RC24417 (W0711-094) November 14, 2007
Computer Science

IBM Research Report

On the Usability of Virtualization Technologies for Application with Stringent Requirements

Claris Castillo
North Carolina State University

Arun Iyengar, Amol Nayate
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

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

On the Usability of Virtualization Technologies for Applications with Stringent Requirements

Claris Castillo[1], Arun Iyengar[2], Amol Nayate[2]

Abstract:
Virtualization technology has brought about key benefits - *simplified management* by decoupling the virtual machine from the physical hardware it runs on and allowing virtual machines to be dynamically migrated away from bottlenecked or faulty machines toward newer or better suited ones, *better resource utilization* by allowing consolidating multiple virtual machines onto fewer physical machines to save power, and *strong isolation* by ensuring that one faulty virtual machine cannot affect the performance or health of another. In this paper, we discuss a fourth benefit yielded by virtualization technology – generating virtual machines checkpoints (snapshots) so that they can be quickly restarted when they crash – and discuss the cost/benefits trade-offs between various checkpointing techniques. We evaluate taking periodic checkpoints on *Market Matching*, a real-world application with stringent availability and performance requirements, and show that although existing checkpointing mechanisms can be beneficial for applications with simple workloads, they are not practical for use with applications with stringent availability requirements. Finally, we discuss bottlenecks that slow down the process of taking snapshots in current hypervisor implementations.

Introduction:
In the last five years virtualization technologies have experienced a significant grown in terms of development and deployment. We have witnessed their application in different fronts such as in large server infrastructures, Grids and virtual appliances. Despite all these advances the usability of virtualization in environments with high requirements, e.g. in financial applications, has been overlooked. This state of affairs has its roots in the fact that since virtualization emerged in response to the need of consolidating servers and easing management tasks, considering QoS requirements has been at best a secondary consideration. Typically, today's environments exhibit availability requirement values of 99.999%, response time requirement values in the range of tenths of milliseconds and zero tolerance to losses. To address these challenges system designers rely on the composition/superposition of multiple technologies which are orchestrated by means of sophisticated mechanisms. Given the benefits of virtualization in terms of simplifying and easing the allocation and management of resources, we believe that it has the potential to alleviate some of the challenges imposed by applications with stringent requirements.

In this work we investigate the usability of virtualization within such environments. As a first step we study the potential of existing virtualization tools/mechanisms in providing high availability. More specifically, we leverage the ability of creating snapshots of virtual machines for the purpose of providing backup capability in view of failures. In our study we consider an application with high availability, throughput and reliability

[1] North Carolina State University
[2] IBM TJ Watson Research Center

requirements - *Market Matching* - developed in house at IBM. The Market Matching application is a key component of the market exchange system and has been an application of interest for IBM. Currently this application is being used at the Chicago Market Exchange.

In this document we report some preliminary results of our investigation. The results indicate that although virtualization technologies have achieved remarkable success/adoption on many fronts such as the ones mentioned earlier in this report, in environments with more strict requirements virtualization still needs significant refinement and development. We recommend mechanisms to adapt existing virtualization tools to accommodate the specific requirements of the application under study and define future directions of investigation/research.

Application targeted: Market Matching

The market matching application is the core component of a market exchange and is responsible for managing trading in a set of financial instruments. The market matching engine maintains a set of order books with a separate order book for each financial instrument traded, and processes arriving orders to buy and sell those instruments. Each request arriving at the market matching service consists of a mixture of new orders and cancellations and modifications of previous orders. Each simple order is a request to buy or sell a specific quantity of a specific instrument on behalf of a specific customer at a named price.

Market exchange systems impose several challenges to designers and administrators due to the stringent requirements associated with the exchange's business. Such systems typically require availability index values of 99.999% and average response times (i.e., end to end latency) of 50 milliseconds. Moreover, the demand volume is expected to triple in the next years. As a result of this fact, scalability has become a key design parameter towards guaranteeing a more efficient and effective market exchange business in the future.

Providing a complete solution capable of accommodating all these requirements has proven to be a challenging task. Consequently, most existing systems are built from the composition of multiple techniques orchestrated by means of sophisticated and complex mechanisms. For instance, parallelization, deployment of additional hardware, etc have become the *de facto* components in market matching systems currently deployed, with the high cost of ownership and maintenance widely justified by the financial nature of the application *per se*. Given the effectiveness of virtualization in providing server consolidation and simplified resource management we believe that this kind of application could highly benefit from virtualization technologies and its potential should be investigated.

Virtualization-- background

Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility. It allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine, which each virtual machine having its own set of virtual hardware (e.g., RAM, CPU, disk, network interface, etc.)

Since virtual machines are encapsulated into files, virtualization software has the ability to rapidly save, copy, and provision a virtual machine. Existing implementations leverage this feature to create snapshots of virtual machine. There are three different techniques to create snapshots of VMs, referred to as *cold*, *warm* and *hot* backups. Each has a different level of impact on the services running in the VMs and the way in which recovery actions are executed. A summary of their main differences is presented in Table 1.

	Colds	Warm	Hot
Service Impact	Complete shutdown of application and server	Momentary suspension of processing	Imperceptible
Snapshot File System Contents	Closed and unmounted	Point in time cached I/O completely resumed	IO flushed up to point in time. File system unaware
Recovery Process	Cold boot guest OS	Guest OS resumes from point in time	Guest OS performs system crash recovery

Table 1 : Three snapshot techniques, cold, warm and hot snapshot.

In order to provide high availability to the market matching application we propose to run each market matching engine within a VM and create snapshots of the VM in a periodic basis. On failure a new VM is then bootstrapped from the latest available snapshot. Given the application's requirements the process responsible of creating the snapshot must incur minimum overhead and be imperceptible to the clients. Note that an intrusive VM snapshot scheme could delay incoming traffic until the VM undergoing the snapshot process has fully recovered and is ready to handle requests.

As demonstrated in Table 1, *hot backup* is the most suitable technique within the context considered given its imperceptible impact to the service; i.e., the Market Matching engine continues running while the snapshot is being created. The warm snapshot technique on the other hand is less suitable since it suspends the market matching while creating the snapshot. The cold snapshot technique requires a complete shutdown of the VM and therefore is not discussed in our study. Below we give a description of the warm and hot snapshot techniques.

Warm Snapshot

A warm snapshot copy makes use of the capability to suspend a guest OS running in a VM. When the suspend action is performed, the program counter for the VM CPU is stopped, all active memory is saved to a state file inside the file system of the VM where the boot disk resides, and the VM is paused. At this point in time, a Snapshot copy can be taken of the entire VM, including the memory contents file and all logical units with the associated active file systems. In that copy, the machine and all data will be frozen at the point in processing time when the suspend operation was completed.

Since all of the contents of memory, the active virtual CPU registers, and the associated state are written to a file, the instruction pointer, as well as all memory data structures

such as the I/O queue and any applications that were running are included in this information.

When the Snapshot action is complete, the VM can be restarted and will resume at the exact point where the suspend action was initiated. The guest OS will continue processing, including handling I/O activity and any in-memory transactions that were loaded from the state file. Applications and server processes will resume processing from the same point in time. The outward appearance is as if a “pause” button had been pressed for the duration of the snapshot activity. To any network client of the guest OS server, it will appear that there was a temporary interruption of network service. Our measurements show that this interruption lasts on the order of 30 to 120 seconds for moderately loaded servers.

Hot Snapshot

Virtual disks that contain the boot partition for a guest OS running in a VM should be configured as “Persistent” files inside of the file system of the VM. In that state, all writes that occur are immediately applied to the virtual disk to maintain a high level of consistency in the file system. Virtualization software (e.g., VMware) provides a facility to place a persistent virtual disk into a hot backup mode for the purposes of taking snapshot copies at the disk subsystem layer by adding a REDO log file. When the REDO log file mode is initiated, all outstanding writes are flushed to the virtual disk file and all future writes are written as log data into the REDO file. Much like putting a database into hot backup mode, the guest OS root disk or any other virtual disk is placed into hot backup mode with this operation.

Once the REDO log has been activated, it is safe to take a snapshot of the logical unit containing the file system of the VM. After the snapshot operation is complete, another command can be issued which will commit the REDO log transactions to the underlying virtual disk file. When the commit activity is complete, all log entries will have been applied and the REDO file will be removed. During this operation, a marginal slowdown of processing will occur, but all operations will continue to function. It is possible to stack a secondary REDO file behind primary REDO in order to smooth the application of log entries via staging.

Maintaining as short a time window as possible to perform the snapshot prevents excessive transactions from queuing in the REDO log, which in turn minimizes the external impact of the commit phase of the operation.

The snapshot copy is taken of a “running” process, but typically without any provision for dumping the contents of RAM in the manner that the warm snapshot copy maintains; thus, recovery of a hot Snapshot copy will appear to the guest OS as if it is cold booting after a total loss of power. Applications running during a hot snapshot operation will also require media or transaction recovery as appropriate for the software product.

The outward appearance during a hot snapshot operation is an imperceptible server slowdown. At worst, it will appear that network congestion or an overloaded CPU may be causing general server sluggishness. At best, there is no noticeable impact.

There are three kinds of hot backups: *full*, *differential* and *incremental* snapshots. A full snapshot includes all persistent data and machine state and takes time proportional to the size of the VM. A differential snapshot provides a faster mechanism for backing up VMs than a full backup since it only includes those blocks that have changed since the last full snapshot. An incremental backup is the fastest backup mechanism since it only includes those files that have changed since the most recent backup. The advantage of the fast incremental backups comes with a price, however; the process of restoring a VM potentially takes a long time since it involves reading the most recent full backup as well as every incremental backup made since then. In our experiments we favor the use of the differential backup since it offers a good compromise between the time it takes to create the snapshot and the restore time.

Conventional High Availability mechanisms

Below, we describe two techniques currently implemented in systems in production to guarantee high availability: *primary-backup* and *primary-primary* scheme. Both schemes rely on some sort of hardware replication to guarantee the availability of the market matching service upon failure; with the primary-primary being the successor of the primary-backup technique.

Primary-Backup

In the primary-backup scheme, every primary node is associated with a backup node. The backup node behaves as a follower, in that it receives the same set of requests as the primary. To do so, it subscribes to responses sent by the primary to request originators and extracts the order in which the primary executed those requests. It then applies these messages to its in-memory copy of the book and performs an “assertion” comparison of the result with the intercepted response generated by the primary.

On a failure of the primary node, the primary node is locked out of the log and the backup node effectively becomes the new primary node. To do so, the backup-node catches up to the log tail, processing all requests which have been completed and logged in the order of arrival at the primary.

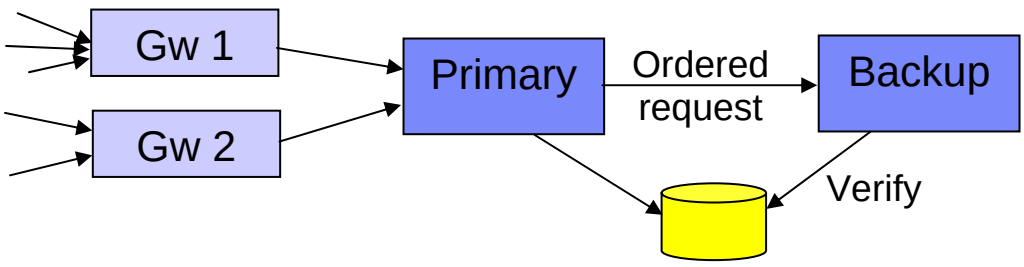


Figure 1: Primary-Backup Scheme.

Primary-Primary

In the Primary-Primary scheme, there are two primary nodes functioning continuously. Different from the Primary-Backup, both nodes are an exact copy of each other; both primary nodes subscribe to the same set of messages and execute all orders in the same exact number. To guarantee the order requirement, a *coupling facility* determines a total order among all the orders as they arrive and informs the nodes of this order before they can execute them.

On a failure, the failing node is locked out of the system for maintenance. The second primary node continues handling existing and incoming requests.

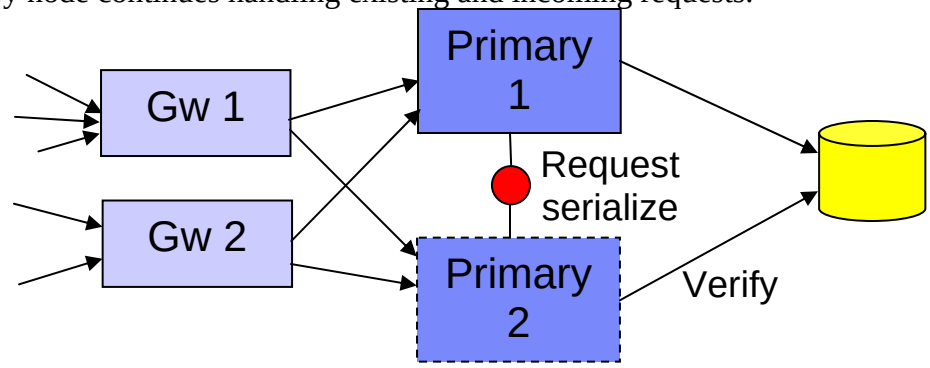


Figure 2 The Primary-Primary scheme

Experimental Setup

Figure 3 illustrates the test bed used in our study. We set up three servers: one VM server running the market matching engine (VM MM in figure), one VM server serving the role of a gateway responsible of forwarding incoming requests (VM GW in figure) and one server responsible for creating snapshots of the VM hosting the Market Matching engine (MM). The physical node hosting the MM VM runs VMware.

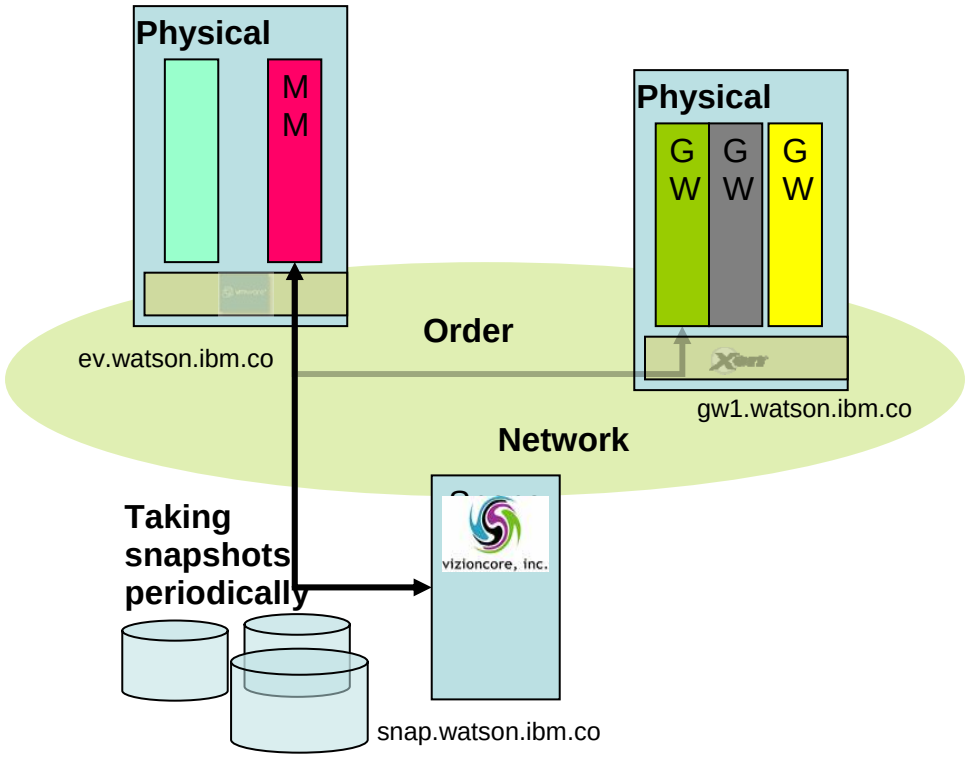


Figure 3: Experimental Setup in our study

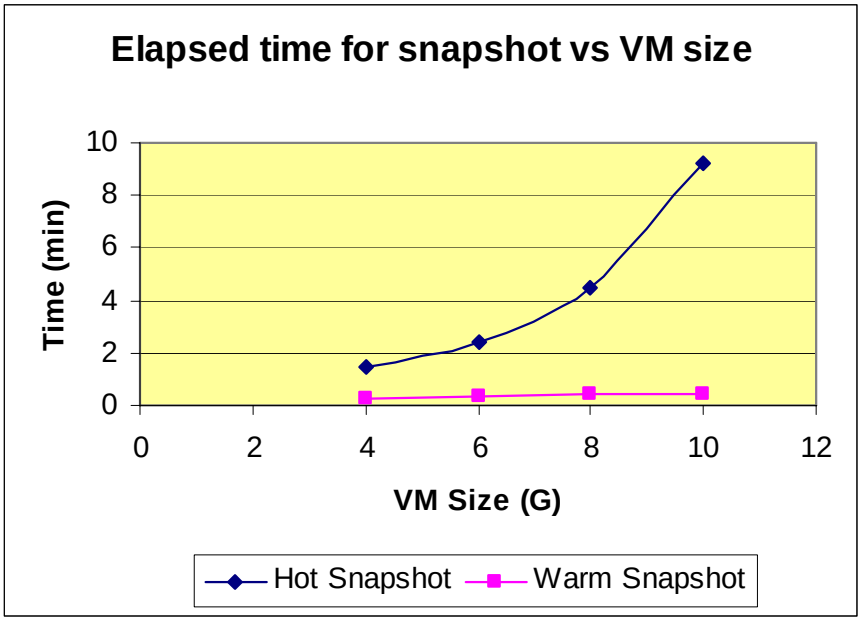
Results and Discussion

To measure the efficiency of our approach we measure the elapsed time between the time the snapshot process is invoked and the time it finishes the creation of the snapshot as a function of the size of the virtual machine. In Figure 4 we plot our measurements for the hot and warm snapshot. In our experiment a VM hosting a market matching application has a size of ~6 GB; from Figure 4 we can see that the maximum feasible frequency to create snapshots is ~2 minutes and ~20 seconds for hot and warm snapshot, respectively. Considering that a typical market matching engine receives between 2,000 and 5,000 requests per second, the loss window resulting from using snapshots is ~(240,000, 600,000) and ~(40,000, 100,000) requests for hot and warm snapshot respectively. Such large loss windows are prohibited by the reliability requirements of the application.

A naive approach to overcome this problem is to use a mechanism similar to the one used in the backup-primary approach; i.e., by logging transactions into a file. In the event of a failure, the application can be restarted from the latest available snapshot and can catch up to its pre-failure state by replaying the transactions extracted from the log file. While this approach requires additional memory and may incur some delay, it still represents significant savings on hardware and maintenance cost.

Notice that although using warm snapshots results in a smaller loss window when compared to using hot snapshots, the warm snapshot process suspends the VM, dropping existing and incoming connections. To have a secondary VM executing incoming transactions one would have to transfer the books in memory to an external storage

device for their access. Unfortunately, such an approach would incur a high overhead and therefore was discarded in our study. A more interesting/appealing solution would be to buffer incoming requests during the time the VM is suspended; doing so would appear as if network congestion or an overloaded CPU may be causing the server sluggishness.



To create the snapshots both methods scan the VM in its totality, the only difference being that in the hot snapshot all futures writes are written as log data into a REDO file. Two processes influence the performance of the hot snapshot: the logging process responsible of writing into the REDO file all futures writes in the VM, and the process responsible of committing the REDO file into the VM once the snapshot has been created. The former is I/O intensive and therefore is the primal factor responsible of the duration of the whole process. The outward appearance of the former is an overloaded VM (slowdown) and is perceived only when the REDO file is significantly large. In our experiments the REDO file corresponding to the market matching application grows as large as 0.6 Gigabyte; therefore, the impact of committing the changes to the VM is imperceptible.

Since in our experiments we perform hot differential snapshots the amount of data that needs to be stored in checkpoint is fairly small when compared to performing a full snapshot. On average the size of the checkpoint file for the market matching VM was near 30M, which represents only 0.005 fraction of the full size of the VM. Notice that the size of the REDO file is an important parameter to consider in the design of backup systems since typically one wants to transfer the snapshot to an external database over the network as soon as the snapshot has been created.

Backup from within the application

Given the discouraging results obtained when using virtualization based backup, we looked at the viability of providing additional availability support from within the application. More specifically, we measured the overhead incurred by the application if it has to dump the average loss window of 540,000 orders into the file system. We found that doing so takes an additional 115 seconds, which represent an overhead of 63% (relative to the 3 minutes window for the virtualization based backup). Pursuing such an approach would be detrimental given the high requirements of the application under consideration; however, by utilizing parallelization it may be possible to have an additional thread playing the role of the backup agent and dumping the online book into the file system.

Conclusions

In this report we have presented preliminary results on the usability of virtualization to provide with high availability to application with stringent requirements. Our results indicate that virtualization still needs significant refinement to overcome the challenges imposed by this kind of application.