# Abstract

The aim of memory management techniques in virtual environment is to optimize, across time, the distribution of machine memory (RAM) among a maximal set of virtual machines (VMs). Hence over-commitment of memory becomes inevitable to achieve this lofty goal. When memory is being over-committed efficient management of memory becomes a necessity to ensure smooth running of the show. There are many techniques for managing memory in an over-committed scenario. E.g demand paging or ballooning based dynamic memory provisioning, duplicate content elimination etc.

But previous work have shown that there is no 'one-size-fits-all' technique among these that addresses the different challenges over-commitment throws at us. In fact using just any one of these as the lone management policy can prove counter productive and negatively impact the performance of applications running in virtual machines. This is because these techniques can end up adversely affecting existing memory management techniques of native OS in the VM or applications that manage their own memory.

This thesis explores a relatively new virtualized memory management paradigm called hypervisor caches which, when provisioned as a second chance page cache, mitigates the negative effects of traditional management techniques like ballooning on applications. A second chance hypervisor cache does this by caching pages that were evicted from a VM experiencing memory pressure due to balloon inflation. In fact such a hypervisor cache if combined with ballooning and memory sharing has the potential to become a holistic and efficient memory management policy. Specifically this thesis tries to exploit the presence of similar content in data centres, the fact that many data centres have their secondary storage on network and the presumption that accessing memory over a network should be faster than accessing the disk across the same. This thesis implements a second chance, disk page caching, hypervisor cache that pursues content de-duplication opportunities within itself and in other hypervisors. It also profiles the run-times of individual operations involved in the implementation and finds out the overheads of having such a cache on the end-to-end delays experienced by a read intensive synthetic workload.