Increasing Virtual Machine Density With KSM
Introduction to KSM

KSM is Linux kernel module that aim to save memory by searching and merging identical pages inside one or more memory areas in a way invisible to the user.
The Need For KSM

As an hypervisor one of the more critical missions kvm have is sharing resources among guests, the two most critical resources are:

Cpu resources sharing:
handled by the linux scheduler very effectivly,
when one guest idle its cpu time go into another guest.

Physical memory resources sharing:
handled by the linux swapping mechanism,
when a page is not used frequently by the guest it move into the disk and another page replace it.
The Problem With Swapping And How KSM Solve It

For some workloads such as desktops, high interactive responsive is needed, swapping is just not an option for such cases.

KSM solve this problem by allow the system to overcommit while still all the memory is stored in the physical ram, this allow fast access to the memory and keep the system interactive.
1. Applications using memory that part of it was merged by KSM should have the same behavior as if this pages were never merged. (reading / writing to that pages would result the same as if this pages were not merged)

2. Finding identical pages should be fast.

3. Merged pages should be treated by the VM like any other pages (including the ability to swap/migrate this pages)
Keep Applications Behavior The Same

KSM mark all the page tables pointed to merged page as readonly.

When one application write into merged page it receive a new private copy of this page.

this is all run in the page fault handler without the application will ever know about it.
KSM use hash table data structure in order to find identical pages in an effective way.

The hash table hold the page frame numbers and is accessed by the hash value of the page.
Fast Way To Find Identical Pages (reverse mapping)

Since pages can changed, KSM need to update its hash table after each iteration.

KSM use another hash table accessed by the page frame number that hold reverse mapping for each page.

Using this reverse mapping KSM check if the hash value of the page was changed, and update the hash table if needed.
Fast Way To Find Identical Pages (safe compare)

To avoid a race when page is changed after KSM found it identical to another page but before the page was merged:

KSM write protect the two pages and do full compare on both of them again, then it is safe to merge the two pages.
Let The VM Treat The Pages Like Any Other Page.

The VM use reverse mapping in order to do nice tricks such as swapping and page migration.

Unfortunately the VM reverse mapping is designed to use the fact that ptes pointing to a shared page have the same offset for each vma holding shared page inside linux, the rmap use this in order to find the pte inside given vma that point to a shared page, this is not true for pages that ksm merge, because of this KSM cannot use without modifications the current reverse mapping found in linux. (we dont want to use nonlinear vma)

to solve this, we created what we call: External Rmaps.
External Rmaps
what is it?

What is it?
External Rmaps is a new method added to the kernel that allows a driver to manage by itself its rmap.

Drivers using External Rmaps benefit from all the current code of the main VM, and able to use any feature that is supported by the VM just like any other drivers.
External Rmaps
The Goals

In any case not to increase the size of the page data structure!!!

Allow users of External Rmaps, to use all the great features of the main VM.
External Rmaps main data structure is:

```c
struct ext_rmap_ops {
    void (*page_add_rmap) — use to add new pte
    void (*page_remove_rmap) - use to remove pte
    void (*page_free) — call back to receive when page it release
    void (*pte_next) — call back to allow walking on the rmap
    swap_entry swp — used when the page is swapped.
};
```

In addition pointer to the `ext_rmap_ops` struct is added as a new field to the index and freelist union, this field is used when the page is external rmap page.
External Rmap Page is identified by checking the new PAGE_MAPPING_EXTERNAL_MAP bit, if this bit is set, the VM will call the Driver private External Rmap methods instead of the VM rmap.

(External Rmap page can be file backed or anonymous page)
In order to swap External Rmaps page, the VM uses the following logic:

when the pte is unmapped by `try_to_unmap_one()` the `swp_entry` is saved inside the `ext_rmap` structure and pointer to the `ext_rmap` structure is saved inside the pte (the pte is marked as `pte_file`)

when page is swapped in, Linux checks if the pte is `file_pte` and if the vma is not nonlinear, in case that these two cases are meet, we get the `ext_rmap` structure from the pte and in turn we get the `swp_entry` from there.
Woof!