Application-Assisted Live Migration of Virtual Machines with Java Applications

Kai-Yuan Hou, Kang G. Shin, Jan-Lung Sung

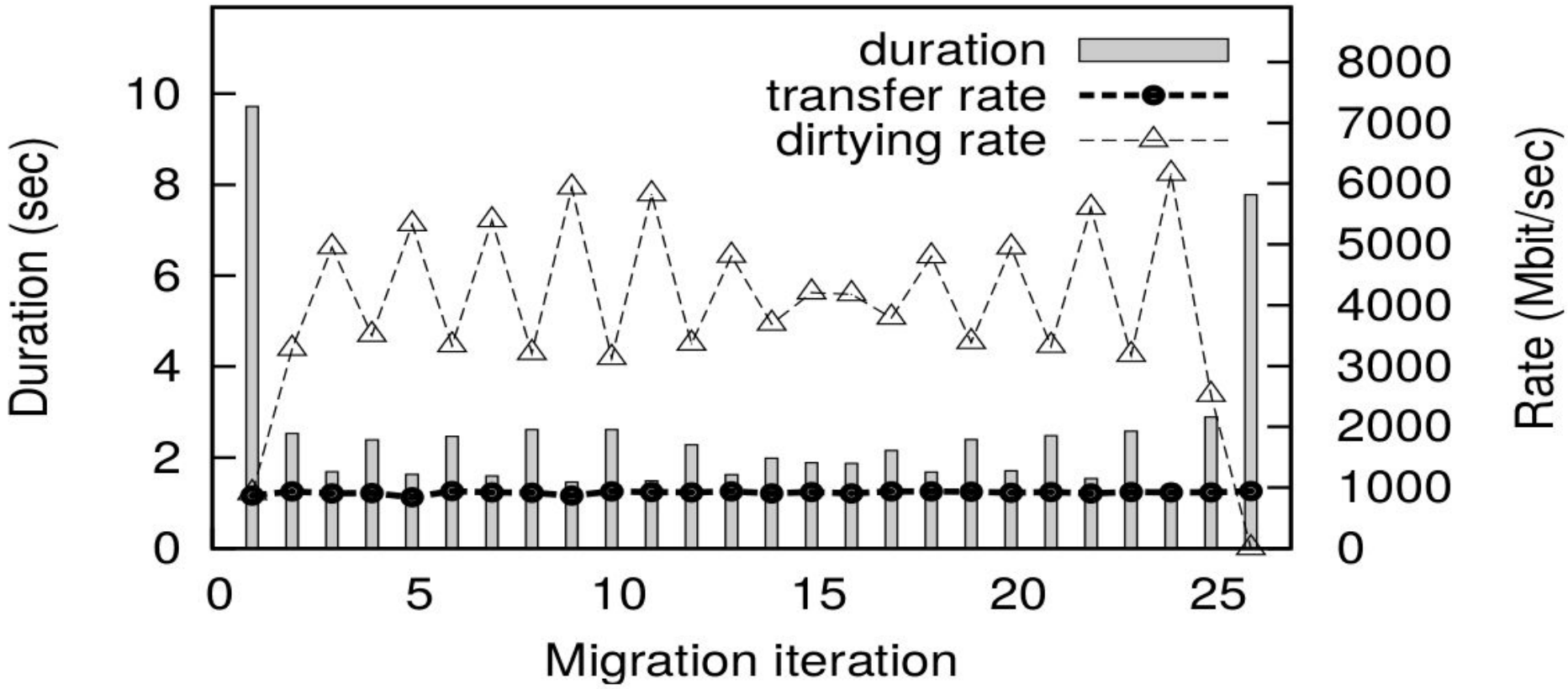
Presentation by Jan Kopański

Problem

Live migration of virtual machines (VMs) can consume:

- excessive time
- excessive resources

if VM memory pages get dirtied faster than their content can be transferred to the destination.


duration
transfer rate
dirtying rate
Duration (sec)
Rate (Mbit/sec)
Migration iteration
0
2
4
6
8
10
0
1000
2000
3000
4000
5000
6000
7000
8000
0
5
10
15
20
25

Existing solutions

1. transfer memory pages faster
    a. using high-speed networks capable of Remote Direct Memory Access (RDMA) like InfiniBand
2. slow down the memory dirtying rate
    a. moving processes to a wait queue after they generate more than a certain number of dirty pages
3. transfer less data for the dirty memory pages
    a. compression - CPU intensive
    b. deduplication
    c. skip over frequently dirtied pages during live iterations - must be transferred in the last iteration
    d. page cache pages can be skipped - need to be reproduced, VM performance may degrade after migration
    e. free pages can be skipped - works only in lightly-loaded VMs

JAVMM Solution Idea

Skip transfer of selective memory pages during migration.

Skip transfer of garbage in the frequently-dirtied Young generation of the Java heap.

Skip transfer of selective VM memory based on application semantics, by exploiting applications’ assistance.

What Memory to Skip Migrating?

- reproducible
- not required for correct application execution
- recoverable from application logs and intermediate results
- caches (web cache and database buffer pool)
- garbage

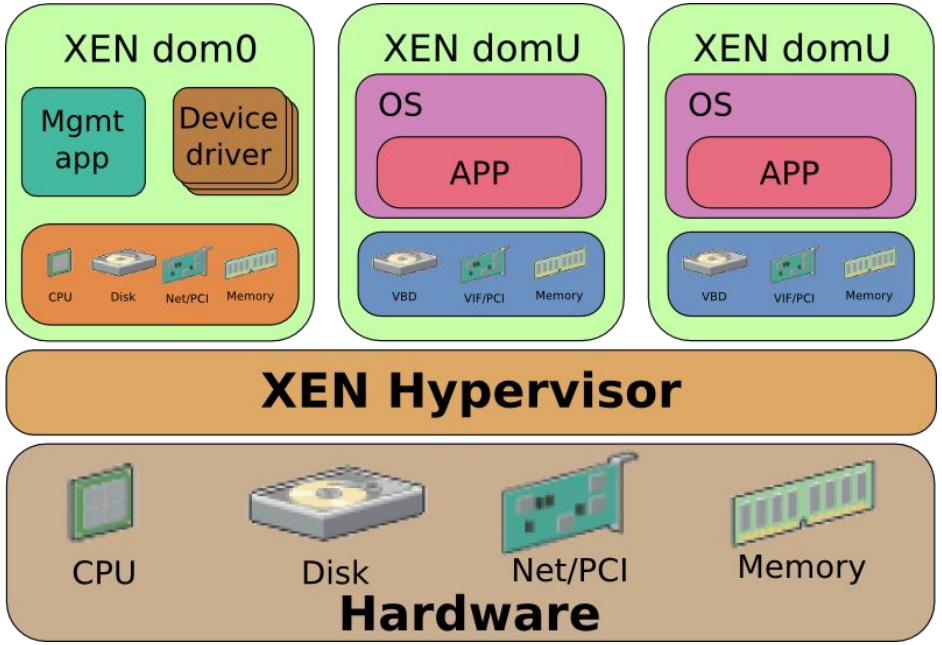
Challenges and Design Principles

- migration tool and an application in the guest VM are unaware of the execution of each other
- They address memory differently
  - the migration tool transfers VM memory pages based on Page Frame Numbers
  - the application executes based on Virtual Addresses

- A running application identifies which areas of its memory need not be migrated, and informs the migration tool
- The migration tool needs to know which memory pages to skip transfer, without incorporating application semantics.

Xen

Hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.



Framework

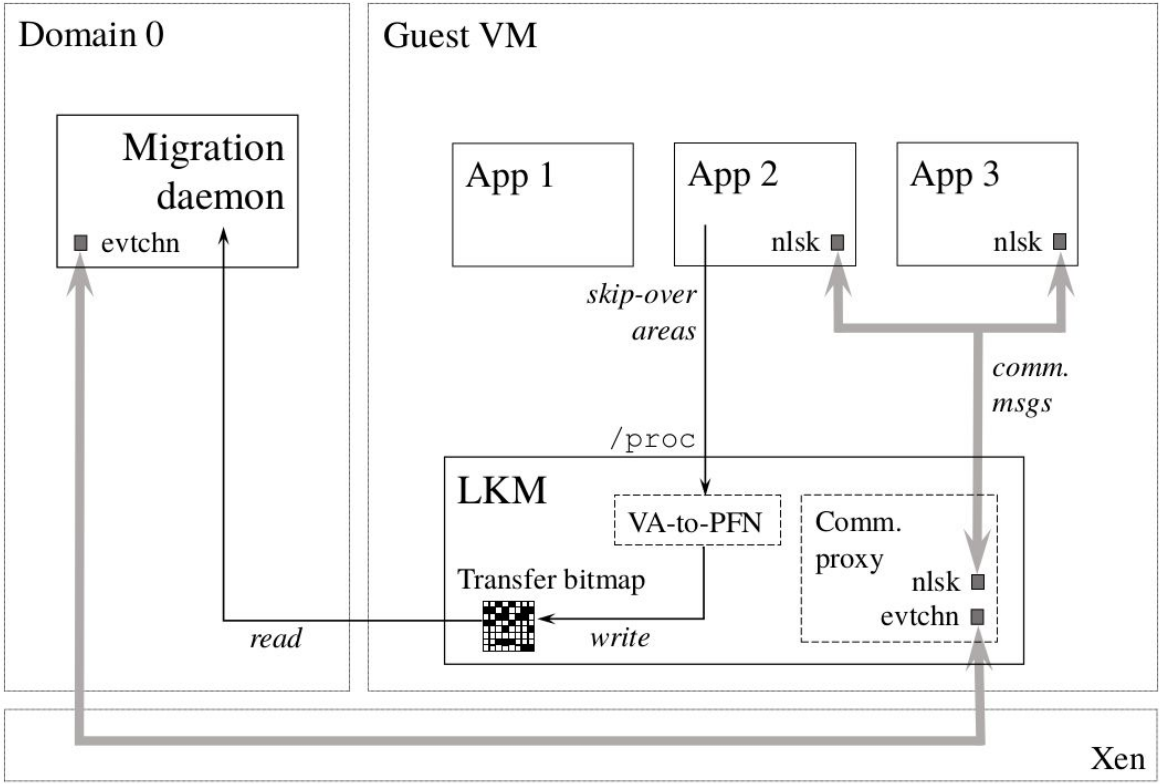
LKM - Loadable Kernel Module

VA - Virtual Address

PFN - Page Frame Number

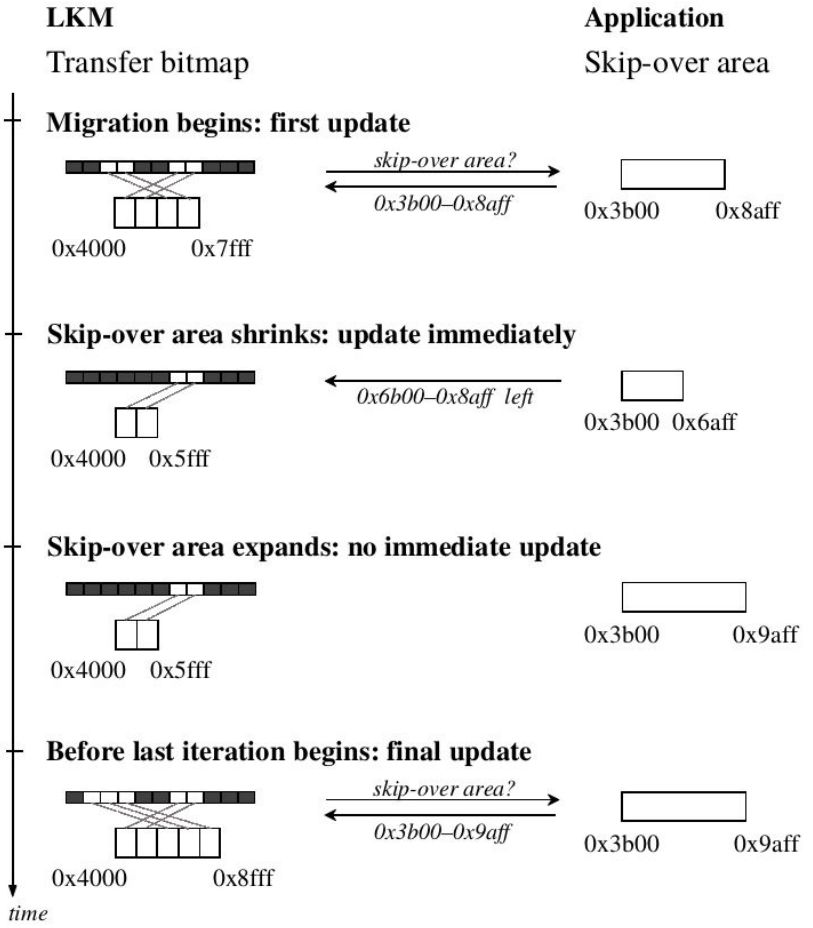
nlsk - netlink socket

evtchn - event channel



Updating the Transfer Bitmap

1. The transfer bitmap is initialized with all bits set
2. LKM remembers the VA range, finds the associated PFNs by page table walks
3. VM continues to run, and each skip-over area may expand or shrink
4. Application notify the LKM of the VA ranges leaving the area. The LKM updates its memory of the area's VA range, and immediately, sets the transfer bits of the PFNs



Current implementation

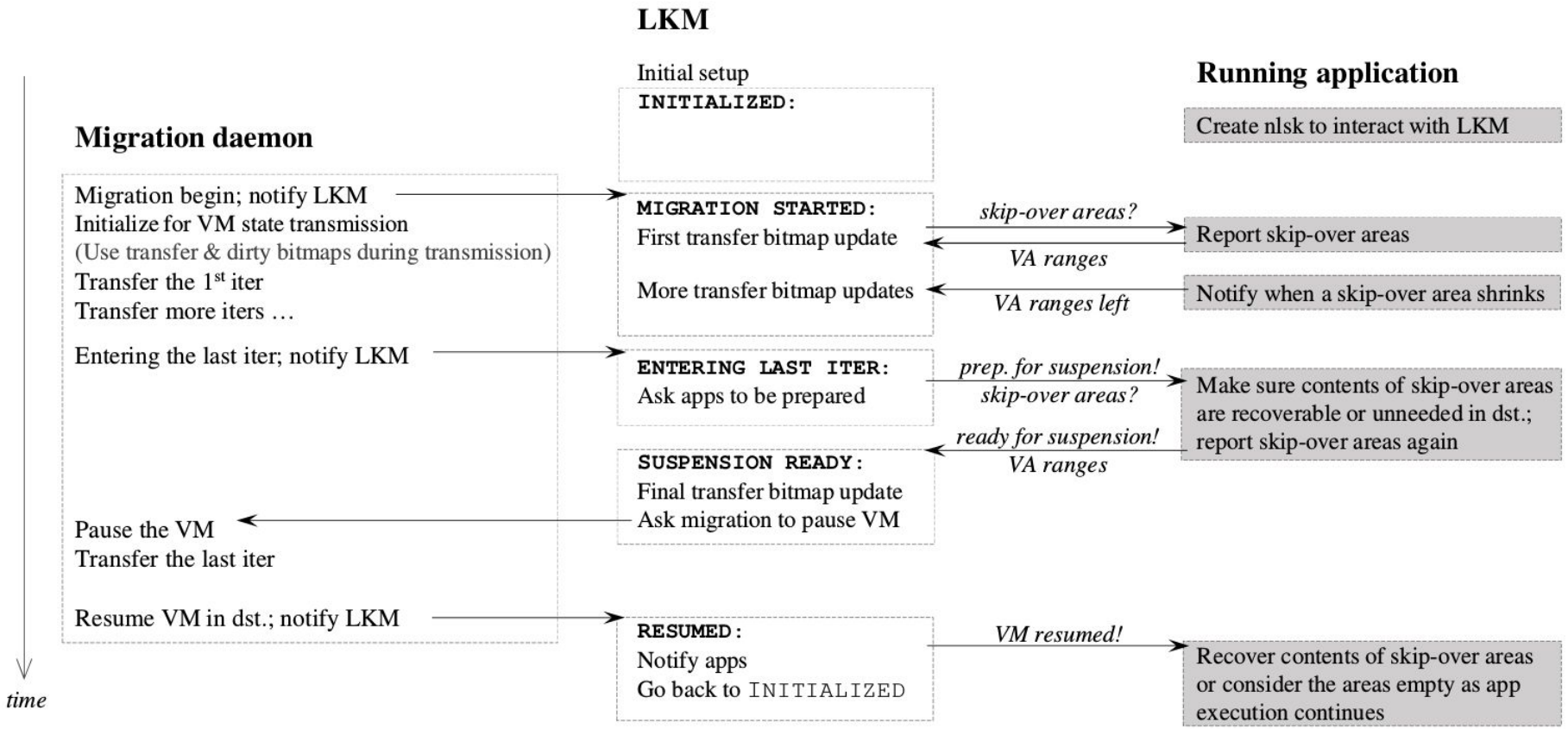
If a PFN joins or leaves a skip-over area with no changes in the area's VA range, the transfer bitmap is not updated.

PFN mapping changed in three possible ways:

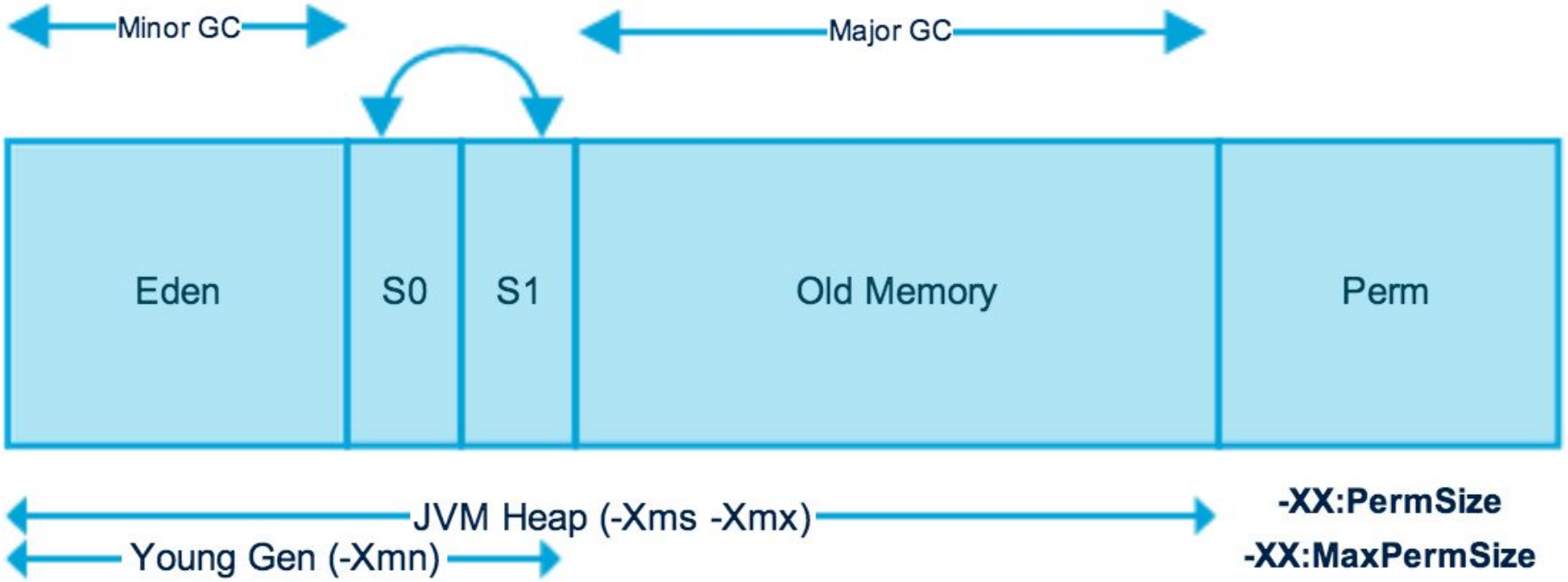
1. from null to p, when a page frame is allocated
2. from p old to p, when the page is remapped due to page sharing
3. from p old to null, when the page is swapped out

Case 2, 3 - currently assume their absence

Migration Workflow

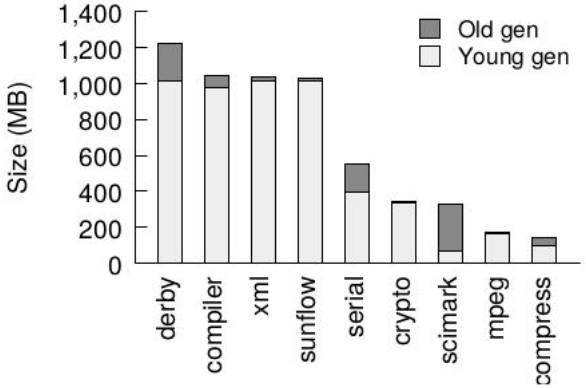


Background on Java Heap Management

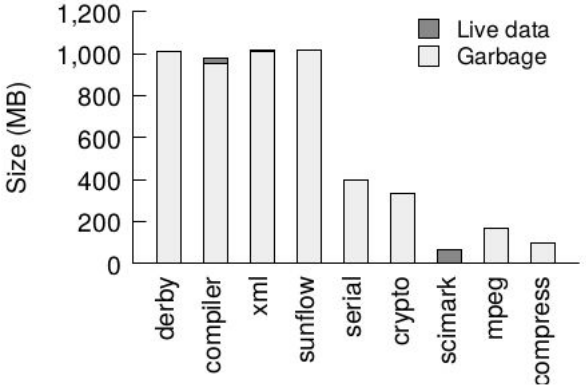


Garbage in Java Heap

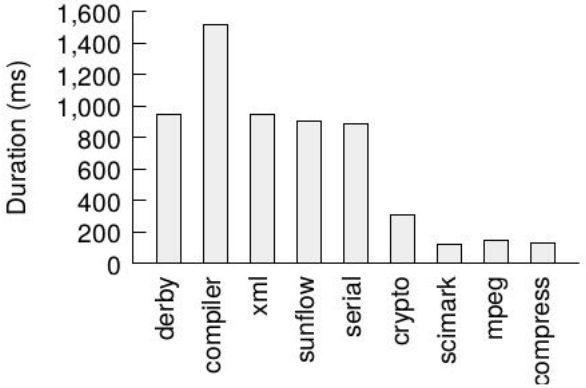
Workload	Description
derby	Apache Derby [6] database with business logic
compiler	OpenJDK 7 front-end compiler [7]
xml	Apply style sheets to XML documents
sunflow	An open-source image rendering system [9]
serial	Serialize and deserialize primitives and objects
crypto	Sign and verify with cryptographic hashes
scimark	Compute the LU factorization of matrices
mpeg	MP3 decoding
compress	Compression by a modified Lempel-Ziv method



(a) Memory consumption of the Java heap



(b) Garbage vs. live data in a minor GC



(c) Duration of a minor GC

Observations

Observation 1. The Young generation can be large and continuously dirtied, due to the high object allocation rate of the workload.

- up to 98% of the heap memory is consumed by the Young generation

Observation 2. A significant portion of the Young generation memory may contain garbage, due to the workload's use of short-lived objects.

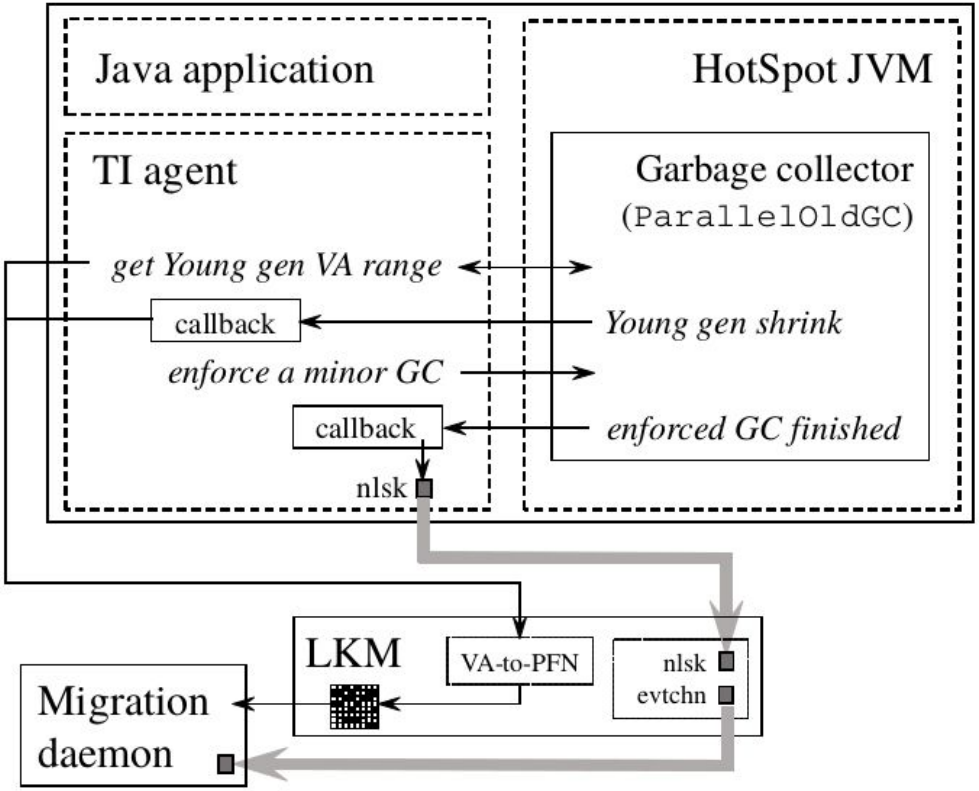
- over 97% of the Young generation memory is garbage collected in a minor GC

Observation 3. Collecting Young generation garbage may be faster than sending them over a bottleneck network link.

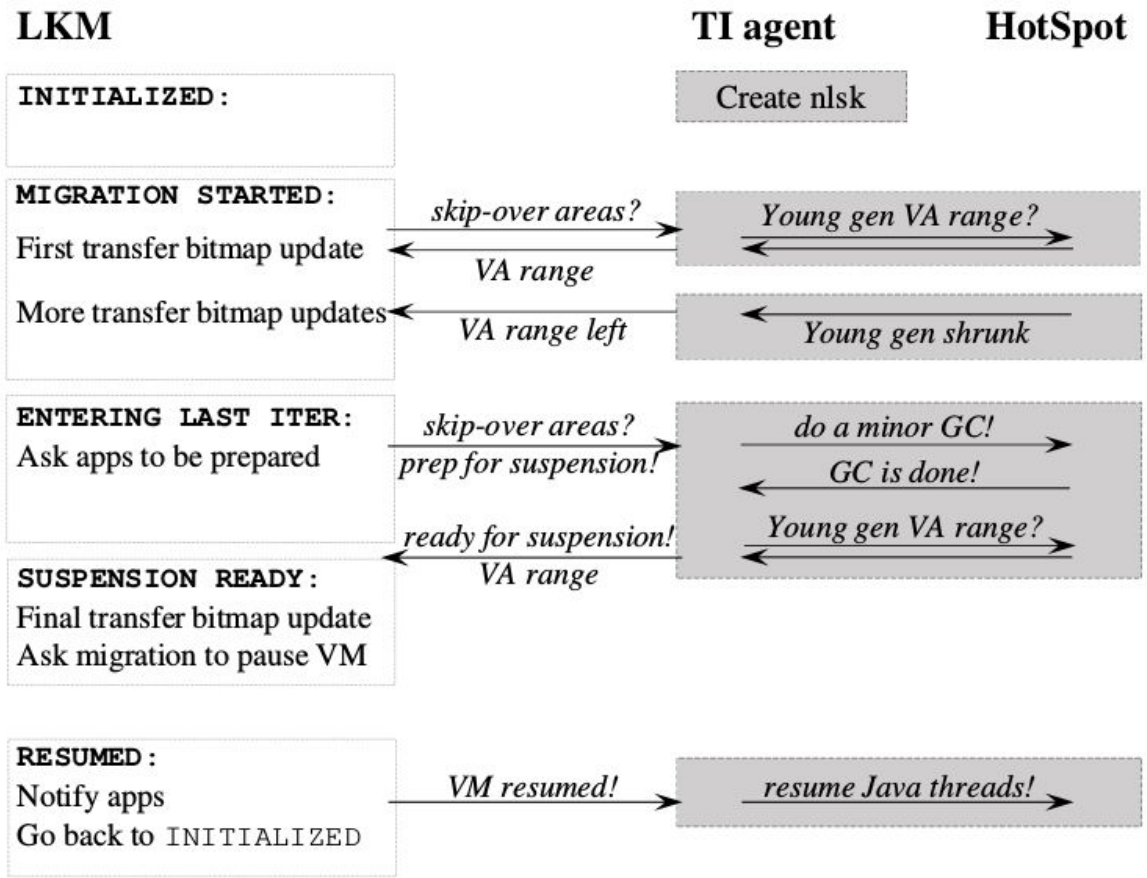
System Overview of JAVMM

TI - JVM Tool Interface

- JAVMM skips transfer of the garbage with assistance of JVM, which knows where garbage objects are located in memory.
- Garbage objects are scattered among live data, and their locations keep changing as objects become unreferenced.



Workflow of JAVMM

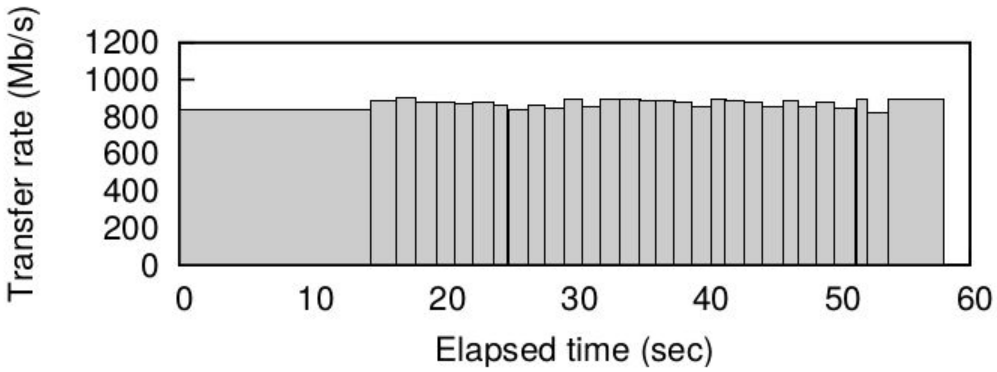


Evaluation - specification

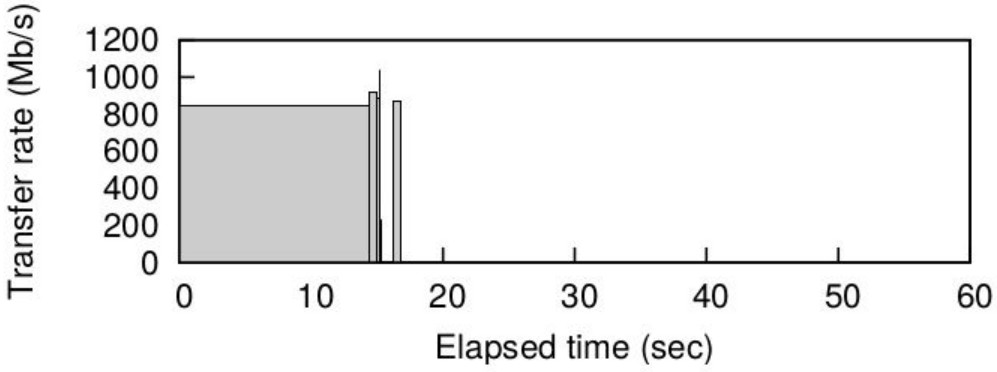
- run each workload for 10 minutes
- VM configured with 2GB memory
- 4 vCPUs

- migrate the VM, between two HP Proliant BL465c blades
- gigabit Ethernet LAN
- each blade is equipped with two dual-core AMD Opteron
- 2.2 GHz CPUs
- 12GB RAM.

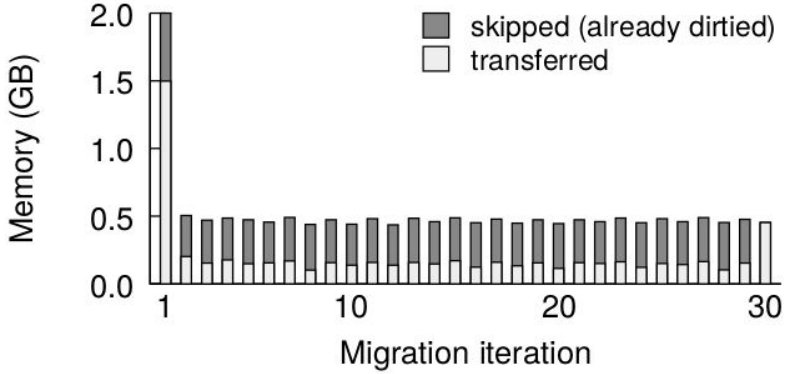
Progress of Migration - compiler workload



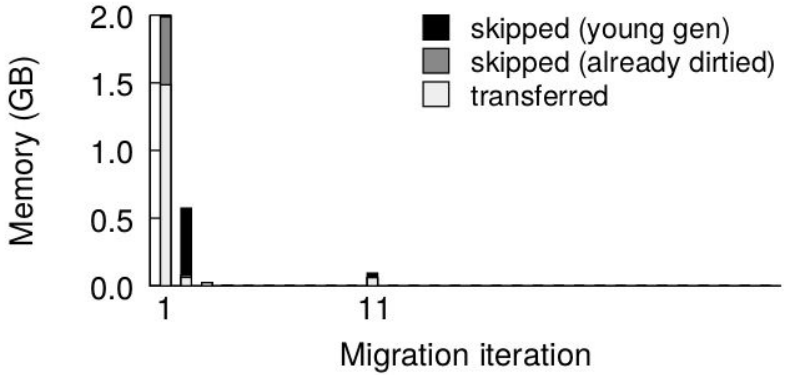
(a) Xen



(b) JAVMM



(a) Xen



(b) JAVMM

Performance of Migration

Workload	Max allowed Young gen (MB)	Observed when migrated Young gen (MB)	Old gen (MB)
derby	1024	1024	259
crypto	1024	456	18
scimark	1024	128	486

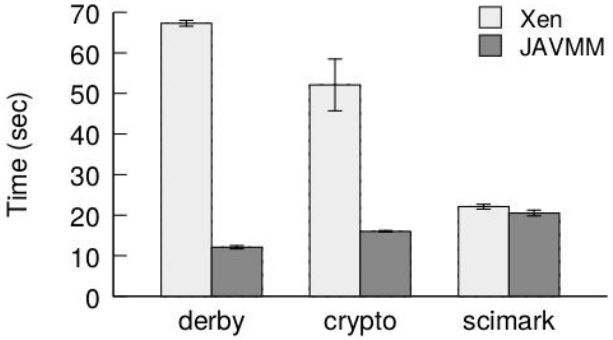
- Category 1. The workload has a high object allocation rate, and uses mostly short-lived objects. As a result, the Young generation quickly grows to the maximum size. The derby, compiler, xml and sunflow workloads are in this category.

- Category 2. The workload has a medium object allocation rate, and uses mostly short-lived objects. The Young generation grows faster than the Old generation, albeit not maximally utilized. The serial, crypto, mpeg and compress workloads are in this category.

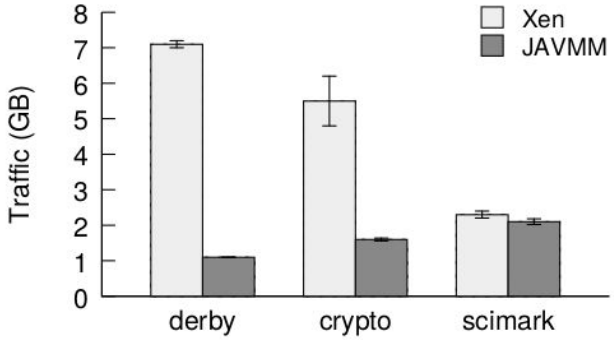
- Category 3. The workload has a low object allocation rate, and uses mostly long-lived objects. It thus has a small Young generation and a large Old generation. Scimark is the only workload in this category.

How fast does JAVMM migrate a Java VM?

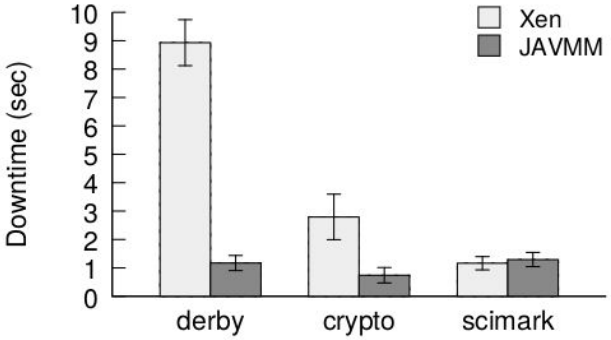
JAVMM reduces the migration time of derby by 82%. JAVMM also achieves a 69% reduction of migration time for the crypto VM. For scimark, JAVMM can skip over little Young generation memory.



(a) Total migration time



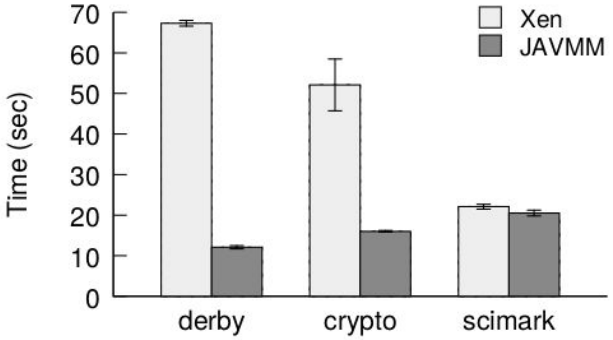
(b) Total migration traffic



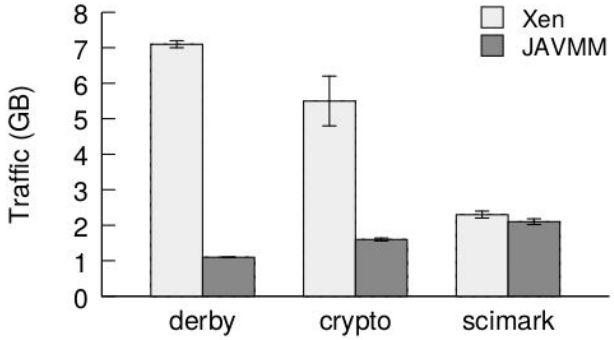
(c) Workload downtime due to migration

How much resource does JAVMM use for migration?

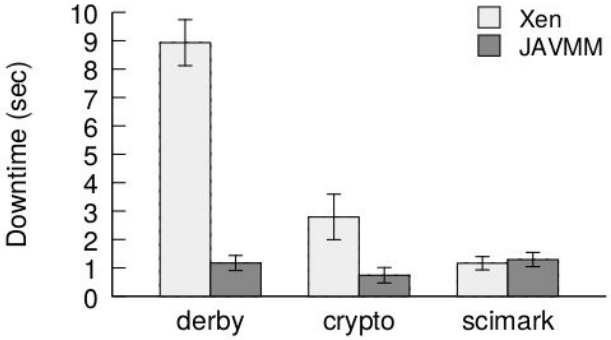
Compared to Xen, JAVMM reduces migration traffic for derby and crypto by 84% and 72%, respectively. For scimark, JAVMM achieves a 10% reduction of migration traffic.JAVMM also uses up to 84% less CPU time than Xen in migrating the VMs.



(a) Total migration time



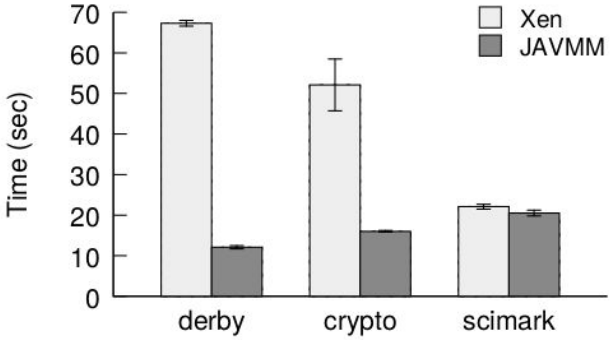
(b) Total migration traffic



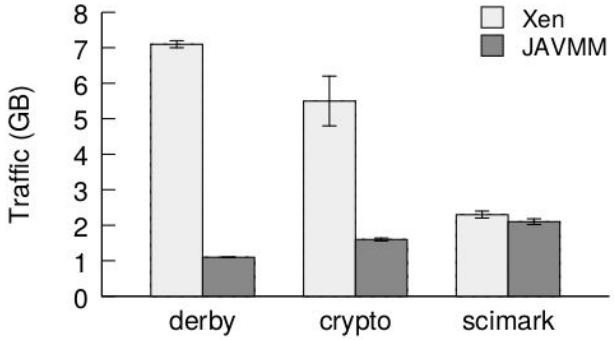
(c) Workload downtime due to migration

How much does JAVMM affect application performance?

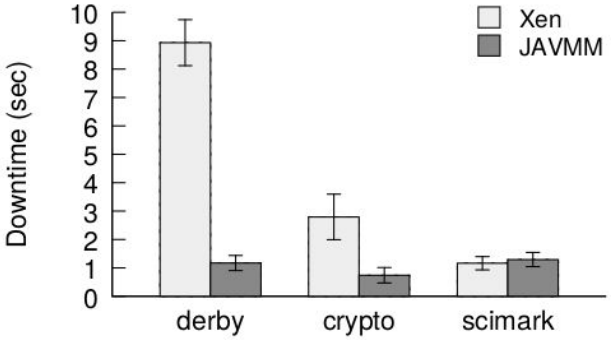
Derby experiences 1.2 seconds of downtime when the VM is migrated by JAVMM, 83% shorter than the 9-second downtime when the VM is migrated by Xen. However, for scimark, JAVMM imposes a 10% longer downtime than Xen. JAVMM takes time to perform the enforced GC.



(a) Total migration time



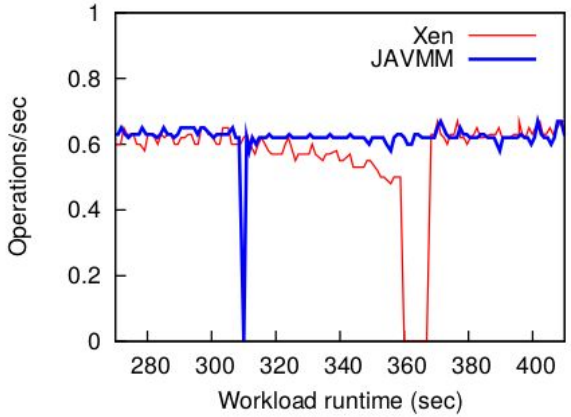
(b) Total migration traffic



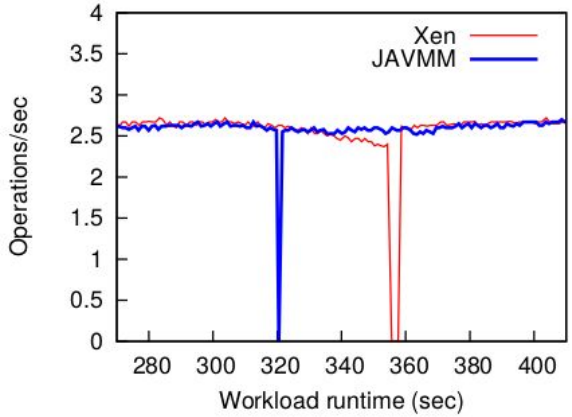
(c) Workload downtime due to migration

How much does JAVMM affect application performance?

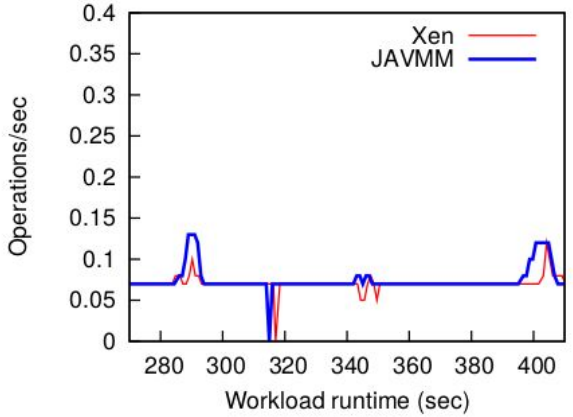
VM is migrated after the workload runs for 300 seconds.



(a) Derby

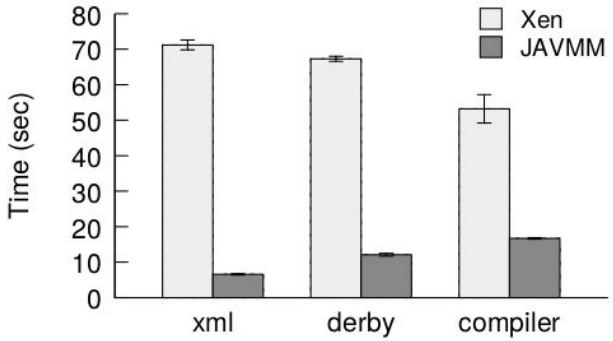


(b) Crypto

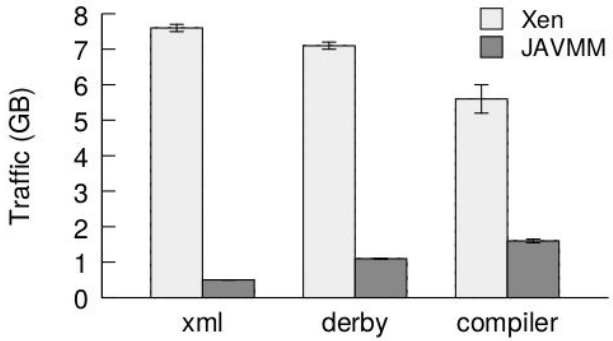


(c) Scimark

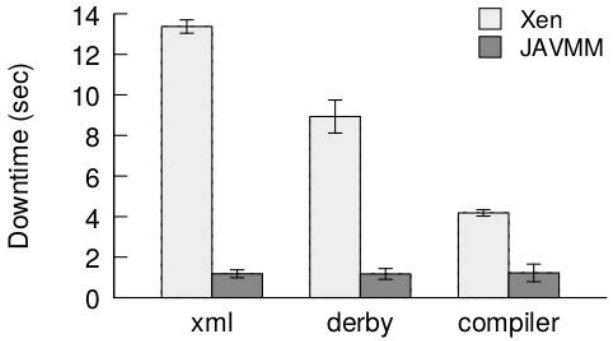
Workload	Max allowed Young gen (MB)	Observed when migrated Young gen (MB)	Old gen (MB)
xml	1536	1536	28
derby	1024	1024	259
compiler	512	512	86



(a) Total migration time



(b) Total migration traffic



(c) Workload downtime due to migration

Future Extensions

- Use JAVMM for large VMs with fast networks.
- Use JAVMM with other garbage collectors.
- Apply the proposed application-assisted live migration framework to other applications. JAVMM can be used for applications written in other languages that run on JVM and use JVM’s garbage collectors.
- Support large and multiple applications. In our proposed framework, the LKM updates the transfer bitmap on applications’ behalf. It can coordinate concurrent bitmap updates from multiple applications.
- Enhance the proposed framework for security.

Future Extensions

- Make the proposed framework intelligent. For example, while an application may report all of its skip-over areas to the LKM, the LKM can make a more informed decision on which ones to skip transferring in a particular migration event, if information such as the sizes of the VM and the skip-over areas, the current network speed and the average memory dirtying rate is available and taken into account.
- Incorporate compression.