Working Set

BACKGROUND

These VMs represent various applications or services, each with varying levels of importance. The challenge lies in allocating memory fairly and efficiently.

Traditionally, the system allocates memory based on overall usage, similar to splitting rent equally among roommates. However, this approach has limitations:

- **Focus on Usage, Not Importance:** A high-usage VM might not be critical, while a low-usage VM might be essential but simply require more memory for optimal performance.
- **Inefficient Allocation:** Less important VMs could occupy memory that would be more beneficial for critical ones.

This approach doesn't consider the varying needs of the VMs, leading to potential performance issues for crucial applications.

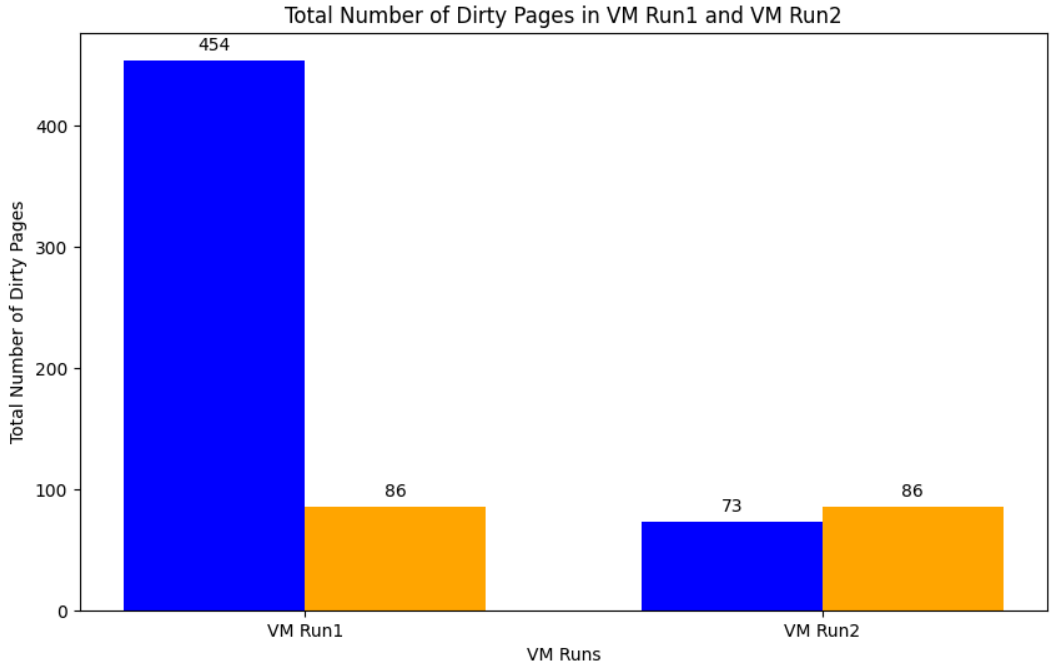
Our Solution: Working Set Estimation

To address this challenge, we propose a solution based on **working set estimation**. This approach focuses on understanding the actual memory needs of each VM, not just its current usage.

By estimating the working set, we can:

- **Prioritize Critical VMs:** Ensure essential VMs have the resources they need first.
- **Optimize Remaining Memory:** Allocate remaining memory more efficiently based on the potential benefit for each VM.

Working set estimation allows us to move beyond a simplistic allocation system and establish a fair and efficient approach tha



It prioritizes critical services while maximizing resource utilization.

Problem solution detail

We have used the approach as mentioned in the memory management paper. We tried to invalidate the TLB entries and MMU entries but that did not work. So we tried to track the dirty page by using KVM API - KVM_GET_DIRTY_LOG. Through this we aim to track the working set of the running VM. We tried setting up the the page table for only 2MB VM because we were getting few errors while setting the page table for a 32MB/1GB VM.

Experimental evaluation

We tested using two different synthetic workloads by simulating a code similar to matrix multiplication for two arrays in two scenarios. One array is of size 2^17 and of int data type and the other array is of size 2^16 of int data type. We see that the number of dirty pages in first VM

run is 454/512 and 86/512 for VM run 1 between one hypercall by the guest. Similarly, for the VM run 2 the number of dirty pages are 73/512 and 86/512 between one hypercall by the guest.

Discussion

We observe that this technique is not very effective as this technique tracks on ly dirty pages to estimate the working set. A better technique could be to invalidate the TLB and MMU entries and then track the accessed entries.