Host memory overcommit may cause guest memory to be swapped. When guest vcpu access memory swapped out by a host its execution is suspended until memory is swapped back. Asynchronous page fault is a way to try and use guest vcpu more efficiently by allowing it to execute other tasks while page is brought back into memory.
Part I

How KVM Handles Guest Memory and What Inefficiency it Has With Regards to Host Swapping
Mapping guest memory into host memory

Guest Memory

shadow/nested page tables

Host Memory
But we do it on demand
Page fault happens on first guest access
What happens on a page fault?

1. VMEXIT
2. kvm_mmu_page_fault()
3. gfn_to_pfn()
4. get_user_pages_fast()
   - no previously mapped page and no swap entry found
   - empty page is allocated
5. page is added into shadow/nested page table
What happens on a page fault?

1. VMEXIT
2. `kvm_mmu_page_fault()`
3. `gfn_to_pfn()`
4. `get_user_pages_fast()`

   no previously mapped page and no swap entry found

   empty page is allocated

5. page is added into shadow/nested page table
What happens on a page fault?

1. VMEXIT
2. kvm_mmu_page_fault()
3. gfn_to_pfn()
4. get_user_pages_fast()
   - no previously mapped page and no swap entry found
   - empty page is allocated
5. page is added into shadow/nested page table
What happens on a page fault?

1. VMEXIT
2. kvm_mmu_page_fault()
3. gfn_to_pfn()
4. get_user_pages_fast()
   - no previously mapped page and no swap entry found
   - empty page is allocated
5. page is added into shadow/nested page table
What happens on a page fault?

1. VMEXIT
2. kvm_mmu_page_fault()
3. gfn_to_pfn()
4. get_user_pages_fast()
   - no previously mapped page and no swap entry found
   - empty page is allocated
5. page is added into shadow/nested page table
What happens on a page fault?

1. VMEXIT
2. `kvm_mmu_page_fault()`
3. `gfn_to_pfn()`
4. `get_user_pages_fast()`
   - no previously mapped page and no swap entry found
   - empty page is allocated
5. page is added into shadow/nested page table
What happens on a page fault?

1. VMEXIT
2. `kvm_mmu_page_fault()`
3. `gfn_to_pfn()`
4. `get_user_pages_fast()`
   - no previously mapped page and no swap entry found
   - empty page is allocated
5. page is added into shadow/nested page table
On each page fault one page is mapped
At the end all used pages are mapped
Swapped out page is removed from shadow pt
Page is accessed again

guest access

Guest Memory

shadow/nested page tables

Host Memory

swap
What happens on a page fault now?

1. VMEXIT
2. kvm_mmu_page_fault()
3. gfn_to_pfn()
4. get_user_pages_fast()
   - swap entry is found
   - page swap-in process is initiated
   - vcpu thread goes to sleep until page is swapped in
5. page is added into shadow/nested page table
What happens on a page fault now?

1. VMEXIT
2. kvm_mmu_page_fault()
3. gfn_to_pfn()
4. get_user_pages_fast()
   - swap entry is found
   - page swap-in process is initiated
   - vcpu thread goes to sleep until page is swapped in
5. page is added into shadow/nested page table
New shadow pt mapping is created
Part II

Lets take close look inside a guest
Different pages belong to different processes

Guest Memory

shadow/nested page tables

Host Memory

Process A
Process B
Page belonging to Process A is swapped out
Process A tries to access its page again
New shadow pt mapping is created
Part III
What is Asynchronous Page Fault and How it Can Help us
Asynchronous Page Fault (APF)
New kind of exception

Actually it is not one, but two kind of exceptions:

**APF: Page not Present**
Guest tried to access page which is swapped out by a hypervisor.

**APF: Page Ready**
Page is now swapped in and can be accessed from a guest
APF shares exception vector with regular #PF

PV guest can distinguish between regular page fault and APF by checking fault reason in per cpu memory location. It would be nice to have one exception vector to be reserved for virtualization purposes by Intel and AMD.
How it Work

- Process A accesses page swapped out by the host.
  - GUP is done by dedicated thread. Vcpu gets “Page not Present” exception.
  - Guest puts Process A to sleep and schedule another process.
  - Page is ready. Vcpu gets “Page Ready” exception.
  - Guest can schedule Process A back to run on vcpu.
How it Work

- Process A accesses page swapped out by the host.
- GUP is done by dedicated thread. Vcpu gets “Page not Present” exception.
- Guest puts Process A to sleep and schedule another process.
- Page is ready. Vcpu gets “Page Ready” exception.
- Guest can schedule Process A back to run on vcpu.
How it Work

- Process A accesses page swapped out by the host.
- GUP is done by dedicated thread. Vcpu gets “Page not Present” exception.
- Guest puts Process A to sleep and schedule another process.
- Page is ready. Vcpu gets “Page Ready” exception.
- Guest can schedule Process A back to run on vcpu.
How it Work

- Process A accesses page swapped out by the host.
- GUP is done by dedicated thread. Vcpu gets “Page not Present” exception.
- Guest puts Process A to sleep and schedule another process.
- Page is ready. Vcpu gets “Page Ready” exception.
- Guest can schedule Process A back to run on vcpu.
How it Work

- Process A accesses page swapped out by the host.
- GUP is done by dedicated thread. Vcpu gets “Page not Present” exception.
- Guest puts Process A to sleep and schedule another process.
- Page is ready. Vcpu gets “Page Ready” exception.
- Guest can schedule Process A back to run on vcpu.
Enhancing GUP

- Need GUP version that will succeed only if page can be acquired without IO.
- \_\_get\_user\_pages\_fast() is not good enough. Will fail if page is in page or swap cache.
- Introduce new GUP variant: get\_user\_pages\_noio().
Part IV

Test Results
Benchmark

Application:
- 4 threads doing random memory access (faulting threads)
- 4 threads incrementing per thread counter (working threads)
- running for 1 minute
- output per thread counter value and sum of all counters

Execution environment:
- 4 VCPUS
- 2G guest memory
- runs inside 512M memory group *

* \frac{1}{4} overcommit
## Results

<table>
<thead>
<tr>
<th>With async pf:</th>
<th>Without async pf:</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker 0: 63972141051</td>
<td>worker 0: 30619912622</td>
</tr>
<tr>
<td>worker 1: 65149033299</td>
<td>worker 1: 33951339266</td>
</tr>
<tr>
<td>worker 2: 66301967246</td>
<td>worker 2: 31577780093</td>
</tr>
<tr>
<td>worker 3: 63423000989</td>
<td>worker 3: 33603607972</td>
</tr>
<tr>
<td>total: 258846142585</td>
<td>total: 129752639953</td>
</tr>
</tbody>
</table>

50% improvement!
Perf data from inside the guests

With async pf:

97.93% bm bm [.] work_thread
1.74% bm [kernel] [k] retint_careful
0.10% bm [kernel] [k] _raw_spin_unlock_irq
0.08% bm bm [.] fault_thread
0.05% bm [kernel] [k] _raw_spin_unlock_irqrestore
0.02% bm [kernel] [k] __do_softirq
0.02% bm [kernel] [k] rcu_process_gp_end

Without async pf:

63.42% bm bm [.] work_thread
13.64% bm [kernel] [k] __do_softirq
8.95% bm bm [.] fault_thread
5.27% bm [kernel] [k] _raw_spin_unlock_irq
2.79% bm [kernel] [k] hrtimer_run_pending
2.35% bm [kernel] [k] run_timer_softirq
1.28% bm [kernel] [k] _raw_spin_lock_irq
The end.

Thanks for listening.