RC23069 (W0401-092) January 19, 2004
Computer Science

IBM Research Report

Supporting Isolation for Fault and Power Management with Fully Virtualized Memory Systems

Freeman Rawson
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Supporting Isolation for Fault and Power Management with Fully Virtualized Memory Systems

Freeman Rawson

January 21, 2004

Abstract

Fully virtualized systems offer significant commercial advantages in certain markets by allowing users to consolidate many small servers onto a single large system that offers lower total cost and more efficient and flexible use of computing resources. With full virtualization, which means that the underlying hypervisor virtualizes processor, memory and I/O resources, the system becomes even more flexible since there is no fixed assignment of resources to partitions and operating system images. However, with such systems, the costs associated with failures and power management are much higher. For example, if a virtualized system fails, rather than losing a single system, the customer loses all of the consolidated system images, often on the order of 10^2 or even 10^3 logical servers. Similarly, unless the larger system offers power-management features comparable to those found in smaller systems, its power costs and power-related problems such as heat and potential failures are higher than the systems that it replaces. This work provides a partial solution to these problems in the area of system memory by offering a scheme for identifying and isolating if necessary the memory used by particular virtual images. It applies the well-known *reverse-mapping* technique to hypervisors and takes advantage of some likely characteristics of the hypervised environment to provide a compact and simple reverse mapping. With this reverse mapping, it then indicates how to reduce memory power consumption during periods of low utilization and how to isolate faulty portions of memory, affecting only those system images using the bad memory.

1 Motivation

IBM and other computer-systems vendors are increasingly relying on hypervisors supporting dynamic logical partitioning and full virtualization to support server consolidation, and it markets server consolidation to its customers as a cost-saving idea. In this context, full virtualization means that all of the resources of the system, the processors, the memory, and the I/O devices are all virtualized, so that each operating system image that the hypervisor runs uses *virtual resources*, ones that have no necessary or fixed relationship to the physical resources of the machine. In particular, the type of memory partitioning found in simple physical and logical partitioning commonly used in some previous-generation systems including the IBM pSeries assigned a fixed, static, contiguous region of memory to each partition and changed the assign only during re-partitioning.

Two examples of fully virtualizing hypervisors are IBM's zVM and its predecessors and the VMware ESX hypervisor for x86-based server systems.

Moreover, much of IBM's On-Demand strategy is based on the use of virtualized systems to support a disparate set of customers with varying computing requirements. But the use of large, partitioned or virtualized systems exacerbates the consequences of certain types of failures and complicates the problem of power management. For example, in a fully virtualized memory environment, if a portion of memory fails, the hypervisor may be unable to determine what partitions are affected and need to be terminated. The historically acceptable approach of having the entire system fail is not viable in a server consolidation environment where the failure of the machine may lead to the failure of hundreds of servers. Similarly, if there is a need to reduce power consumption, due, for example, to low utilization, high operating temperature or tight power supplies, the hypervisor needs to be able to determine what partitions are affected and make intelligent decisions about what part of memory to put into a low-power state.

These changes motivate the investigation of techniques for memory management that retain the full virtualization of memory while allowing one to control the scope of failures and reduce memory power consumption whenever possible. In both cases, the key is successful *isolation* of the portions of memory belonging to a particular partition, virtual machine or system image. Isolation allows the system to:

- identify affected system images when it detects a hardware error in a memory page frame
- limit the damage caused by a hardware memory error to a subset of the currently started system images
- put memory for inactive or lightly loaded system images into lower power states including turning it off entirely.

2 Characteristics of Memory Management in Modern Hypervisors

In order to understand the use of reverse mapping and how it applies to memory power-management and fault-isolation, it is helpful to standardize some terminology and describe some of the important features of memory management in modern hypervisors. Since different implementations of hypervisors use somewhat different memory management techniques, this document selects a particular hypervisor, VMware ESX [7], that has a good, public description of its memory management given in a recent paper by Waldspurger [8]. To avoid overlapping and confusing terminology, this document also provides a fixed set of terms used subsequently along with some common synonyms for each.

2.1 Terminology

The definitions here roughly follow the ones that Waldspurger uses in [8]. Much of the terminology there is taken from the earlier papers on Disco [2], [3].

- hypervisor: The *hypervisor* is the lowest layer of systems software that operates directly on the hardware and supports the execution of multiple operating systems in isolation on a single machine. The hypervisor is also called the *virtual machine monitor* in the literature.

- virtual machine (VM): A *virtual machine* is an isolated environment created by the hypervisor that is sufficiently like the hardware that an operating system can run inside it with little or no modification. The terms *partition*, *domain* and *system image* are sometimes used instead of *virtual machine*.
- guest system: A *guest system* is an operating system and the software running on it executing inside a virtual machine.
- virtual address: A *virtual address* is an address generated by a program prior to translation by a guest operating system.
- physical address: A *physical address* is the result of address translation by a guest operating system and represents what would be the actual address if the guest operating system were running on real hardware. However, here it represents an address within the address space of the virtual machine.
- machine address: A *machine address* is a true hardware address that the software can pass on the bus to memory. It is the result of the final translation of the physical address by the hypervisor.
- logical page: A *logical page* is an operating system that is part of an address space and is not necessarily tied to any particular physical unit of memory. In this context, a logical page is associated with a guest operating system.
- physical page: A *physical page* is a page of the memory of the virtual machine.
- machine page: A *machine page* is a page-sized unit of hardware memory. The hypervisor maps the physical pages of the virtual machines to machine pages or provides an invalid mapping indicating that the page is not present in memory.
- virtual page number (VPN): A *virtual page number* is that portion of the virtual address that determines which page of virtual memory the address references. With 4096-byte pages, it is, for example, all but the bottom 12 bits of the virtual address.
- physical page number (PPN): A *physical page number* is the result of a guest operating system translating a VPN using its translation tables. It gives the number of the physical page being referenced in the virtual machine's physical address space.
- machine page number (MPN): A *machine page number* is the result of the hypervisor's translation of a PPN to address a machine page in the hardware memory. It is worth noting that underneath this address, the memory controller may do some additional mapping of addresses before converting the address to the signals required to address the cells of DRAM.

2.2 Memory Management in Modern Hypervisors

Memory management for modern hypervisors must support operating systems that make use of address translation, paging and memory overcommit and which assume that they control the details of the translations and the location of logical pages. At the same time, to be of value to customers, they must offer features such as higher utilization and smoother resource re-allocation than one can get by running operating system images directly on the underlying hardware. This subsection recounts Waldspurger's description of the basic mechanisms for translation and memory control used in VMware ESX running on x86-based machines along with a few comments on how it changes in an environment that uses software-managed

TLBs and does not define a hardware-page-table format.

2.2.1 Basic Translation Mechanism

ESX gives each virtual machine a zero-based set of pages that the guest operating system treats as its physical memory. Although not explicitly mentioned, since most x86-based operating systems of interest (generally, considered to be limited to members of the Windows, Linux and BSD families) assume either a completely contiguous range of addresses or a small number of distinct memory regions, the assignment of pages to virtual machines probably matches the assumed architecture.

To provide each virtual machine with its own physical address space, ESX must provide an additional level of translation. To do so, ESX uses a *pmap* data structure for each virtual machine that maps the physical addresses or physical page numbers to their corresponding machine addresses or machine page numbers. Although the *pmap* effectively defines a virtual machine's physical address space, the *pmap* is not directly usable as the virtual machine's page tables since it does not map from virtual to machine addresses and because it is not in the proper, hardware-defined format. To avoid having to trap all references, ESX constructs a *shadow page table* for each virtual machine, to which the x86 CR3 register points when the virtual machine is running. This table is in the hardware-defined format and directly maps from virtual to machine addresses. Figure 1 is a simplified representation of these structures.

For architectures that use software-managed TLBs, there is no need for the shadow page tables except as a cache to speed up TLB reload.

2.2.2 Memory Control and Minimization Techniques

Modern hypervisors like ESX such as often make aggressive use of memory sharing techniques beneath the system images to reduce the overall load on memory and to increase the capacity of the hardware. As described by Waldspurger, these techniques include

- memory overcommit
- ballooning
- page-content hashing.

Not all of the techniques described by Waldspurger are mentioned here since not all of them affect the introduction of reverse mapping and its use within a hypervisor.

Ballooning creates artificial memory pressure within the executing system images, causing the operating systems running in them to discard or page-out pages and release page frames. These frames are released to the program creating the memory pressure, which is a special driver under the control of the hypervisor. Once it acquires the page frames, it notifies the hypervisor which steals them. Page-content hashing is a form of blind sharing between operating system images. Since many of the pages in each system image have the same contents such as shared library text pages for pages of the same library, the hypervisor can

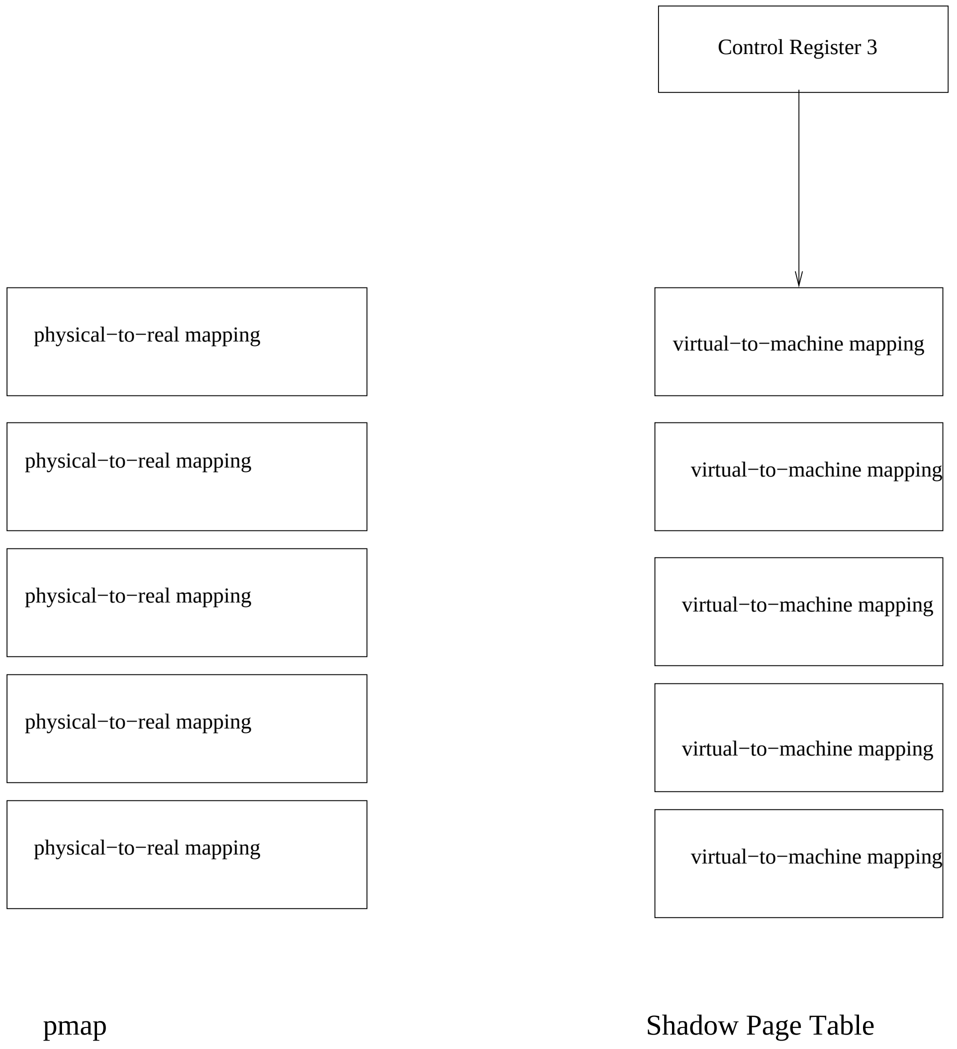

Figure 1: The ESX pmap and shadow page table structures for a single, executing virtual machine

share these pages between system images without the support of the operating system images. If comparison indicates that two pages have the same content, the hypervisor assigns them a single page frame with read-only hardware permissions, so that any changes cause a hypervisor-handled fault whose processing undoes the sharing operation.

2.3 Summary

The net effects of full virtualization and the memory sharing and reduction techniques of modern hypervisors are that there is no stable, straightforward relationship between a system image and the page frames that it uses and that a system image cannot unilaterally manage memory for fault isolation or power. This motivates the hypervisor features described in the rest of this document.

3 Reverse Mapping in a Hypervisor

Reverse mapping is a technique often used in operating systems to enable them to track efficiently which address spaces use which physical pages. Such reverse-mapping techniques are not new, and many operating systems either implement or plan to implement them. For example, there is a reverse-mapping patch to Linux [6] that provides a reverse paging from page frames to the Linux logical pages. A reverse mapping is simply a data structure or a set of data structures that maps from the physical to the logical pages in an operating system. In the case of a hypervisor, it is a mapping from the page frames to the pages of the virtual machines. Since many page frames are shared across multiple virtual machines, this mapping is one-to-many. Its primary advantage is that it makes it easy to go from knowledge about a page frame to the set of pages and, thus, the set of virtual machines that are using the page frame. Figure 2 shows the general relationship of the reverse mapping to the standard memory mapping tables maintained in an ESX-like implementation.

3.1 Requirements and Assumptions

Although well-known and often used in system implementations, the specific reverse-mapping techniques proposed here take advantage of the unique features of the hypervisor to reduce the overhead associated with reverse mapping. The reverse mapping used must exhibit the following characteristics to be suitable for a hypervior environment.

- The reverse mapping needs to be very space-efficient to reduce the memory overhead of maintaining it.
- It must have a low maintenance overhead, so that when entries change it does cost very time in terms of processing time to update the reverse mapping.
- The reverse mapping should take advantage of the relative stability of the operating system images. Partitions come and go less frequently than operating system processes.

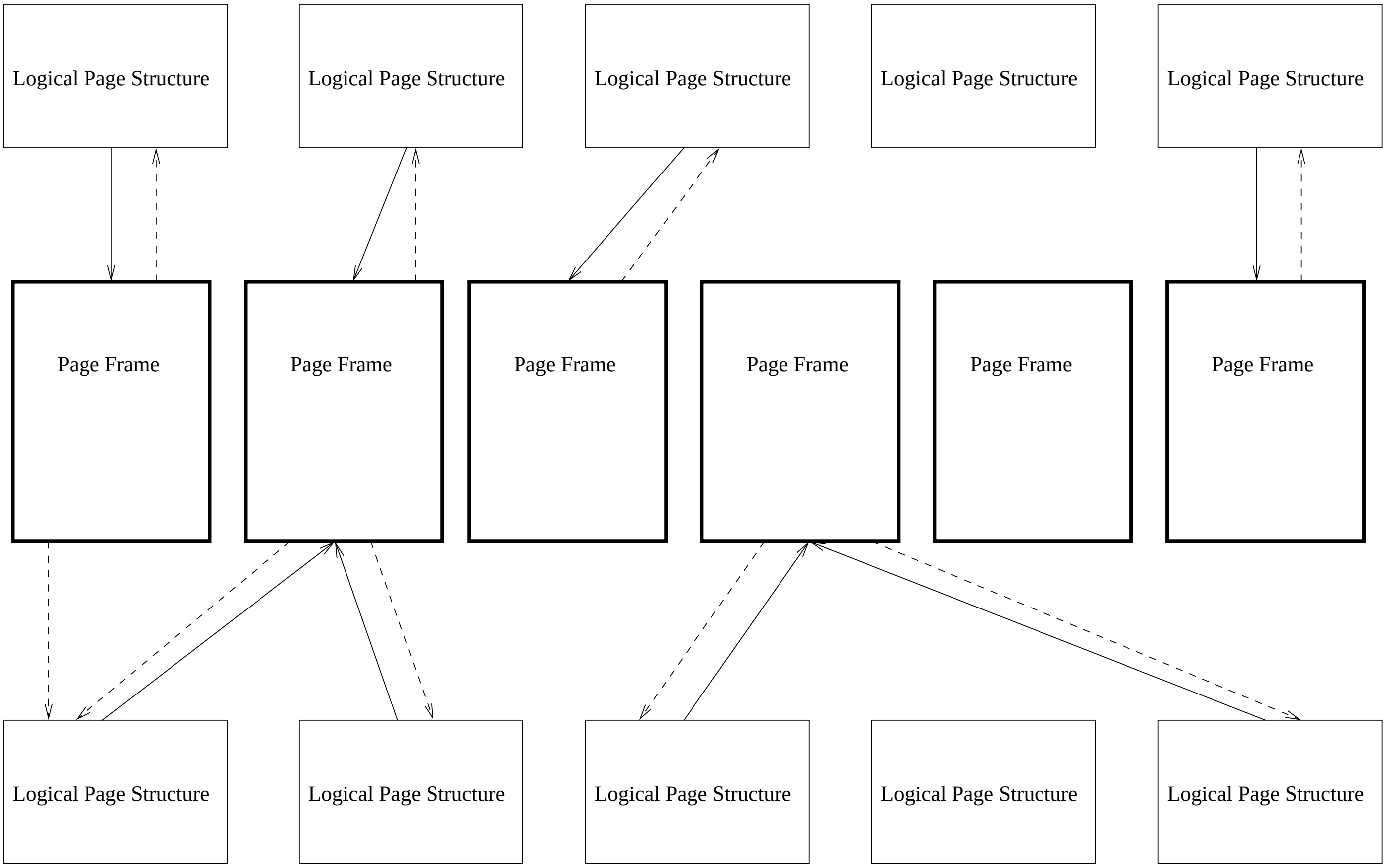

Figure 2: The relationship between standard and reverse memory maps in an ESX-like hypervisor

Despite the fact that many reverse-mapping schemes are possible, the one proposed here is optimized for a particular type of hypervisor environment in order to allow for the discussion of some of its details. The proposed design makes the following assumptions.

- The hypervisor does not necessarily allocate the pages in any particular fashion. In particular, in a running system, there is no reason to believe that it allocates runs of machine pages to the same virtual machine.
- The hypervisor does not use super-pages to map physical pages to machine pages.
- From the perspective of the reverse mapping, the discovery of shared machine pages through the use of content hashing, for example, further randomizes the usage.
- Even with aggressive sharing techniques, there are a very large number of machine pages that are private to a particular virtual machine. It is, therefore, worthwhile to consider the private machine pages as a special case.
- Data structure compactness is more important than search or update speed.
- From the virtual machine identifier and a physical page number, it is easy to find the address of the appropriate data structure for the physical page from other data structures or algorithms. In particular, there is no need to store address structure addresses for the virtual machine machine and the physical page in the reverse mapping.

3.2 Reverse Mapping Design

The reverse mapping uses a single, contiguous table indexed by MPN, referred to here as the *rtable*. The *rtable* entries have one of two formats, depending on the value of the first of three flag bigs. The first format, used for private, unshared machine pages starts with three flag bits – the format bit that indicates whether or not the machine page is private, a bit that indicates whether or not the remaining contents of the entry are valid and a bit that indicates whether or not the contents of the machine are discardable. If the entry is a private machine page and is valid, the virtual machine identifier of the owner of the machine page and the PPN of the corresponding physical page in the virtual machine follow the flag bits. The second format, which the hypervisor uses for shared pages, includes the initial flag bit, which, in this case, indicates that the entry is a shared machine page entry, and the address of the first array of reverse mapping pointers: each array is called an *rarray*. Although it may not be true in every case, in most implementations, the use of virtual machine identifiers and physical page numbers reduces the total number of bits required in the representation over the number needed if pointers were used instead. This leaves room for the flag bits without increasing the size of the data structures or making the alignment awkward. Each *rarray* consists of a set of entries in a format similar to the private entries in the base *rtable*. However, in the *rarray*, the first flag bit is not used. The second flag bit indicates whether or not the the entry is valid: this is necessary since the deallocation of mappings leaves unused entries in the *rarray*. As before, the third flag bit indicates whether or not the page is discardable. Since the *rarray*s are of fixed size, the reverse mapping may require multiple *rarray*s for a single machine page if it is heavily shared. As a result, the last entry of each *rarray* is a pointer to another *rarray* for the same machine page if it is marked as valid. Setting the size of the *rarray*s

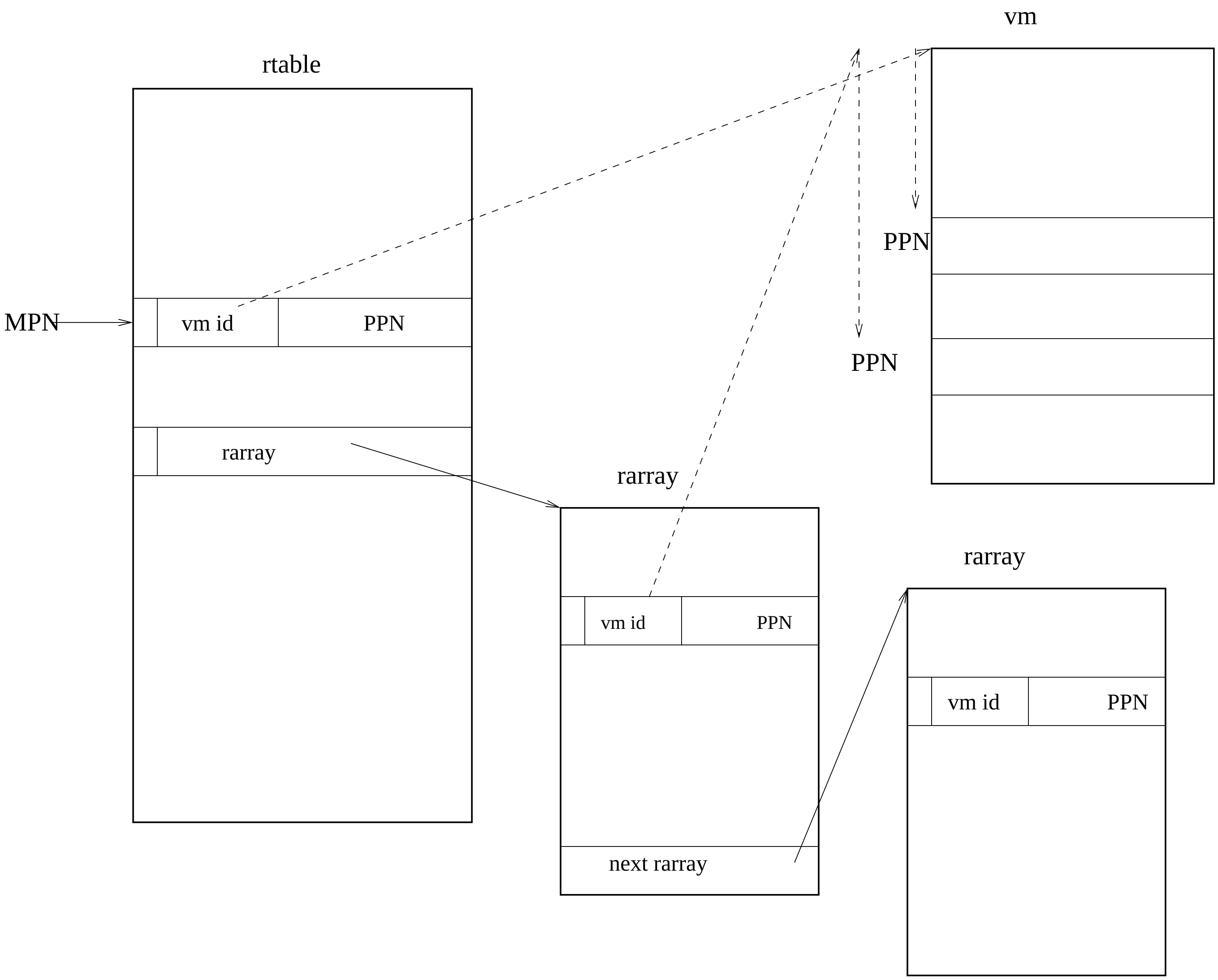

Figure 3: The *rtable* and the *rarray*s

a priori is difficult, and the optimal setting depends on the expected number of virtual machines sharing a machine page. This design allows the developer to fix the number at compilation or make it a parameter to the hypervisor. Figure 3 is a conceptual representation of these data structures.

The reverse mapping table proposed above explicitly allows a single machine page to map to the physical pages of multiple virtual machines since with techniques like content hashing, different virtual machines may share the same physical memory if the contents are identical. It also tracks whether or not the contents are discardable due to the presence of a copy on disk managed by the hypervisor or due to an indication from the operating systems involved. In this case, the hypervisor can quietly discard the contents of the memory and either re-allocate physical memory in another location or reduce the memory sizes of the affected partitions. Using the MPN to index the *rtable* and virtual machine ids and PPNs reduces the amount of space needed.

3.3 Search

The search procedure for the reverse mapping is very simple. The MPN indexes the *rarray* with a prior check to ensure that it is in range. If the selected *rarray* entry indicates a private machine page, the algorithm uses the virtual machine id and PPN to determine the address of the data structures describing the physical page. Otherwise, the address in the *rtable* entry is that of the first *rarray* for the machine page. The search then scans each *rarray* in linear order, accepting the valid entries and ignoring the invalid ones. For the last entry, rather than treating it as another reverse-mapping entry, the search checks the valid bit. If the bit is invalid, the search ends. Otherwise, the entry is a pointer to the next *rarray* for the machine page. Since the goal of the reverse-mapping search is to find *all* of the physical pages affected by some event occurring in the machine page, a linear search of each *rarray* is a perfectly acceptable method of access. Although search requires some form of locking or serialization, its details are very specific to the particular hypervisor implementation and are beyond the scope of this document.

3.4 Addition of a New Reverse-Mapping Entry

Upon any change in the mapping from physical to machine pages, the hypervisor must update the reverse mapping, and there are a number of different cases to consider depending on the current state of the reverse mapping and the change the hypervisor is making. The descriptions given here assume that other code ensures that all accesses and updates are properly serialized.

If the hypervisor is adding a new reverse mapping of a machine page to a virtual machine and physical page, it first determines if there is any entry in the machine page's slot in the *rtable*. If not, it creates a private *rtable* entry for the machine page using the virtual machine identifier and physical-page number mapping to the machine page. It marks the entry as private and sets the discardable bit based on other information about the use of the page. If there is no indication that the page is discardable, the hypervisor marks it as non-discardable.

If there is a private entry in the *rtable* slot for the machine page, this is the first use of the machine page as shared. The hypervisor allocates an *rarray* for the machine page, copies the reverse mapping from the *rtable* slot to the first entry in the *rarray* and converts the *rtable* slot to a pointer to the *rarray*. Finally, it marks the last entry in the *rarray* to indicate that there are no additional *rarray*s.

If the *rtable* entry for the machine page indicates that it is already shared, the hypervisor searches the first *rarray* for an invalid entry that it can use for the new mapping. If one is found, it marks the entry as valid and sets the virtual machine identifier and physical page number in it. As always, the discardable bit is set based on other information that the hypervisor may have about how the virtual machine uses the page. If there are no available entries in the first *rarray*, the hypervisor checks the final entry. If it is valid, it goes to the next *rarray* and repeats the search. This process continues until either it finds an invalid entry or it finds the final *rarray* entry in the current *rarray* marked as invalid. In this situation, the hypervisor allocates a new *rarray*, uses the first entry in the newly allocated *rarray* for the new reverse mapping and updates the last entry in the previous *rarray* to point to the next *rarray*. Finally, it marks the last entry in the newly allocated *rarray* as invalid to indicate that there are no more *rarray*s for the machine page.

3.5 Removal of a Reverse-Mapping Entry

As with the addition of a reverse-mapping entry, the hypervisor handles the removal of one on a case-by-case basis. It begins by locating the machine page's slot in the *rtable*. Barring a problem with the hypervisor implementation, the entry must be valid, and it has to match the mapping being removed. If the *rtable* entry is marked as private, the hypervisor marks it as invalid. Otherwise, the hypervisor searches the *rarray*s for the entry to remove. It follows the pointer to the first *rarray* and searches the entries, looking for a valid entry that matches the virtual machine identifier and physical page number being removed. During the search it counts the number of valid entries in the *rarray*. If it finds a matching entry, it marks it as invalid but continues the search. The hypervisor does this to determine if the machine page is now private as opposed to shared. When the hypervisor reaches the final entry of the *rarray*, if there are no remaining valid entries in the *rarray*, the pointer must be valid since the machine page was originally shared. The hypervisor deallocates the now completely invalid *rarray* and chains the next *rarray* to the *rtable*.

If there is no matching valid *rarray* entry in the first *rarray*, the hypervisor moves on to the next *rarray* and repeats the search on it, continuing through the *rarray*s until it finds the matching entry. If it has already found the match, it continues the search, deallocating a newly empty *rarray* if necessary. When it finally gets to the end of the *rarray* chain, indicated by an invalid final, pointer entry, the hypervisor compares the count of valid entries to 1. If it is equal, it copies valid entry to the *rtable* and marks it as private. It then deallocates the surviving *rarray*. Otherwise, the machine page is still shared, and the surviving *rarray*s all contain at least one valid entry.

3.6 Super-pages

Although there is an explicit assumption that the hypervisor, due to its aggressive virtualization does not use super-pages, one can modify the reverse mapping to accomodate them. With a mix of super-pages and regular pages, the reverse mapping consists of two tables, one for the machine pages that back the hypervisor's super-pages and one for the hypervisor's regular pages. The super-pages reverse mapping maps an appropriately sized group of machine pages to the super-page but, otherwise, has the same format as the ordinary reverse mapping.

4 Uses of the Reverse Mapping

For the purposes of this document, the hypervisor uses the reverse mapping described above for two primary purposes. The first is to support power management, especially cooperative power management of the memory. The second is to isolate memory failures to particular virtual machines to reduce their overall impact on the users of the system.

For the purposes of the following discussion, the schemes described below assume that memory is physically manageable for power and fault isolation by groups of contigous machine pages. In the best and limiting case, the group size is a single machine page, but the physical design of the memory generally makes it significantly larger, especially for power management. The sizes of the groups used for power management and fault isolation are not necessarily the same.

4.1 Memory Power Management by the Hypervisor

With the information provided by the reverse mapping, the hypervisor can implement memory power-management techniques that take the behavior and activity levels of the system images running within the virtual machines into account. This subsection considers two specific schemes that depend critically on the use of the reverse mapping introduced above.

4.1.1 Reactive Memory Power Management with Notification

Reactive memory power management with notification puts the responsibility for the power state of the memory primarily on the memory controller with some help from the hypervisor. This scheme uses three low-power states – power-down, self-refresh and destructive-off, where destructive-off turns off the memory entirely, destroying its contents. The memory attempts to remain in the lowest power state by first requesting that the hypervisor scan the *rtable* and determine which machine pages are used. The hypervisor returns a list of the unused pages to the memory hardware. For any unused machine pages that are grouped into power-manageable units, reactive memory power management puts the memory into the destructive-off state to minimize power. Whenever the hypervisor allocates a page from a group in the destructive-off state, it must put the remaining machine pages in the group into a higher power state, generally self-refresh. The hypervisor must also update all of its data structures including the reverse mapping to reflect the new allocation.

Although it is possible to have the memory-controller hardware do the scan, the complexity of the data structures involved and the need for flexibility when changing the hypervisor suggest that the hypervisor should do it instead. This also avoids a number of serialization questions that arise when the hardware and the hypervisor share a data structure.

In addition, in order to save additional power, the memory controllers notify the hypervisor of page groups not recently accessed that they have put into a lower power state. The hypervisor uses the reverse mapping to notify the guest systems that the memory is in a lower power state and, thus, has a higher-than-normal latency. It may also use the notification to request that the hypervisor either check for discardable pages or page out pages. This is helpful if there only a few pages in use in a power-manageable group. The memory hardware can request their removal so that it can turn off the group. To a first-order approximation, neglecting the performance cost, this is beneficial if and only if there is a net conservation of energy. This, in turn, is the case if the following inequality holds.

$$P_{sf} \times E(duration) > P_{removal} \times t_{removal}$$

Here P_{sf} is the self-refresh power of the memory group, $E(duration)$ is the expected value of the duration that the memory group is off and $P_{removal}$ and $t_{removal}$ are power and time required to remove all of the pages that the system from powering off the memory group. In general, the value of $E(duration)$ is unknown, but one can approximate it adaptively by collecting a history of the durations and averaging them.

4.1.2 Hypervisor-Directed Memory Power Management

In the hypervisor-directed memory power management scheme, the hypervisor monitors the usage of machine memory on a page-by-page basis and uses the reverse mapping table to correlate that usage with the virtual machines and physical pages using the memory. Based on some function of the usage and the relative importance of the guest systems running in the virtual machines using a particular piece of memory, the hypervisor selects a power state for the memory. The goal is to optimize, for a chosen objective function, the power and performance of the collection of virtual machines running on the hypervisor. This scheme also provides for an optional notification to the guest operating systems affected that they have less or lower performance memory than previously allocated to them.

4.2 Memory Fault Isolation and Handling

One of the worst scenarios that can occur in a server-consolidation environment is a memory failure that brings down all of the guest systems running on top of the hypervisor. If this happens, rather than losing a small percentage of the capacity, as would occur in a cluster of smaller machines, the user loses all of the guest systems or, effectively, the entire capacity. Avoiding such failures is critically important if server consolidation is to be a commercial success.

In the case of a memory failure, the hardware detects the failure and notifies the hypervisor through a standard, hardware-defined mechanism such as a machine check. The reporting of the failure is in terms of the machine address of the affected memory. Using the machine address, the hypervisor computes the machine page number. It then uses the MPN as an index into the *rtable* to find the affected virtual machines and physical page numbers. Using that information, it can either terminate all of the affected virtual machines, leaving all of the other virtual machines running, or it can present a virtual machine check to the guest operating systems if they have machine-check-handling logic. Once each affected virtual machine either is terminated or has handled the machine check, the hypervisor can instruct the memory hardware to isolate the failing memory. The hypervisor then marks the memory, perhaps using an additional feature of the *rtable* to indicate that it can no longer be used.

5 Related Work and Prior Art

Hypervisors have a long history dating back to the 1960s. One of the very first hypervisors was IBM's CP-67 developed to support multiple virtual machines, primarily for testing and interactive computing purposes, running either a specialized monitor program or a copy of IBM's OS/360. IBM has marketed the successors of CP-67 continously since the early 1970s, and the current generation is known as zVM. It is heavily marketed today as a server consolidation solution that allows a very large number of Linux guests to run on a single IBM zSeries machine.

During the 1990s, researchers at Stanford developed Disco [2] and Cellular Disco [3] in support of their work on the Flash [4] project. They believed that it was difficult or impossible to scale the existing standard operating system for the processor that they were using to their unique hardware, and they used their hypervisors to allow them to run the standard operating on their research prototype.

Using some of the technology developed in the Disco work, a number of the Stanford researchers formed VMware. VMware has introduced two fundamentally different types of hypervisors, a hosted version that executes on top of an underlying, standard operating system such as Linux, and ESX, which is a traditional hypervisor in the spirit of zVM.

Finally, a group from the Cambridge University Computing Laboratory has released a hypervisor called *Xen* [1]. Xen supports what the Cambridge team calls *paravirtualization* in that it requires small modifications to the source of every operating system that it runs.

None of these efforts are particularly concerned about either fault isolation or memory power, and none of them have published anything indicating that they use a reverse-memory-mapping technique to support either of these features. However, it is possible to view the reverse-mapping technique, when applied to memory power management, as an implementation of the selective-memory-powering or file-cache-pruning techniques described in [5].

6 Conclusion

By adding a specialized, compact reverse mapping, it is possible for a hypervisor to track accurately the usage of the page frames of physical memory by the operating system images that it runs. With this information, the hypervisor can, based on the apparent memory demands of the operating-system images that it is running, determine the most efficient power state for each page frame of physical memory. In addition, should the hardware detect a memory failure in a particular page frame, the reverse mapping allows the hypervisor to determine the affected virtual machines. This, in turn, permits it to minimize the number of operating system images that are either notified of the memory failure or terminated.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[2] Edouard Bugnion, Scott Devine, and Mendel Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4), 1997.

[3] Kingshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel Rosenblum. Cellular Disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 18(3):229–262, 2000.

[4] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Stanford FLASH multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, April 1994.

[5] Freeman Rawson. Selectively powering portions of system memory in a network server to conserve energy. United States Patent Application US20030061448A1.

[6] Rik van Riel. Towards an O(1) VM. In *Proceedings of the 2003 Ottawa Linux Symposium*, 2003.

[7] VMware, Inc. VMware ESX Server 1.5.

[8] Carl A. Waldspurger. Memory resource management in VMware ESX server. In *Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, 2002.