Research Disclosure, July 2007 ***www.researchdisclosure.com***

Improving Operating System Crash Dump Performance on a Virtual Machine

A method for improving the performance of operating system (OS) crash dumps in a virtualized computing environment is disclosed. This method involves copying OS kernel memory to a shared virtual memory region.

At times operating systems fail and it is necessary to save a copy of OS memory to enable failure root cause analysis. The most common existing technique for saving a copy of OS memory is to use software drivers to write the contents of OS memory to disk. Existing techniques to improve performance include copying the contents of OS memory to a reserved area of RAM, compressing the contents of OS memory before copying it to disk or RAM, copying OS memory in parallel to several disk devices, and copying only selected areas of OS memory to disk or RAM.

Copying OS memory to disk is slow, even with existing performance improvement techniques. Copying OS memory to RAM requires sufficient free RAM to contain OS memory, plus non-destructive reboot functionality which is not available on all systems.

Virtualization is a method of emulating a computer system hardware entity as a software entity. Commercial solutions exist which virtualize all hardware resources of a computer system such as CPU, memory, I/O controllers etc. as software entities. In the context of this disclosure, a VIRTUAL MACHINE provides an environment which permits an unmodified instance of an OS to execute and interact with hardware resources abstracted as software entities. The vitualized environment is provided on the VIRTUAL MACHINE by an operating system known as the HOST OS, whereas an operating system that runs in the virtualized environment on the VIRTUAL MACHINE is known as a GUEST OS. The HOST OS views a GUEST OS as just another process.

A GUEST OS is an instance of a specific application, known as VMAPP for the purposes of this disclosure, that knows how to create the virtual environment. Physical memory is allocated for the GUEST OS by allocating a range of virtual memory in the VMAPP process. VMAPP loads a GUEST OS instance by reading a configuration file and loading the contents of the OS image into the range of allocated memory. VMAPP then passes control to the GUEST OS. A method of communication between the VMAPP process and the GUEST OS must exist. For the purposes of this disclosure the method of communication is assumed to be a defined set of firmware calls available to the GUEST OS that pass control to the VMAPP process for the duration of the call (that is, until the call exits), but the specific method of communication could be implemented in various ways.

The typical crash dump process for a GUEST OS is the same as for an OS in a non-virtualized environment. This disclosure proposes a new model for the crash dump of a GUEST OS in which the GUEST OS memory is copied to a shared region in virtual memory instead of disk or RAM. The crash dump model proposed exhibits the following features and benefits:

- GUEST OS memory is copied to a shared region in virtual memory.
- The target region being contained in virtual memory obviates the need for it to be pre-allocated or reserved at boot.
- In-memory root cause analysis: analysis tools are able to access the failed GUEST OS kernel memory directly, thus obviating the need to copy the memory to disk.
- Significantly improved performance: copying the GUEST OS memory to virtual memory is significantly faster than copying to disk. If necessary the memory region can be copied to disk by a background process after the GUEST OS reboot.

- Concurrent GUEST OS reboot and crash dump. In the typical model, crash dump and reboot occur serially by necessity, whereas the proposed model allows reboot concurrent with crash dump by allowing the GUEST OS to boot into a memory region different from that used for crash dump.

Several approaches to copying GUEST OS kernel memory to a shared memory region are proposed:

- via the VMAPP process.

  With this approach the normal crash dump code path of the GUEST OS is modified to communicate with the VMAPP process. After experiencing a catastrophic failure the GUEST OS enters the crash dump code path and transfers crash dump metadata to the VMAPP process. The metadata includes, for a UNIX GUEST OS for example, a timestamp, the panic string, page table information, crash dump size, etc. Upon receipt of the crash dump metadata from the GUEST OS, the VMAPP process creates a shared memory region equal to the size of the GUEST OS crash dump. The VMAPP process then initiates a memory-to-memory copy of the GUEST OS kernel memory from the virtual memory region allocated (previously) to the GUEST OS to the newly allocated crash dump region. The shared memory region being allocated dynamically at GUEST OS crash dump-time distinguishes this method from existing methods which require the memory region to be pre-allocated. Ideally there should be sufficient physical memory available to allocate to the crash dump shared memory region, otherwise paging activity to secondary storage would result and some of the benefit of writing to memory rather than disk would be forfeited.

  A consequent benefit of this approach is that the dump is written out in a safe environment (the VMAPP process) and not, as is typically the case, a compromised environment (the failed GUEST OS).

- by hyper-call.

  A hyper-call on the GUEST OS is analogous to a UNIX system call except that it transfers control from the GUEST OS to the VMAPP process, allowing the GUEST OS to request service from the VMAPP process. A single hyper-call is implemented based on the UNIX write() system call. The hyper-call passes a buffer containing the contents of GUEST OS kernel memory to be copied for the crash dump.

  The primary motivation for this approach is to reuse existing dump solutions without having to code them explicitly in the VMAPP process. Existing crash dump solutions such as using multiple CPUs to write the contents of OS kernel memory in parallel to several dump devices can be made to easily work in this solution: the VMAPP process creates multiple shared memory regions in response to a request from a GUEST OS to perform a crash dump, and the GUEST OS is instrumented to contain a set of crash dump drivers capable of copying the contents of kernel memory in parallel to multiple shared memory regions.

  This approach allows significant code re-use (or significantly less change to existing crash dump code), and though it could be slower due to context switching overhead than performing the crash dump directly via the VMAPP process (as described above) it would still be substantially faster than existing methods which write the contents of kernel memory to disk.

- to added “physical” memory

  In this approach, once the GUEST OS enters the crash dump code path, it requests the VMAPP process allocate it additional “physical” memory. It is important to note that no physical memory needs to be added to the system: additional physical memory can be allocated to the GUEST OS by the VMAPP process allocating one or more virtual memory regions and communicating the physical address(es) of the region(s) to the GUEST OS. The GUEST OS then copies OS kernel memory to the newly allocated region(s) in physical mode (using multiple CPUs in parallel if multiple target regions are allocated).

Although this method increases implementation complexity due to the allocation of additional “physical” memory for the GUEST OS, it avoids the context switching overhead inherent in the hyper-call approach described above. For optimal performance the additional memory region(s) should be locked in memory.

A variant approach to performing a crash dump for a GUEST OS in a virtualized environment is to leave the OS kernel memory in place and remap the shared virtual memory region created for the crash dump to the existing GUEST OS kernel (physical) memory pages. This remapping would result in multiple virtual addresses mapping to the same physical page; this is known as “virtual aliasing” and is available in most operating systems.

In order to implement this approach a function similar to the mremap() system call found in LINUX/BSD is required. This function would differ from the mremap() system call in that the virtual memory region would be resized from both ends rather than only from the upper memory range as mremap() does. Resizing the virtual memory region from both ends is required so that crash dump metadata can be inserted as necessary at both the beginning and end of the virtual memory region.

This approach would be extremely fast since it involves only the remapping of virtual memory addresses rather than copying of the contents of memory, but it does have some drawbacks. For example, it does not allow:

- compressing the contents of the GUEST OS kernel memory, but since analysis tools would access the crash dump in situ this shouldn’t be a problem. Compression is a means by which disk write times and disk space consumed by the crash dump can be reduced, so could be performed at some later stage if it was considered necessary to copy the crash dump to disk.

- dumping selected pages from the GUEST OS kernel memory, so a full crash dump is effectively created in situ. As mentioned above, this is only a problem if the crash dump is required to be copied to disk, and then it could be compressed or selectively written. This could be overcome by coding the GUEST OS to pass a list of selected pages to the VMAPP process to include in the crash dump. The list could be an array of tuples, each tuple containing the GUEST OS virtual memory address and the number of pages to be copied (from the starting address), and the VMAPP process would determine the physical addresses and remap the pages as necessary. Alternatively, an OS driver could be written to insert translations into the shared memory region.

There is no existing solution that describes an OS crash dump to virtual memory in a virtualized environment as described in this disclosure. It is noted that there have been solutions that describe OS crash dump to physical memory: these can be found in open source implementations such as LINUX, or in proprietary systems such as TRU64 or AIX. These solutions, as well as dumping to physical memory rather than virtual memory, do not describe a mechanism to dump the OS memory while concurrently rebooting to a different region in memory as described by this disclosure.