits undesirable behavior when migrating contending VMs, for two fundamental reasons. First, it focuses on the down time of the target VM, rather than considering all the VMs affected. Second, it uses pre-copy and monitors the VM workload to achieve its objective, delaying execution transfer. Our goals, in contrast, are to 1) free resources on the source rapidly through fast execution transfer and short total duration, 2) handle loaded VMs

[1] Similar results with VMware ESXi 5 are publicly available in [7].

without relying on the reduction in their workloads, and 3) with these properties, salvage the aggregate performance of the VMs under contention. These requirements motivate the guest's active cooperation, which allows the departure from pre-copy and workload monitoring.

3. Enlightened Post-Copy Migration

Our approach to the above goals, called enlightened post-copy, exploits guest cooperation and post-copy-style state transfer. We derive the key ideas behind this approach specifically from our goals. First, minimizing execution transfer time requires that VM execution be immediately suspended on the source and resumed on the destination. This early suspension upon the start of migration also ensures minimal total duration, because the frozen VM state necessitates no retransmission as done in live migration. Therefore, post-copy follows naturally as the desirable method of state transfer. Second, fast performance recovery of the migrated VM requires that the part of its state needed for its current workload arrive at the destination as early as possible. The guest's enlightenment is the key that enables identifying this part of the VM state; with state transfer following the instructed prioritization, the VM on the destination can start recovering its performance without the completion of entire state transfer, and thus before the total duration of migration.

Figure 7 illustrates the workflow of enlightened post-copy. When migration is initiated, the hypervisor makes a request for enlightenment to the guest OS. The guest OS traverses its data structures and prepares priority information of the memory pages. Once the priority information is available, the guest OS notifies the hypervisor. The hypervisor then suspends the VM on the source, and resumes it on the destination immediately after sending the device state necessary for the resume operation. As the VM starts execution, the hypervisor parses the priority information and accordingly transfers the remaining memory page contents to the destination; it attempts to proactively push as many pages as possible before the access to them.

3.1 Guest's Enlightenment

In enlightened post-copy, the guest identifies those memory pages containing the working set of the currently active processes. As a design principle, the guest OS should be able to obtain a list of these pages without incurring noticeable overhead. Otherwise, the approach does not justify the guest instrumentation due to the resulting performance loss. As the types of memory page classification, therefore, we use straightforward notions such as the code and data of the OS kernel and running processes. Such bookkeeping information of memory pages is already available in the OS for its regular tasks, and re-usable without significant implementation effort for the purpose of migration.

For prioritized state transfer, the general idea is to transfer memory pages essential for running the guest OS, those for the actively running processes, and then the other less critical pages such as the kernel page cache and those for the non-active processes. Also, we can eliminate the transfer of the memory pages that are not allocated by the guest OS for any use, because the actual contents of such pages do not affect the correctness of guest execution [23].

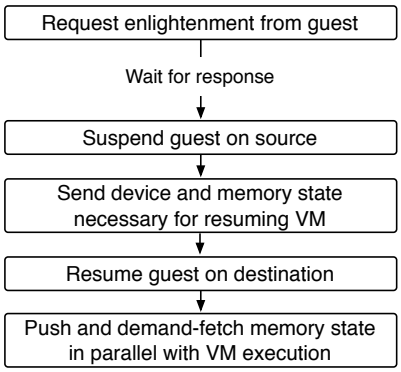

Figure 7. Enlightened Post-Copy Migration Algorithm

The guest OS needs to prepare these types of information, as enlightenment to the hypervisor, in two distinct forms. The memory page priorities can be determined by a one-time operation upon the request by the hypervisor. There is no need for always tracking them during the course of the guest's normal operation. On the other hand, keeping track of allocated and non-allocated memory pages requires real-time processing, performed with or without migration, that maintains the information in a manner easily passed to the hypervisor. The reason is that the source hypervisor needs to know the exact allocation by the guest OS right at the time of VM suspension, for the destination hypervisor to construct a consistent memory image. For the one-time operation, the associated costs are that of guest-host communication delay upon migration start, and the impact of the in-guest processing on performance. For the real-time processing, the cost is the overhead added to relevant memory management operations of the guest OS. Minimizing these costs motivates the use of the above types of enlightenment, which are adequately informative but not excessively fine-grained.

3.2 Integration into State Transfer

The source hypervisor can integrate enlightenment into state transfer in a straightforward manner, because of the use of post-copy. Since the VM is frozen at the start of migration, enlightenment at that time reflects its latest state before execution transfer, from which the VM resumes on the destination. After receiving enlightenment and suspending the VM, the source hypervisor pushes the memory pages as instructed by the guest OS. While the destination hypervisor receives the memory pages, it also issues demand-fetch requests to the source for those that are accessed by the guest before their arrival. Although their arrival may incur delays due to the interference with the push traffic, these demand fetches

help reduce VM execution stalls due to the divergence of the push order from the actual access order by the guest.

3.3 Design Trade-Offs

The design of enlightened post-copy is in sharp contrast to that of common live migration based on pre-copy. Enlightened post-copy targets VMs under load, while live migration expects idleness from them. Enlightened post-copy, therefore, employs a state transfer method that enables timely load balancing through fast physical resource reallocation. Down time and execution transfer time are expected to be instantaneous, and total duration corresponds to the one-time transfer of the entire state. At the same time, the disruption of the migrated VM's performance spans a longer period than down time itself, since post-copy is used. Guest cooperation is the key to alleviating this disruption.

On the other hand, live migration focuses on one aspect of the migrated VM's performance, down time. Being a guest-agnostic approach without an external source of knowledge, it relies on letting the VM stay on the source and tracking dirtied memory pages. Execution transfer time is equivalent to total duration; these time frames become longer when more iterations are done. The sole use of pre-copy ensures the migrated VM's performance on the destination, since all the state resides there on VM resume. Thus, down time approximately represents the duration of application-level disruption. However, dirty page tracking incurs a certain cost while the VM lingers on the source. Results in Section 5 demonstrate the effects of these trade-offs made by enlightened post-copy and live migration.

4. Implementation

We implemented enlightened post-copy on guest Linux 3.2.0 and hypervisor qemu-kvm 2.3.0. Figure 8 shows its architecture. When the source hypervisor initiates migration, it sends a request to guest Linux through a custom VirtIO device [6] (Step 1). The guest OS driver for this virtual device triggers enlightenment preparation, which scans data structures (Step 2) and writes priority information in the priority bitmap (Step 3). Page allocation information is always kept up-to-date in the free bitmap, so that its content is valid whenever the the hypervisor suspends the VM. These bitmaps maintained in the guest's memory facilitate the processing by the hypervisor; they are an abstract enough representation of the passed information, and the guest OS can avoid communicating it through the VirtIO device and instead have the hypervisor directly parse it. When the priority bitmap has been written, the guest OS notifies the hypervisor through the VirtIO device (Step 4). The hypervisor then sends to the destination the device state, including some in the guest memory, which is used by the destination hypervisor for the initial VM set-up. Finally, it starts transferring the remaining page contents in the prioritized order (Steps 5 and 6). On the destination, the hypervisor resumes the VM once the device state has arrived. While receiving the pushed page contents, it writes them into the guest memory. When the guest OS accesses pages whose content has not yet been received, it generates demand-fetch requests to the source hypervisor. On the source, the hypervisor frees all the VM resources once all the page contents have been sent.

Kernel	Kernel executable code
Kernel_Allocated	Allocated for kernel use
Memory_I/O	Used for memory-mapped I/O
Page_Table	Page table of active process
User_Code	Executable page of active process
User_Data	Non-executable page of active process
File_Active	Active cache of file
File_Inactive	Inactive cache of file
Other	Not belonging to any of the above

Table 1. Page Types Categorized by the Guest OS

4.1 Guest OS

In our guest Linux, memory management and process scheduling code is instrumented to label each memory page with a priority level. The instrumentation follows naturally in the relevant existing parts of the source code, and requires only a modest number of changes to the original kernel.

4.1.1 Enlightenment Preparation

Taking advantage of memory management information that already exists, the guest OS classifies the memory pages in use into the priority categories shown in Table 1. The system-wide categories, such as Kernel, Kernel_Allocated, Memory_I/O, File_Active, and File_Inactive, are derived from kernel data structures or through the flags of page frame descriptors (e.g., `struct zone` and `struct page`). For the process-specific categories, the enlightenment preparation code parses the virtual memory area descriptors of each active process (`struct vm_area_struct`). These categories are each assigned a numerical value, in a descending order of priority from the top to the bottom in the above list. This order is decided such that the core system services, the active processes, and caching by the OS are given priority in that order. If a particular page belongs to multiple categories, it is treated with the highest of these priorities. Pages such as those belonging to the inactive processes and those used for the priority bitmap itself belong to the Other category. The bitmap simply contains the priority values, without the hypervisor needing to understand their exact semantics.

In order to decide the group of active processes, the scheduler maintains the last time each process was scheduled for execution (in `struct task_struct`). A process is considered active if it has been run recently at the time of generating enlightenment. In our implementation, we empirically use a threshold of the past 16 seconds for this purpose, considering the order of seconds migration is roughly expected to take.

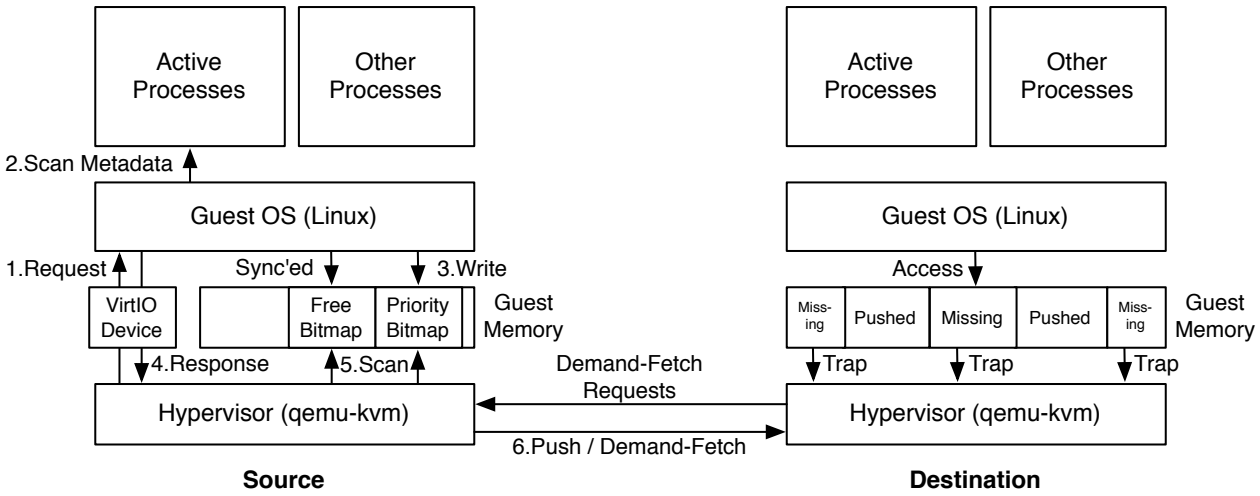

Figure 8. Implementation of Enlightened Post-Copy. The numbers represent the steps in order during the course of migration.

When the guest OS receives a request from the hypervisor, the guest OS runs code that generates the priority bitmap as explained above. Once it notifies the hypervisor, it is ensured that the guest can be suspended with the bitmaps available for parsing in its memory. Note that the free bitmap is always kept up-to-date and ready for processing. The response from the guest OS includes the addresses of the two bitmaps, and the hypervisor scans these locations while performing state transfer.

4.1.2 Implementation Cost

The modifications to the core Linux kernel code are minimal, mostly under `mm/` and `kernel/` in the source tree. Maintaining the free bitmap requires adding at most several lines of code in 16 locations. Keeping track of the last schedule time for each process requires a few variable assignments added in the scheduler code. Marking each page with kernel or user allocation needs less than 15 lines of code. These operations only add atomic variable assignments in the code paths of memory allocation and process scheduling. As shown in our experiments, compared to the original kernel, these minor modifications incur negligible performance overhead. The VirtIO device driver and the priority bitmap preparation code, which make use of the instrumentation, are implemented as loadable kernel modules.

4.2 Hypervisor

On the source, the hypervisor scans the bitmaps provided by the guest. It first scans the free bitmap and sends the corresponding page frame numbers in a packed format, along with the device and some memory state, right after which the VM is resumed on the destination. Next, the hypervisor traverses the priority bitmap and starts pushing the page contents over a TCP connection. The transfer is performed in rounds, starting with the Kernel pages and ending with the Other pages. While this push continues, a separate thread services demand-fetch requests from the destination hypervisor over another TCP connection. Note that while we attempt to give a higher priority to the demand-fetched pages through the TCP_NODELAY socket option, the push transfer can still interfere with their arrival timings.

On the destination, the hypervisor registers userfault [5] handlers with the guest memory region. Userfault is a mechanism on Linux that enables a user process to provide a page fault handler of its own for specified pages. This interposition enables post-copy state transfer. The userfault handler is first registered for the entire main memory of the VM when the hypervisor starts on the destination. On the receipt of the unallocated page information, the handler is removed from the respective addresses. Then, as the memory page contents arrive, they are written to the corresponding addresses and the userfault handler is unregistered from these addresses. When the guest accesses these memory pages whose content is already available, no further interposition by the hypervisor is carried out. On access to a page whose content is still missing, the hypervisor issues a demand-fetch request to the source hypervisor.

5. Experiments

We performed experiments to demonstrate the effectiveness of enlightened post-copy in resolving resource contention and the performance trade-offs resulting from its design principles. Our experiments address the following points. First, we show how enlightened post-copy salvages the throughput of contending VMs through end-to-end results, while comparing them to those of original live migration of qemu-kvm 2.3.0. Next, we investigate the efficacy of our approach in robustly dealing with workloads by examining the performance of two baseline methods, stop-and-copy and simple post-copy, as reference points. Finally, we show the cost of enlightened post-copy in terms of state transfer amounts and guest OS overhead.

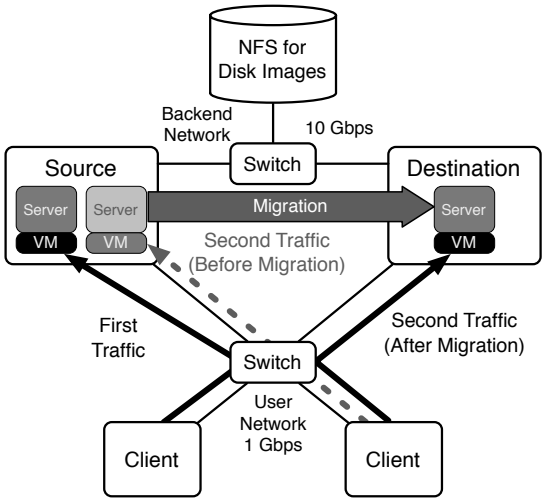

Figure 9. Experimental Set-Up

5.1 Set-Up and Workloads

Figure 9 shows our experimental set-up. VMs are migrated between a pair of source and destination hosts, which are connected through a backend 10 Gbps network for migration traffic. They are also connected on this network to an NFS server that stores VM disk images. The client machines use a separate 1 Gbps network for user traffic to and from the VMs. In the experiments except those with idle VMs, initially two VMs are running and contending with each other on the source machine. We migrate one of the VMs to the destination machine, after which each VM has its own dedicated machine. The VM hosts and client machines are each equipped with two Intel Xeon E5-2430 CPUs at 2.20 GHz and 96 GB memory, running Ubuntu 14.04. The VM hosts execute Linux kernel 4.1.0-rc3 with a userfault patch applied, while the other machines run version 3.16.0. As userfault currently does not support the use of huge pages, they are disabled in our measurements. Time on the machines is synchronized via an NTP server, and the backend bandwidth between the VM hosts is controlled using Linux Traffic Control for measurements at 5 Gbps and 2.5 Gbps. The VMs run Ubuntu Server 12.04 with unmodified kernel 3.2.0 in all the cases except those with enlightened post-copy, in which we use our modified version of the kernel.

We use the following workloads that exhibit different types of resource intensity and reveal performance trade-offs made by enlightened post-copy:

Memcached: The VMs run an in-memory key-value store, Memcached 1.4.13 [2], and the clients execute its bundled benchmark memslap 1.0, which is modified to report percentile latencies. The VMs are each allocated 30 GB of memory and 8 cores, with Memcached configured with 4 threads (due to its known scalability limitation) and 24 GB cache. We first run the benchmark against Memcached to fill up its cache, and then perform measurements with concurrency level of 96 and set-get ratio of 1:9. At the time of migration, approximately 24 GB of memory is in use, almost all of which is by Memcached.

MySQL: The VMs run MySQL 5.6, and the clients execute OLTPBenchmark [3] using the Twitter workload with scale factor of 960. The VMs are each allocated 16 cores and 30 GB of memory, and MySQL is configured with a 16 GB buffer pool in memory. The concurrency of OLTPBenchmark is set to 64. After generating the database contents, we execute Tweet insertions for 25 minutes and then the default operation mix for 5 minutes as a warm-up. At the time of migration, MySQL uses approximately 17 GB of memory, and almost all of the 30 GB memory is allocated by the guest OS for use.

Cassandra: The VMs run a NoSQL database, Apache Cassandra 2.1.3 [4], and the clients use YCSB [8] 0.1.4 with 24 threads and core benchmark F, which consists of 50% read and 50% read-modify-write operations. The VMs are each configured with 16 cores and 30 GB of memory. Before measurements, the benchmark is run for approximately 10 minutes to warm the servers up. At the time of migration, the server uses around 8.4 GB of memory out of 12 GB in use by the guest OS.

In the above workload configurations, Memcached is the most memory- and network-intensive, while consuming relatively low CPU resources. Also, the VM's memory is almost exclusively used by the server process itself. MySQL is more CPU-intensive, and also less memory-intensive in terms of the access footprint per unit time. Finally, Cassandra is the most compute-intensive among these workloads, making CPUs the source of contention. In the MySQL and Cassandra cases, the guest OS uses a non-trivial amount of memory in addition to that allocated by the server processes themselves. These characteristics make Memcached the hardest case, and MySQL and Cassandra more winning cases for enlightened post-copy in comparison to live migration.

5.2 End-to-End Performance

In this section, we compare application-level performance of the three workloads during migration with enlightened-copy and live migration. In addition to the throughput of the server applications, we also report the impact of migration on application-level latency.

5.2.1 Memcached

Figure 10 (1) compares Memcached throughput of enlightened post-copy (labeled "EPC") and live migration (labeled "Live") under varied bandwidth. The y-axis shows operations per second in thousands (x1000), and the total duration of migration is shown as shaded areas. The dark lines indicate the performance of the migrated VM, and gray lines are that of the contending VM. The dotted lines represent the aggregate of the the two. The source of contention is user traffic handling by the source hypervisor. As Memcached accounts for almost all the guest memory pages in use (which

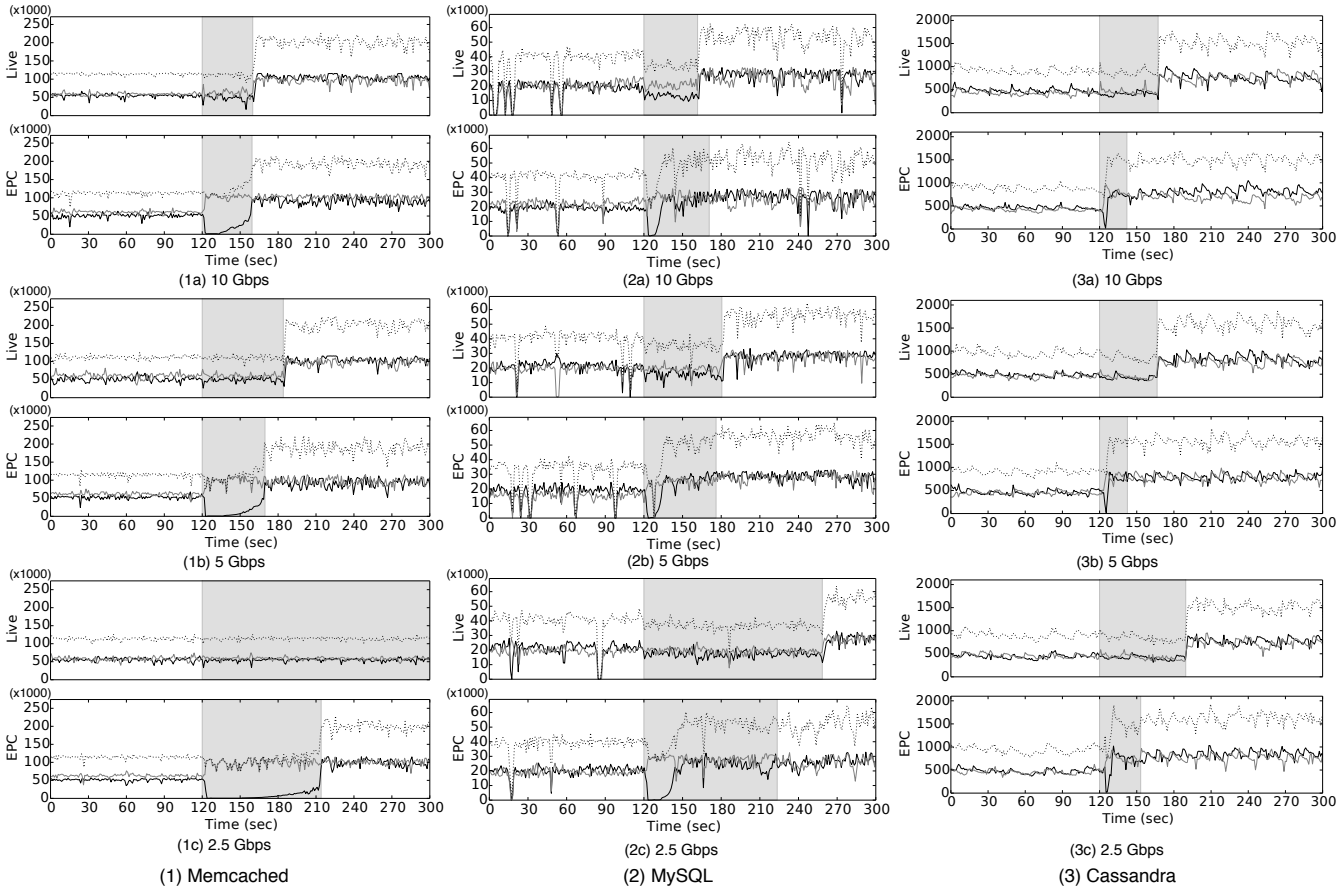

Figure 10. End-to-End Results with Enlightened Post-Copy and Live Migration. The y-axis indicates throughput in operations or transactions per second. The black and gray lines represent the migrated and contending VMs, respectively, and the dotted lines the aggregate of the two. The shaded areas show the duration of migration.

are categorized into User_Data) and accesses them at a great speed, it leaves little room for memory prioritization through enlightenment. Thus, the performance recovery during enlightened post-copy migration is not significant because it requires most pages for the Memcached server to be present. Immediate execution transfer, still, lets the contending VM recover its performance as soon as the migration starts. On the other hand, live migration handles the read-mostly workload relatively well. At 10 Gbps, it finishes almost as soon as enlightened post-copy does. However, it performs more state retransmission as bandwidth becomes lower, and at 2.5 Gbps it fails to finish while the benchmark continues. Note that, because of the migration thread causing the saturation of a core at 10 Gbps, the results at this speed are not as good as can be expected from the 5 Gbps results.

The latency characteristics of the 10 Gbps measurements are shown in Figure 11 (1). The top two graphs present the 90th percentile latency of the server responses over time. The latency stays roughly between 1 and 2 ms before migration, and around 1 ms once it completes. Live migration sustains mostly good latency until the end of migration. Enlightened post-copy leaves the 90th percentile latency close to 1 second right after the start of migration, while it starts to decrease as more state arrives at the destination. The bottom two graphs show CDFs of the response times during the same 5-minute period. Despite its longer tail to the right side, enlightened post-copy still maintains the curve of the migrated VM close to that of the contending VM and those with live migration. While the differences in throughput should also be considered when interpreting these results, they indicate that the impact on latencies of the served requests is moderate at the 5-minute granularity.

5.2.2 MySQL

Figure 10 (2) shows the throughput results with MySQL. Although the workload is less network-intensive than Memcached, multiplexing user traffic between the VMs on the source causes contention. We also attribute to this bottleneck the ephemeral performance drops that are observed especially before migration. As the workload has more memory access locality, as well as memory allocated besides the cache of MySQL itself, enlightened post-copy gains signif-

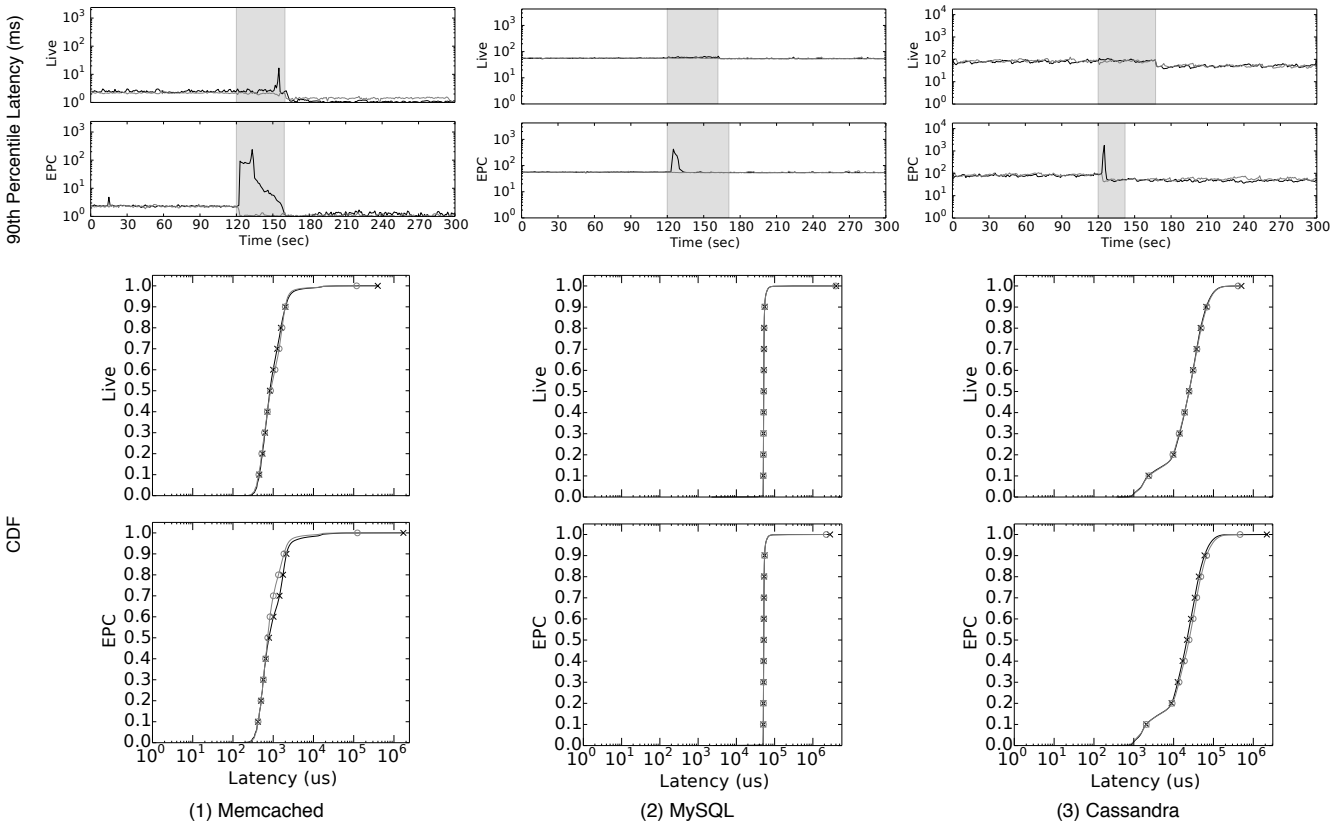

Figure 11. Latency with Enlightened Post-Copy and Live Migration at 10 Gbps. The black and gray lines correspond to the migrated and contending VMs, respectively. In each column, the top two graphs show the 90th percentile latency over time on a log scale, with the shaded areas indicating the duration of migration. The bottom two figures are CDFs of the response latencies during the 5-minute period, with markers at every 10th percentile.

icantly from prioritized state transfer. The throughput of the migrated VM starts recovering shortly after the migration start, and well before its total duration. In addition to taking longer as the workload size increases with respect to bandwidth (i.e., at lower bandwidth), live migration also exhibits more interference with the throughput of the migrated VM at higher bandwidth. The reason is that the workload is fairly CPU-intensive, and that the migration thread performs dirty state checking more frequently per unit time. Also, unlike all the other cases, live migration completes sooner than enlightened post-copy at 10 Gbps. As the interference of the hypervisor slows the guest, it consumes less computational resources besides those spent for network transfer than enlightened post-copy does. As a result, live migration can better utilize the bandwidth.

The latency results for the workload are shown in Figure 11 (2). The 90th percentile latency with enlightened post-copy recovers quickly as the throughput does, lowering to the level of a VM without contention before the completion of migration. The CDFs also indicate that the response time distributions are comparable between the two methods, including the tails to the right representing the maximum latency.

5.2.3 Cassandra

Finally, Figure 10 (3) shows the results with Cassandra. This workload makes the CPUs on the source the bottleneck for the VMs before migration. Its total duration not being affected by the resource intensity of the workload, enlightened post-copy finishes as soon as the amount of memory in use has been transferred. It also starts recovering the throughput of the migrated VM halfway through the migration process. With severe CPU contention on the source, live migration is prevented from performing dirty state checking and state transmission frequently. Thus, we do not observe its interference with the migrated VM's throughput, but instead see total duration heavily penalized at higher bandwidth; effective state transfer rate stays low enough that the duration differs only slightly between 10 and 5 Gbps. Overall, the difference in the duration between the two methods is more significant than with the other workloads.

As shown in Figure 11 (3), the good performance of enlightened post-copy is also reflected in the latency results. The 90th percentile latency increases for a short period with enlightened post-copy, and soon drops to the ideal level without the contention. Also, the response time distributions

of enlightened post-copy and live migration compare well to each other. Except for the right tail of the migrated VM being a little longer with enlightened post-copy, the two methods show similar distribution curves.

5.3 Comparison with Baseline Approaches

We have so far compared enlightened post-copy with live migration based on pre-copy, which is predominantly used in today's virtualized environments. We further describe our design trade-offs by way of comparison to two fundamental approaches: stop-and-copy [36, 43] and simple post-copy. Stop-and-copy is an early form of migration that stops the VM, transfers all its state, and resumes the VM, in a sequential manner. It achieves the shortest total duration possible at the cost of making down time equivalently long. Simple post-copy solely uses demand fetches. It transfers only those memory pages that are being accessed by the guest on the destination, making each access incur an RTT between the hosts. These approaches can be considered as extreme design points: stop-and-copy as live migration that eliminates iterations for the sake of minimal duration, and simple post-copy as enlightened post-copy without, or with completely ineffective, enlightenment. They thus serve as baselines that reveal the benefits of using the sophistication in enlightened post-copy.

Figure 12 illustrates the behavior of stop-and-copy and simple post-copy with the Memcached and MySQL workloads at 10 Gbps. The two workloads exemplify cases in which they perform well or poorly compared to enlightened post-copy (whose corresponding cases are shown in Figure 10 1a and 2a). Stop-and-copy works relatively well for Memcached, and poorly for MySQL. Its performance is determined by the state transfer amount, regardless of the workload, while enlightened post-copy copes better with the MySQL workload than with the Memcached workload. The gain by enlightened post-copy, therefore, becomes clearer in the MySQL case. Simple post-copy is ineffective for Memcached and fairly adequate for MySQL. It significantly impacts the Memcached performance once the VM starts on the destination, as the cost of page retrieval is prohibitive for the memory-intensive workload. MySQL, on the other hand, exhibits enough memory access locality to prevent this cost from significantly affecting its performance. As a result, enlightened post-copy shows a clear advantage in the Memcached case. In summary, stop-and-copy and simple post-copy have cases they can handle and those they cannot; enlightened post-copy performs comparably to them in their best cases, and outperforms them considerably in the other cases.

5.4 Costs of Enlightenment

Enlightened post-copy targets VMs under load and makes explicit design trade-offs. One question that arises is the overhead incurred due to its design when used in other situations. Table 2 shows time and state transfer statics of migrating an idle VM with 30 GB memory over 10 Gbps. The guest OS uses approximately 1 GB of memory, with no user applications actively running. In part (a), the columns from left to right indicate guest communication time for obtaining enlightenment, time until the VM is suspended on the source, execution transfer time, and total duration. Although enlightened post-copy pays the price of communicating with the guest OS, the cost is insignificant in this idle VM case. Live migration, even when the VM is idle, needs to scan the entire guest memory and thus takes some time until completion. Overall, enlightened post-copy is no worse than live migration in terms of the time metrics. Part (b) in the figure shows the amount of state transfer by the transfer method used. "Free memory information" for enlightened post-copy represents how much data is sent to inform the destination hypervisor of all the unallocated memory pages. Since enlightened post-copy performs one-time transfer and live migration needs little retransmission, they transfer similar amounts in total.

(Unit: ms)	Enlightenment	Suspension	Execution Transfer	Total Duration
Live	-	6327	6548	6548
EPC	691	706	1280	2571

(a) Time Metrics

(Unit: KB)	Pre-Copy	Demand Fetch	Push	Free Memory Information	Total
Live	1029496 (100%)	-	-	-	1029496
EPC	36028 (4.0%)	892 (0.1%)	871188 (95.9%)	10 (<<0.1%)	908118

(b) State Transfer Amounts

Table 2. Costs of Idle VM Migration. The tables show time and state transfer statistics of migrating an idle VM, with no active applications inside. The numbers in parentheses in part (b) represent percentages of the total transfer amount.

In order to measure the real-time cost of tracking page allocation in the free bitmap, we ran a microbenchmark program inside the guest. The program performs repeated memory allocation, as fast as possible, in chunks of 1000 individual 4KB malloc() and free() calls each. With the original Linux kernel and our modified kernel, one pair of these calls takes 1.394 us and 1.455 us (4.4% increase), respectively. As demonstrated in the preceding results, this difference typically has negligible impacts on regular applications because they do not allocate and free memory as frequently.

6. Related Work

VM migration has improved over the past decade, with previous work targeting different environments. Live migration focuses on minimizing the down time of migrated VMs, and exemplifies pre-copy approaches. Early work on VM transfer started with stop-and-copy, in which guest execution is suspended before, and resumed after, entire state transfer. It was used by Internet Suspend/Resume [22, 37], and adopted by μDenali [43]. Stop-and-copy was also augmented with partial demand fetch and other optimizations [36] for virtu-

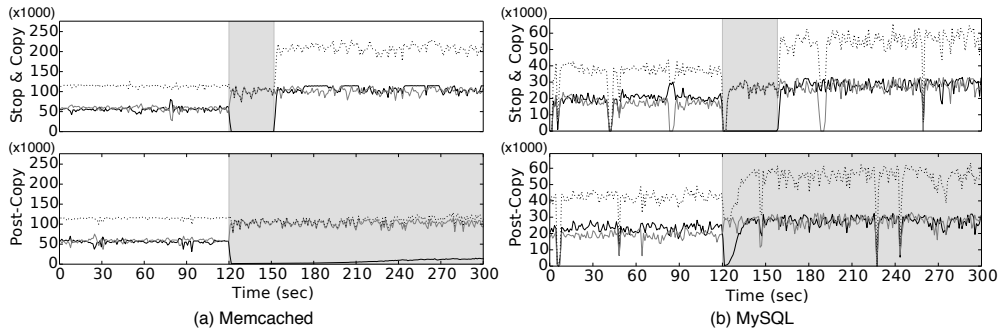

Figure 12. Behavior of Baseline Approaches. The y-axis indicates throughput in operations or transactions per second. The black and gray lines represent the migrated and contending VMs, respectively, and the dotted lines the aggregate of the two. The shaded areas show the duration of migration.

alizing the user's desktop environment, as *virtual desktops*, in Internet Suspend/Resume and Collective [11, 21, 34, 35]. The approach opposite to the pre-copy style is post-copy migration. VMs are resumed on their destination first, and then their state is retrieved. Examples of this approach include work by Hines et al. [16, 17] and by Liu et al. [24]. Post-copy migration is often desirable when migrating all state as fast as possible is prohibitive with respect to available network resources. Also in such situations, optimized techniques have proven to be effective that are based on a pre-copy approach [9, 44] and large-scale solutions such as VM distribution networks [31–33]. Hybrid approaches utilizing pre-copy and post-copy have also been proposed [26]. Finally, various optimizations to migration have been used, such as page compression [15, 40] and guest throttling [9, 27]. Page compression can considerably reduce the amount of state transfer, being effective when the migration process is not bottlenecked by CPUs. Guest throttling helps live migration to complete early by limiting the rate of page dirtying. In this work, on the other hand, we aim to let VMs that are already slowed down by contention consume as many physical resources as possible.

The other key aspect of our work is enlightenment, which has been used in various ways. Satori [28] uses the knowledge of guests about their reclaimable memory, for memory consolidation between multiple VMs. Ballooning [41] is another well-established form of explicit guest involvement in the memory reclamation by the hypervisor. Our work applied the concept of enlightenment to migration, and investigated how explicit guest support can improve migration performance. An alternative approach to full enlightenment through guest cooperation is to use hypervisor-level inference. Kaleidoscope [10] exploits memory semantics inferred from architecture specifications, and uses the obtained information for fast VM cloning. JAVMM [18] expedites migration of VMs containing Java applications, by having them inform the hypervisor of memory containing garbage-collectable objects and avoiding its transfer. There also exists previous work that takes the task of migration into the application layer, instead of passing available knowledge down to systems software. Zephyr [14], Madeus [29], and ElasTraS [13] are examples of such approaches applied to databases. Imagen [25] targets active sessions for JavaScript web applications, migrating them between devices for ubiquitous access. Wang et al. [42] proposed a fault tolerance scheme for MPI applications that triggers their live migration.

7. Conclusion

We presented enlightened post-copy, an approach to urgently migrating VMs under contention. It addresses aspects of VM migration differing from the focus of the existing approaches: urgent execution transfer of the migrated VM, thereby recovering the aggregate performance of the contending VMs rapidly. Live migration, which is the current standard, exhibits undesirable characteristics in these aspects due to its design choices. Departing from its blackbox nature, we treat migration as a native functionality of the guest OS. Enlightened post-copy exploits this cooperation between the guest OS and the hypervisor, allowing prioritized post-copy state transfer that achieves the above objectives. Our prototype, implemented in guest Linux and qemu-kvm, requires only moderate changes to the guest kernel, and it demonstrates that the cooperative approach resolves the contention between VMs up to several times faster than live migration.

Acknowledgements

This research was supported by the National Science Foundation under grant number CNS-1518865, the Alfred P. Sloan Foundation, and the Defense Advanced Research Projects Agency under grant number FA8650-11-C-7190. Additional support was provided by the Intel Corporation, Vodafone, Crown Castle, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to their employers or funding sources.

References

[1] AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting. http://aws.amazon.com/ec2.

[2] memcached - a distributed memory object caching system. http://memcached.org.

[3] OLTPBenchmark. http://oltpbenchmark.com/wiki.

[4] The Apache Cassandra Project. http://cassandra.apache.org.

[5] userfaultfd v4 [LWN.net]. https://lwn.net/Articles/644532.

[6] Virtio - KVM. http://www.linux-kvm.org/page/Virtio.

[7] VMware vSphere vMotion Architecture, Performance and Best Practices in VMware vSphere 5. http://www.vmware.com/files/pdf/vmotion-perf-vsphere5.pdf.

[8] Yahoo! Cloud Serving Benchmark (YCSB). https://github.com/brianfrankcooper/YCSB/wiki.

[9] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the Third International Conference on Virtual Execution Environments (VEE '07)*, San Diego, CA, USA, June 2007.

[10] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, A. Lagar-Cavilla, and E. de Lara. Kaleidoscope: Cloud Micro-elasticity via VM State Coloring. In *Proceedings of the Sixth ACM European Conference on Computer Systems (EuroSys '11)*, Salzburg, Austria, April 2011.

[11] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The Collective: A Cache-based System Management Architecture. In *Proceedings of the Second Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI '05)*, Boston, MA, USA, May 2005.

[12] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of the Second Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI '05)*, Boston, MA, USA, May 2005.

[13] S. Das, D. Agrawal, and A. El Abbadi. ElasTraS: An Elastic, Scalable, and Self-managing Transactional Database for the Cloud. *ACM Transactions on Database Systems*, 38(1), April 2013.

[14] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*, Athens, Greece, June 2011.

[15] S. Hacking and B. Hudzia. Improving the Live Migration Process of Large Enterprise Applications. In *Proceedings of the Third International Workshop on Virtualization Technologies in Distributed Computing (VTDC '09)*, Barcelona, Spain, June 2009.

[16] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy Live Migration of Virtual Machines. *SIGOPS Operating Systems Review*, 43(3), July 2009.

[17] M. R. Hines and K. Gopalan. Post-copy Based Live Virtual Machine Migration Using Adaptive Pre-paging and Dynamic Self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)*, Washington, DC, USA, March 2009.

[18] K.-Y. Hou, K. G. Shin, and J.-L. Sung. Application-assisted Live Migration of Virtual Machines with Java Applications. In *Proceedings of the Tenth ACM European Conference on Computer Systems (EuroSys '15)*, Bordeaux, France, April 2015.

[19] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, and E. Roman. Optimized Pre-copy Live Migration for Memory Intensive Applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, Seattle, WA, USA, November 2011.

[20] A. Koto, H. Yamada, K. Ohmura, and K. Kono. Towards Unobtrusive VM Live Migration for Cloud Computing Platforms. In *Proceedings of the Third ACM SIGOPS Asia-Pacific Conference on Systems (APSys '12)*, Seoul, South Korea, July 2012.

[21] M. Kozuch, M. Satyanarayanan, T. Bressoud, and Y. Ke. Efficient State Transfer for Internet Suspend/Resume. *Intel Research Pittsburgh Technical Report IRP-TR-02-03*, May 2002.

[22] M. A. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, USA, June 2002.

[23] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proceedings of the Fourth ACM European Conference on Computer Systems (EuroSys '09)*, Nuremberg, Germany, April 2009.

[24] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In *Proceedings of the Eighteenth ACM International Symposium on High Performance Distributed Computing (HPDC '09)*, Garching, Germany, June 2009.

[25] J. Lo, E. Wohlstadter, and A. Mesbah. Live Migration of JavaScript Web Apps. In *Proceedings of the Twenty-Second International Conference on World Wide Web (WWW '13 Companion)*, Rio de Janeiro, Brazil, May 2013.

[26] P. Lu, A. Barbalace, and B. Ravindran. HSG-LM: Hybrid-copy Speculative Guest OS Live Migration Without Hypervisor. In *Proceedings of the Sixth International Systems and Storage Conference (SYSTOR '13)*, Haifa, Israel, June 2013.

[27] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty. XvMotion: Unified Virtual Machine Migration over Long Distance. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC '14)*, Philadelphia, PA, USA, June 2014.

[28] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman. Satori: Enlightened Page Sharing. In *Proceedings of the 2009*

USENIX Annual Technical Conference (USENIX ATC '09), San Diego, CA, USA, June 2009.

[29] T. Mishima and Y. Fujiwara. Madeus: Database Live Migration Middleware Under Heavy Workloads for Cloud Environment. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, Melbourne, Australia, May 2015.

[30] M. Nelson, B.-H. Lim, and G. Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of the 2005 USENIX Annual Technical Conference (USENIX ATC '05)*, Anaheim, CA, USA, April 2005.

[31] C. Peng, M. Kim, Z. Zhang, and H. Lei. VDN: Virtual Machine Image Distribution Network for Cloud Data Centers. In *Proceedings of INFOCOM 2012*, Orlando, FL, USA, March 2012.

[32] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein. VMTorrent: Virtual Appliances On-demand. In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*, New Delhi, India, August 2010.

[33] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein. VMTorrent: Scalable P2P Virtual Machine Streaming. In *Proceedings of the Eighth International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, Nice, France, December 2012.

[34] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual Appliances for Deploying and Maintaining Software. In *Proceedings of the Seventeenth USENIX Conference on System Administration (LISA '03)*, San Diego, CA, USA, October 2003.

[35] C. Sapuntzakis and M. S. Lam. Virtual Appliances in the Collective: A Road to Hassle-free Computing. In *Proceedings of the Ninth Conference on Hot Topics in Operating Systems - Volume 9 (HotOS '03)*, Lihue, HI, USA, May 2003.

[36] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, USA, December 2002.

[37] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, A. Surie, D. R. O'Hallaron, A. Wolbach, J. Harkes, A. Perrig, D. J. Farber, M. A. Kozuch, C. J. Helfrich, P. Nath, and H. A. Lagar-Cavilla. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing*, 11(2), March 2007.

[38] A. Shribman and B. Hudzia. Pre-Copy and Post-copy VM Live Migration for Memory Intensive Applications. In *Proceedings of the Eighteenth International Conference on Parallel Processing Workshops (Euro-Par '12)*, Rhodes Island, Greece, August 2012.

[39] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen. Parallelizing Live Migration of Virtual Machines. In *Proceedings of the Ninth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '13)*, Houston, TX, USA, March 2013.

[40] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In *Proceedings of the Seventh ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*, Newport Beach, CA, USA, March 2011.

[41] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, USA, December 2002.

[42] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Process-level Live Migration and Back Migration in HPC Environments. *Journal of Parallel and Distributed Computing*, 72(2), February 2012.

[43] A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble. Constructing Services with Interposable Virtual Hardware. In *Proceedings of the First Conference on Symposium on Networked Systems Design and Implementation - Volume 1 (NSDI '04)*, San Francisco, CA, USA, March 2004.

[44] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In *Proceedings of the Seventh ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*, Newport Beach, CA, USA, March 2011.