Volume 4, Issue 5, May 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Performance Analysis and Protecting Data Using Post-Copy Based Live VM Migration and Shadow Paging FHE Technique

Sadia Syed
Scholar in Computer Science
Vikrama Simhapuri University
India

Sreenivasa Teja
Scholar in Computer Science
Vikrama Simhapuri University
India

M.Ussenaiah
Assistant Professor
Vikrama Simhapuri University
India

Abstract: ***Cloud computing is an emerging commercial infrastructure paradigm that promises to eliminate the need for maintaining Expensive computing facilities by companies and institutes alike. Through the use of virtualization and resource time-sharing, clouds serve with a single set of physical resources a large user base with different needs. Thus, clouds have the potential to provide to their owners the benefits of an economy of scale and, at the same time, become an alternative for scientists to clusters, grids, and parallel production environments. However, the current commercial clouds have been built to support web and small database workloads, which are very different from typical scientific computing workloads. Moreover, the use of virtualization and resource time-sharing may introduce significant performance penalties for the demanding scientific computing workloads. In this work we analyze the performance of cloud computing services for scientific computing workloads. We quantify the presence in real scientific computing workloads of Many-Task Computing (MTC) users, that is, of users who employ loosely coupled applications comprising many tasks to achieve their scientific goals.***

These resources utilization are pay per use manner here. Previous servers have processing storage, memory levels are less. There is no possibility for encryption complete content. Some content available as a plain text, remaining content available as a cipher text. This two type's format content starts the transmission. Attackers are entering automatically leakage of data problems generate here. That's why this type of network comes under insecure. It can deliver the incorrect data in destination. Users are not satisfies with the help of these services. Security is the primary obstacle that prevents the wide adoption of this promising computing model, especially for customers when their confidential data are consumed and produced during the computation.

The above limitations are overcome using the ShadowPaging3, FHE and Linear Programming Technique in cloud computing. These types of techniques are providing good secure network and optimization solution. User is ready for transfer the large file to another user. Here large file assume as a large problem. Using linear programming large files divide into sub parts using decomposition. Transformation techniques start the allocation of decomposed parts indifferent servers. Different servers provide the perfect infrastructure for encryption with sufficient computational resources. Before starts the outsourcing total content is encrypted inclient side. After deliver the content verifies the proof. Proof it is matched with server proof then performs the decryption. It's delivers as a correct data.

Index Terms: Performance measures, Performance evaluation, shadow computing, Virtual Machines, Operating Systems, Process Migration, Post-Copy, XN, encryption techniques, Outsourcing Techniques

I. Introduction

SCIENTIFIC computing requires an ever-increasing number of resources to deliver results for ever-growing problem sizes in a reasonable time frame. In the last decade, while the largest research projects were able to afford (access to) expensive supercomputers, many projects were forced to opt for cheaper resources such as commodity clusters and grids. Cloud computing proposes an alternative in which resources are no longer hosted by the researchers' computational facilities, but are leased from big data centers only when needed. Despite the existence of several cloud computing offerings by vendors such as Amazon and GoGrid, the potential of clouds for scientific computing remains largely unexplored. To address this issue, in this paper we present a performance analysis of cloud computing services for many-task scientific computing. The cloud computing paradigm holds great promise for the performance-hungry scientific computing community: Clouds can be a cheap alternative to supercomputers and specialized clusters, a much more reliable platform than grids, and a much more scalable platform than the largest of commodity clusters. Clouds also promise to "scale by credit card," that is, to scale up instantly and temporarily within the limitations imposed only by the available financial resources, as opposed to the physical limitations of adding nodes to clusters or even supercomputers and to the administrative burden of over-provisioning resources. Moreover, clouds promise good support for bags-of-tasks, which currently constitute the dominant grid application type However; clouds also raise important challenges in many aspects of scientific computing, including performance, which is the focus of this work. There are three main differences between scientific computing workloads and the initial target workload of clouds: in required system size, in performance demand, and in the job execution model. Size-wise, top scientific computing facilities comprise very large systems, with the top ten entries in the Top500 Supercomputers List together totaling about one million cores, while

cloud computing services were designed to replace the smallto-medium size enterprise data centers. Performance wise, scientific workloads often require High Performance Computing (HPC) or High-Throughput Computing (HTC) capabilities. Recently, the scientific computing community has started to focus on Many-Task Computing (MTC), that is, on high-performance execution of loosely coupled applications comprising many (possibly
Inter-related) tasks. With MTC, a paradigm at the intersection of HPC and HTC, it is possible to demand systems to operate at high utilizations, similar to those of current production grids (over 80% [5]) and Parallel Production Infrastructures (PPIs) (over 60% [6]), and much higher than those of the systems that clouds originally intended to replace (servers with 10-20% utilization). The job execution model of scientific computing platforms is based on the exclusive, space-shared usage of resources. In contrast, most clouds time-share resources and use virtualization to abstract away from the actual hardware, thus increasing the concurrency of users but potentially lowering the attainable performance.

In this we also address the problem of optimizing the live migration of system virtual machines (VMs). Live migration is a key selling point for state-of-the-art virtualization technologies. It allows administrators to consolidate system load, perform maintenance, and flexibly reallocate cluster-wide resources on-the-fly. We focus on VM migration within a cluster environment where physical nodes are interconnected via a high-speed LAN and also employ a network-accessible storage system (such as a SAN or NAS).State-of-the-art live migration techniques use the pre-copy approach, which works as follows. The bulk of the VM's mem-Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. The VM's memory state is migrated to a target node even as the VM continues to execute at a source node. If a transmitted page is dirtied, it is re-sent to the target in the next round. This iterative copying of dirtied pages continues until either a small, writable working set (WWS) has been identified, or a preset number of iterations are reached, whichever comes first. This constitutes the end of the memory transfer phase and the beginning of service downtime. The VM is then suspended and its processor state plus any remaining dirty pages are sent to a target node. Finally, the VM is restarted and the copy at source is destroyed.Pre-copy's overriding goal is to keep downtime small by minimizing the amount of VM state that needs to be transferred during downtime. Pre-copy will cap the number of copying iterations to a preset limit since the WWS is not guaranteed to converge across successive iterations. On the other hand, if the iterations are terminated too early, then the larger WWS will significantly increase service downtime. This approach minimizes two metrics particularly well – VM downtime and application degradation – when the VM is executing a largely read-intensive workload. However, even moderately write-intensive workloads can reduce pre-copy's effectiveness during migration. For this we also need to implement implement and evaluate another strategy for live VM migration, called post-copy, previously studied only in the context of process migration. On a high-level, post-copy migration defers the memory transfer phase until after the VM's CPU state has already been transferred to the target and resumed there.Postcopy first transmits all processor state to the target, starts the VM at the target, and then actively pushes memory pages from source to target. Concurrently, any memory pages that are faulted on by the VM at target, and not yet pushed, are demand-paged over the network from source. Post-copy thus ensures that each memory page is transferred at most once, thus avoiding the duplicate transmission overhead of pre-copy. We also supplement active push with adaptive pre-paging. Preparing is a term from earlier literature on optimizing memory-constrained disk-based paging systems. It traditionally refers to a more proactive form of pre-fetching from disk in which the memory subsystem can try to hide the latency of high-locality page faults or cache misses by intelligently sequencing the perfected pages. Modern virtual memory subsystems do not typically employ pre-paging due increasing DRAM capacities. However, pre-paging can still play a critical role in post-copy VM migration by improving the effectiveness of the active push component. Preparing adapts the sequence of actively pushed pages by treating network page faults (or major faults) as hints to guess the VM's page access locality at the target. Pre-paging thus increases the likelihood that pages in the neighborhood of a major fault will be available at the target before they are accessed by the VM.We show that our implementation can reduce major network faults to within 21% of the VM's working set for large workloads.
Additionally, we identified deficiencies in both migration techniques with regard to the handling of free pages during migration.To avoid transmitting the free pages, we employ a "dynamic selfballooning"(DSB) mechanism. Ballooning is an existing technique that allows a guest OS to reduce its memory footprint by releasing its free memory pages back to the hypervisor. DSB automates the ballooning mechanism so it can trigger frequently (every 5 seconds)

Similarly, post-copy requires the servicing of major network faults generated at the target, which also slows down VM workloads. Without degrading application performance. Our DSB implementation responds directly to OS memory allocation requests without the need for kernel patching. It does not require external introspection by a co-located VM nor does extra communication through the hypervisor.DSB thus significantly reduce total migration time by eliminating the transfer of free memory pages. The original pre-copy algorithm has additional advantages: it employs a relatively self-obtained implementation that allows the migration daemon to isolate most of the copying complexity to a single process at each node. Also, pre-copy provides a clean method of aborting the migration should the target node ever crash during migration because the VM is still running at the source. Our current post-copy implementation does not handle these kinds of failures. However, we discuss a straightforward approach by which post-copy could provide the same level of reliability as pre-copy. We designed and implemented a prototype of our approach in the Xen VM environment. Through extensive evaluations, we demonstrate situations in which post-copy can significantly improve Migration

performance in terms of pages transferred and total migration time. Finally, note that post-copy and pre-copy together enhance the toolbox of VM migration techniques available to a cluster administrator. Depending upon the VM workload type and performance goals of migration, an administrator has the flexibility to choose either of the techniques. For interactive VMs on which many users might depend, pre-copy would be the better approach because processor does not wait for major faults. But for large memory or write-intensive server workloads, post-copy would better suited. Our contributions are to demonstrate that post-copy is a practical VM migration technique and to evaluate its relative merits against pre-copy.

II. Design of Post-Copy Live VM Migration

In this section we present the architectural overview of post-copy live VM migration. The performance of any live VM migration strategy could be gauged by the following metrics.

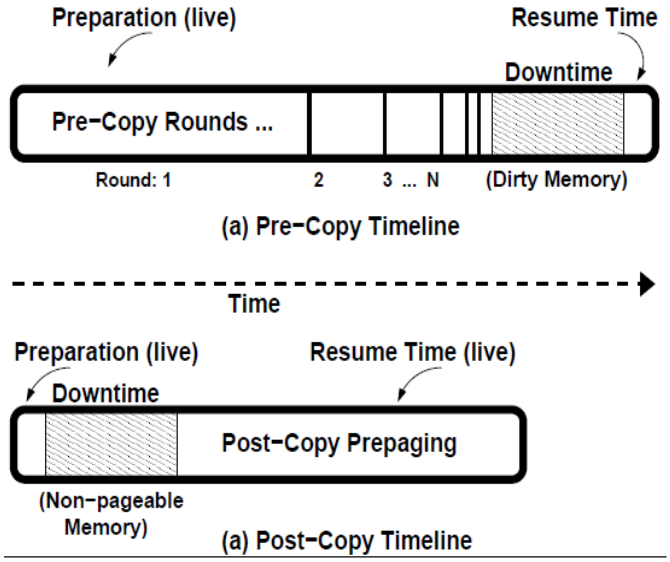


Figure 1. The timeline of (a) Pre-copy vs. (b) Post-copy migration.

1. ***Preparation Time:*** This is the time between initiating migration and transferring the VM's processor state to the target node, during which the VM continues to execute and dirty its memory. For pre-copy, this time includes the entire iterative memory copying phase, whereas it is negligible for post-copy.
2. ***Downtime:*** This is time during which the migrating VM's execution is stopped. At the minimum this includes the transfer of processor state. For pre-copy, this transfer also includes any remaining dirty pages. For post-copy this includes other minimal execution state, if any, needed by the VM to start at the target.
3. ***Resume Time:*** This is the time between resuming the VM's execution at the target and the end of migration altogether, at which point all dependencies on the source must be eliminated. For pre-copy, one needs only to re-schedule the target VM and destroy the source copy. On the other hand, majority of our post-copy approach operates in this period.
4. ***Pages Transferred:*** This is the total count of memory pages transferred, including duplicates, across all of the above time periods. Pre-copy transfers most of its pages during preparation time, whereas post-copy transfers most during resume time.
5. ***Total Migration Time:*** This is the sum of all the above times from start to finish. Total time is important because it affects the release of resources on both participating nodes as well as within the VMs on both nodes. Until the completion of migration, we cannot free the source VM's memory.
6. ***Application Degradation:*** This is the extent to which migration slows down the applications executing within the VM.Pre-copy must track dirtied pages by trapping write accesses to each page, which significantly slows down write-intensive workloads.

Post-Copy via Active Pushing: One way to reduce the duration of residual dependencies on the source node is to proactively "push" the VM's pages from the source to the target even as the VM continues executing at the target. Any major faults incurred by the VM can be serviced concurrently over the network via demand paging. Active push avoids transferring pages that have already been faulted in by the target VM. Thus, each page is transferred only once, either by demand paging or by an active push. Post-Copy via Pre-paging: The goal of post-copy via pre-paging is to anticipate the occurrence of major faults in advance and adapt the page pushing sequence to better reflect the VM's memory access pattern. While it is impossible to predict the VM's exact faulting behavior, our approach works by using the faulting addresses as hints to estimate the spatial locality of the VM's memory access pattern. The pre-paging component then shifts the transmission window of the pages to be pushed such that the current page fault location falls within the window. This increases the probability that the pushed pages would be the ones accessed by the VM in the near future, reducing the number of major faults. Hybrid Pre and Post Copy: The hybrid approach, works by doing a single pre-copy round in the preparation phase of the migration. During this time, the guest VM continues

III. CLOUD PERFORMANCE EVALUATION

In this section we present an empirical performance evaluation of cloud computing services. Toward this End, we run micro-benchmarks and application kernels typical for scientific computing on cloud computing Resources and compare whenever possible the obtained results to the theoretical peak performance and/or the performance of other scientific computing systems.

Using Method:

	Simple criteria				Complex criterion	
	BoTs $\geq$ 100		Tasks $\geq$ 1,000		Jobs $\geq$ 10,000 & BoTs $\geq$ 1,000	
Trace ID	Jobs [%]	CPUT [%]	Jobs [%]	CPUT [%]	Jobs [%]	CPUT [%]
DAS-2	90	65	95	73	57	26
RAL	81	78	98	94	33	28
GLOW	95	75	99	91	83	69
Grid3	100	97	100	97	97	95
SharcNet	40	35	95	85	15	9
LCG	61	39	87	61	0	0
CTC SP2	16	13	18	14	0	0
SDSC SP2	13	93	25	6	0	0
LANL O2	66	14	73	18	34	6
SDSC DS	29	7	44	18	0	0

Our method stems from the traditional system benchmarking. Saavedra and Smith have shown that benchmarking the performance of various system components with a wide variety of micro-benchmarks and application kernels can provide a first order estimate of that system's performance. Similarly, in this section we evaluate various components of the four clouds However; our method is not a straightforward application of Saavedra and Smith's method. Instead, we add a cloud-specific component, select several benchmarks for a comprehensive platform independent evaluation, and focus on metrics specific to large-scale systems (such as efficiency and variability).**Cloud-specific evaluation** An attractive promise of clouds is that they can always provide resources on demand, without additional waiting time. However, since the load of other large-scale systems varies over time due to submission patterns, we want to investigate if large clouds can indeed bypass this problem. To this end, one or more instances of the same instance type are repeatedly acquired and released during a few minutes; the resource acquisition requests follow a Poisson process with arrival rate _ = 1s−1.**Infrastructure-agnostic evaluation** There currently is no single accepted benchmark for scientific computing at large-scale. To address this issue, we use several traditional benchmark suites comprising micro-benchmarks and (scientific) application kernels. We further design two types of test workloads: SI–run one or more single process jobs on a single instance (possibly with multiple cores), and MI–run a single multi-process job on multiple instances.

IV. Related Work

Process Migration: The post-copy technique has been variously studied in the context of process migration literature: first implemented as "Freeze Free" using a file-server then evaluated via simulations and later via actual Linux implementation There was also a recent implementation of post-copy process migration under openMosix In contrast, our contributions are to develop a viable post-copy technique for live migration of virtual machines. We also evaluate our implementation against an array of application workloads during live VM migration, which the above approaches do not. Process migration techniques have been extensively researched and an excellent survey can be found in several distributed computing projects incorporate process migration However, these systems have not gained widespread acceptance primarily because of portability and residual dependency limitations. In contrast, VM migration operates on whole operating systems and is naturally free of these problems.

Pre-Paging: Pre-paging is a technique for hiding the latency of page faults (and in general I/O accesses in the critical execution path) by predicting the future working set and loading the required pages before they are accessed. Pre-paging is also known as adaptive perfecting or adaptive remote paging. It has been studied extensively in the context of disk based storage systems, since disk I/O accesses in the critical application execution path can be highly expensive. Traditional pre-paging algorithms use reactive and history based approaches to predict and prefetch the working set of the application. Our system employs pre-paging for the limited duration of live migration to avoid the latency network page faults from the target to the source node. Our implementation employs a reactive approach that uses any network faults as hints about the guest VM's working set with additional optimizations described in Section and may also be augmented with history-based approaches.

Live VM Migration: Pre-copy is the predominant approach for live VM migration. These include hypervisor-based approaches from VMware Xen, and KVM OS-level approaches that do not use hypervisors from OpenVZ as well as wide-area migration Furthermore, the self-migration of operating systems (which has much in common with process migration) was implemented in building upon prior work atop the L4 Linux microkernel.

All of the above systems currently use pre-copy based migration and can potentially benefit from the approach in this paper. The closest work to our technique is Snow Flock This work sets up impromptu clusters to support highly parallel computation tasks across VMs by cloning the source VM on the fly. This is optimized by actively pushing cloned memory via multicast from the source VM. They do not target VM migration in particular, nor present a comprehensive comparison (or optimize upon) the original pre-copy approach. Further, we use transparent ballooning to eliminate free memory transmission whereas Snow Flock requires kernel modifications to do the same.

Non-Live VM Migration: There are several non-live approaches to VM migration. Schmidt proposed using capsules, which are groups of related processes along with their IPC/network state, as migration units. Similarly, Zap uses process groups (pods) along with their kernel state as migration units. The Denali project addressed migration of check pointed VMs.Work in addressed user mobility and system administration by encapsulating the computing environment as capsules to be transferred between distinct hosts. Internets suspend/resume focuses on saving/restoring computing state on anonymous hardware. In all the above systems, the execution of the VM is suspended, during which applications do not make progress. Dynamic Self-Ballooning (DSB): Ballooning refers to artificially requesting memory within a guest OS and releasing that memory back to the hypervisor. Ballooning is used widely for the purpose of VM memory resizing by both VMware and Xen and relates to self-paging in Nemesis However, it is not clear how current ballooning mechanisms interact, if at all, with live VM migration techniques. For instance, while Xen is capable of simple one-time ballooning during migration and system boot time, there is no explicit use of dynamic ballooning to reduce the memory footprint before live migration? Additionally, self-ballooning has been recently committed into the Xen source tree by Oracle Corp to enable a guest OS to dynamically return free memory to the hypervisor without explicit human intervention. VMware ESX server includes dynamic ballooning and idle memory tax, but the focus is not on reducing the VM footprint before migration. Our DSB mechanism is similar in spirit to the above dynamic ballooning approaches. However, to the best of our knowledge, DSB has not been exploited systematically to date for improving the performance of live migration. Our work uses DSB to improve the migration performance of both the pre-copy and post-copy approaches with minimal runtime overhead. After this related work from three areas: clouds, virtualization, and system performance evaluation. Our work also comprises the first characterization of the MTC component in existing scientific computing Workloads.

Performance Evaluation of Clouds and Virtualized Environments

There has been a recent spur of research activity in assessing the performance of virtualized resources,

Performance studies using general purpose benchmarks have shown that the overhead incurred by virtualization can be below 5% for computation and below 15% for networking [Similarly, the performance loss due to virtualization for parallel I/O and web server I/O has been shown to be below 30% and 10% respectively. In contrast to these, our work shows that virtualized resources obtained from public clouds can have a much lower performance than the theoretical peak. Recently, much interest for the use of virtualization has been shown by the HPC community, spurred by two seminal studies that find virtualization overhead to be negligible for compute-intensive HPC kernels and applications such as the NAS NPB benchmarks; other studies have investigated virtualization performance for specific HPC application domains or for mixtures of Web and HPC workloads running on virtualized (shared) resources Our work differs significantly from these previous approaches in target (clouds as black boxes vs. owned and controllable infrastructure) and in size. For clouds, the study of performance and cost of executing a scientific workflow, Montage, in clouds investigates cost-performance trade-offs between clouds and grids, but uses a single application on a single cloud, and the application itself is remote from the mainstream HPC scientific community. Also close to our work is the seminal study of Amazon S3 which

also includes a performance evaluation of file transfers between Amazon EC2 and S3. Our work complements this study by analyzing the performance of Amazon EC2, the other major Amazon cloud service; we also test more clouds and use scientific workloads. Several small-scale performance studies of Amazon EC2 have been recently conducted: the study of Amazon EC2 performance using the NPB benchmark suite selected HPC benchmarks the early comparative study of Eucalyptus and EC2 performance the study of file transfer performance between Amazon EC2 and S3 etc. An early comparative study of the Dawning Cloud and several operational models extends the comparison method employed for Eucalyptus but uses job emulation instead of job execution. Our performance evaluation results extend and complement these previous findings, and gives more insights into the performance of EC2 and other clouds.

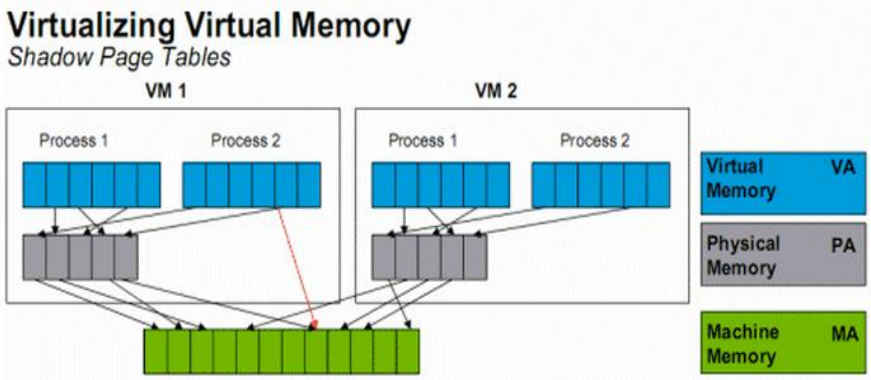
Other (Early) Performance Evaluation much work has been put into the evaluation of novel supercomputers and non-traditional systems for scientific computing. We share much of the used methodology with previous work; we see this as an advantage in that our results are readily comparable with existing results. The two main differences between this body of previous work and ours are that we focus on a different platform (that is, clouds) and that we target a broader scientific computing community (e.g., also users of grids and small clusters).

Other Cloud Work Recent work has considered running mixtures of MTC with other workloads in Cloud-like environments. For this direction of research, our findings can be seen as further motivation and Source of realistic setup parameters.

Shadow Paging Technique in Virtual Environment:

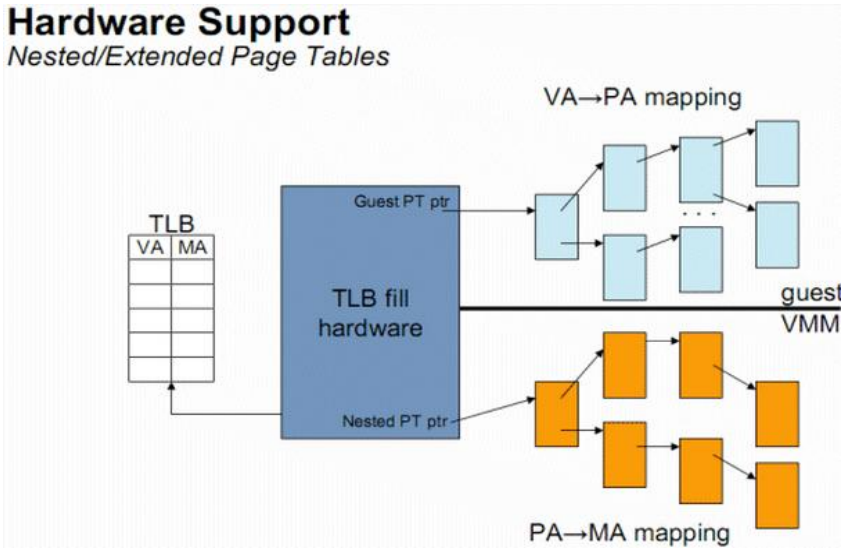
In computer science, shadow paging is a technique for providing atomicity and durability (two of the ACID properties) in database. A page in this context refers to a unit of physical storage (probably on a hard disk), typically of the order of 2^{10} to 2^{16} bytes. Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when

a page is to be modified, a shadow page is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.If the referring pages must also be updated via shadow paging, this procedure may recurse many times, becoming quite costly. One solution, employed by the WAFL file system (Write Anywhere File Layout) is to be lazy about making pages durable (i.e. write-behind caching). This increases performance significantly by avoiding many writes on hotspots high up in the referential hierarchy (e.g.: a file system superblock) at the cost of high commit latency. Write is a more popular solution that uses in-place updates.



Shadow paging is similar to the old master-new master batch processing technique used in mainframe database systems. In these systems, the output of each batch run (possibly a day's work) was written to two separate disks or other form of storage medium. One was kept for backup, and the other was used as the starting point for the next day's work. Shadow paging is also similar to purely functional data structures, in that in-place updates are avoided.

Without shadow pages we would have to translate virtual memory (blue) into "guest OS physical memory" (gray) and then translate the latter into the real physical memory (green). Luckily, the "shadow page table" trick avoids the double bookkeeping by making the MMU work with a virtual memory (of the guest OS, blue) to real physical memory (green) page table, effectively skipping the intermediate "guest OS physical memory" step. There is catch though: each update of the guest OS page tables requires some "shadow page table" bookkeeping. This is rather bad for the performance of software-based virtualization solutions (BT and Para) but wreaks havoc on the performance of the early hardware virtualization solutions. The reason is that you get a lot of those ultra heavy VMexit and VMentry calls. The second generation of hardware virtualization, AMD's nested paging and Intel's EPT technology partly solve this problem by brute hardware force.



EPT or Nested Page Tables is based on a "super" TLB that keeps track of both the Guest OS and the VMM memory management.

As you can see in the picture above, a CPU with hardware support for nested paging caches both the Virtual memory (Guest OS) to Physical memory (Guest OS) as the Physical Memory (Guest OS) to real physical memory transition in the TLB. The TLB has a new VM specific tag, called the Address Space Identifier (ASID). This allows the TLB to keep track of which TLB entry belongs to which VM. The result is that a VM switch does not flush the TLB. The TLB entries of the different virtual machines all coexist peacefully in the TLB… provided the TLB is big enough of course!

This makes the VMM a lot simpler and completely annihilates the need to update the shadow page tables constantly. If we consider that the Hypervisor has to intervene for each update of the shadow page tables (one per VM running), it is clear that nested paging can seriously improve performance (up to 23% according to AMD). Nested paging is especially important if you have more than one (virtual) CPU per VM. Multiple CPUs have to sync the page tables often, and as a result the shadow page tables have to update a lot more too. The performance penalty of shadow page tables gets worse

as you use more (virtual) CPUs per VM. With nested paging, the CPUs simply synchronize TLBs as they would have done in a non-virtualized environment.

There is only one downside: nested paging or EPT makes the virtual to real physical address translation a lot more complex if the TLB does not have the right entry. For each step we take in the blue area, we need to do all the steps in the orange area. Thus, four table searches in the "native situation" have become 16 searches (for each of the four blue steps, four orange steps).

In order to compensate, a CPU needs much larger TLBs than before, and TLB misses are now extremely costly. If a TLB miss happens in a native (non-virtualized) situation, we have to do four searches in the main memory. A TLB miss then results in a performance hit. Now look at the "virtualized OS with nested paging" TLB miss situation: we have to perform 16 (!) searches in tables located in the high latency system RAM. Our performance hit becomes a performance catastrophe! Fortunately, only a few applications will cause a lot of TLB misses if the TLBs are rather large.

Shadow Paging Recovery Technique:

This recovery scheme does not require the use of a log in a single-user environment. In a multiuser environment, a log may be needed for the concurrency control method. Shadow paging considers the database to be made up of a number of fixed-size disk pages (or disk blocks)—say, *n*—for recovery purposes.A **directory** with *n* entries is constructed, where the i th entry points to the i th database page on disk. The directory is kept in main memory if it is not too large, and all references—read or writes—to database pages on disk go through it. When a transaction begins executing, the **current directory**—whose entries point to the most recent or current database pages on disk—is copied into a **shadow directory.** The shadow directory is then saved on disk while the current directory is used by the transaction.

During transaction execution, the shadow directory is *never* modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is *not overwritten.*Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory and the new version by the current directory.

To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery.

In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique. One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle **garbage collection** when a transaction commits. The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation.

Xen 3.3 Feature: Shadow 3:

Shadow 3 is the next step in the evolution of the shadow page table code. By making the shadow page tables behave more like a TLB, we take advantage of guest operating system TLB behavior to reduce and coalesce the number of guest page table changes that the hypervisor has to translate to the shadow page tables. This can dramatically reduce the virtualization overhead for HVM guests.

Shadow paging overhead is one of the largest source of CPU virtualization overhead for HVM guests. Because HVM guest operating systems don't know the physical frame numbers of the pages assigned to them, they use guest frame numbers instead. This requires the hypervisor to translate each guest frame numbers into machine frames in the shadow page tables before they can be used by the guest.

Those who have been around awhile may remember the Shadow-1 code. Its method of propagating changes from guest page tables to the shadow page tables was as follows:

- ✓ Remove write access to any guest page table.
- ✓ When a guest attempts to write to the guest page table, mark it *out-of-sync*, add the page to the out-of-sync list and give write permission.
- ✓ On the next page fault or cr3 write, take each page from the out-of-sync list and:
- ✓ *resync* the page: look for changes to the guest page table, propagate those entries into the shadow page table
- ✓ Remove write permission, and clear the out-of-sync bit.

While this method worked so-so for Linux, it was disastrous for Windows. Windows heavily uses a technique called demand-paging. Resyncing a guest page is an expensive operation, and under Shadow-1, every time a page was faulted in would cause an out-of-sync, write, and a resync.

The next step, Shadow-2, (among many other things) did away with the out-of-sync mechanism and instead emulated every write to guest pagetables. Emulation avoids the expensive unsync-resync cycle for demand paging. However, it removes any "batching" effects: every write is immediately reflected in the shadow pagetables, even though the guest operating system may not have been expecting the address change to be available until later.

Furthermore, Windows will frequently write "transition values" into pagetable entries when a page is being mapped in or mapped out. The cycle for demand-faulting zero pages in 32-bit Windows looks like:

- ✓ Guest process page faults
- ✓ Write transition PTE
- ✓ Write real PTE
- ✓ Guest process accesses page

On bare hardware, this looks like "Page fault / memory write / memory write". Memory writes are relatively inexpensive. But in Shadow-2, this looks like:

- ✓ Page fault
- ✓ Emulated write
- ✓ Emulated write

Each emulated write involves a VMEXIT/VMENTER as well as about 8000 cycles of emulation inside the hypervisor, much more expensive than a mere memory write.

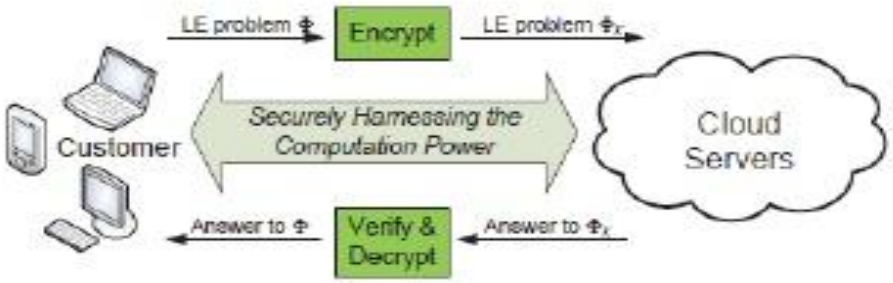
Shadow-3 brings back the out-of-sync mechanism, but with some key changes. First, only L1 pagetables are allowed to go out-of-sync. All L2+ pagetables are emulated. Secondly, we don't necessarily resync on the next page fault. One of the things this enables is to do a "lazy pull-through": if we get a page fault where the shadow is not-present but the guest is present, we can simply propagate that entry to the shadows, and return to the guest, leaving the rest of the page out-of-sync. This means that once a page is out-of-sync, demand-faulting looks like this:

- ✓ Page fault
- ✓ Memory write
- ✓ Memory write
- ✓ Propagate guest entry to shadows

Pulling through a single guest value is actually cheaper than emulation. So for demand-paging under Windows, we have 1/3 fewer trips into the hypervisor. Furthermore, batch updates, like process destruction or mapping large address spaces, are propagated to the shadows in a batch at the next CR3 switch, rather than going into and out of the hypervisor on each individual write.

All of this adds up to greatly improved performance for workloads like compilation, compression, databases, and any workload which does a lot of memory management in an HVM guest.

V. THE PROPOSED SYSTEM ARCHITECTURE



Fully homomorphism encryption (FHE) scheme, a general result of secure computation outsourcing has been shown viable in theory, where the computation is represented by an encrypted combinational Boolean circuit that allows to be evaluated with encryption of their input to produce an encryption of their output. We also investigate duality theorem and derivea set of necessary and sufficient condition for result verification. Such a cheating resiliencedesign can be bundled in the overall mechanism with close-to-zero additional overhead. Both security analysis and experiment results demonstrate the immediate practicality of the proposed mechanism. Duality in linear programming is essentially a unifying theory that develops the relationships between given linear program and another related linear program stated in termsof variables with this shadow-price interpretation. The importance of duality is twofold. First, fully understanding the shadow-price interpretation of the optimal simplex multipliers can prove very useful in understanding the implications of a particular linear-programming model. Second, it is often possible to solve the related linear program with the shadow pricesas the variables in place of, or in conjunction with, the original linear program, thereby taking advantage of some computational efficiency. The importance of duality for computational procedures will become more apparent in later chapters on network-flow problems and large-scale systems.

VI. AIM OF THE SYSTEM

To enable secure and practical outsourcing of LP under the aforementioned model, our mechanism design should achieve the following security and performance guarantees.

Correctness: Any cloud server that faithfully follows the mechanism must produce an output that can be decrypted and verified successfully by the customer.

Soundness: No cloud server can generate an incorrect output that can be decrypted and verified successfully by the customer with non-negligible probability.

Input/output privacy: No sensitive information from the customer's private data can be derived by the cloud server during performing the LP computation.

Efficiency: The local computations done by customer should be substantially less than solving the original LP on his own. The computation burden on the cloud server should be within the comparable time complexity of existing practical algorithms solving LP problems.

VII. ALGORITHM USED

The general working procedure is adopted from a generic approach proposed by R.Gennaro, C. Gentry,and B. Parno while the instantiation in this paper is completely different. According to this approach, the process on cloud server can be represented by algorithm ProofGen and the process on customer can be organized into three algorithms (KeyGen, ProbEnc, and ResultDec).

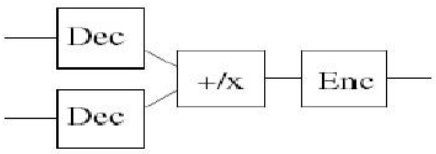
- ✓ KeyGen (1k) → {K}. This is a randomized key generation algorithm which takes a system security parameter k, and returns a secret key K that is used later by customer to encrypt the target LP problem.
- ✓ ProbEnc (K, _) → {_K}. This algorithm encrypts the input tuple _ into _K with the secret key K. According to problem transformation, the encrypted input _K has the same form as _, and thus defines the problem to be solved in the cloud.
- ✓ ProofGen (_K) → {(y,)}. This algorithm augments a generic solver that solves the problem _K to produce both the output y and a proof the output y later Decrypts to x, and is used later by the customer to verify the correctness of y or x.
- ✓ ResultDec (K, _, y,) →{x,} this algorithm may choose to verify either y or x via the proof In any case, a correct output x is produced by decrypting y using the secret K. The algorithm outputs when the validation fails, indicating the cloud server was nonperforming the computation faithfully.

The proposed algorithm provides one-time-pad types of flexibility where we should never use the same secret key K to two different problems. Overall, the basic techniques would choose a secret key K = (Q, λ, θ) and encrypt the input tuple γ into γ k = (A′, B′, b′, θ c), which gives reasonable strength of problem input hiding. Also, these techniques are clearly correct in the sense that solving γ k would give the same optimal solution as solving γ

However, it also implies that although input privacy is achieved, there is no output privacy. Essentially, it shows that although one can change the constraints to a completely different form, it is not necessary the feasible region defined by the constraints will change. Therefore, any secure linear programming mechanism must be able to not only encrypt the constraints but also to encrypt the feasible region defined by the constraints.

VIII. RESULT OF FULLY HOMOMORPHISM ENCRYPTION: SECURITY

Homomorphism encryption schemes that are not semantically secured, like basic RSA, may also have stronger attacks on their one-wayness. A homomorphic encryption (HE) scheme encrypts data in such a way that computations can be performed on the encrypted data without knowing the secret key. Fully homomorphic encryption schemes are based on creating a function to perform two atomic operations which will allow the user to build any kind of circuit. Effectively,



any circuit can be built with two atomic functions, namely addition + and multiplication *. Therefore, to evaluate any circuit, we are only required to beable to add and multiply over F2 two encrypted bits. Gentry used a simple model. Gentry defined the two functions f+ and f* which are equivalent to decrypting both encrypted bits, adding or multiplying such decrypted bits and then encrypting the resulting bits. Hence, it can remove the first encryption securely to perform the addition or the multiplication. Using such a technique, Gentry simplified the quest of constructing a fully homomorphic encryption that can evaluate any circuit on encrypted data by finding an encryption system that can evaluate only some short circuits, namely f+ and f*. That RSA supports multiplications over encrypted data, i.e., given the encryptions of two messages anyone can compute the encryption of their product. It turns out that many public-key encryption schemes are homomorphic including multiplicatively homomorphic to do addition, multiplications and XOR over encrypted data for a long time and even being able to perform these simple operations has been tremendously useful Protecting FHE with Verifiable Computation cloud will not be able to modify the computation circuit without be detected. As a result, our attack will be unsuccessful. However, we argue that in order to use FHE in those models, one must use verifiable computation all the time. Although the cost of verification is low, to generate the minimum circuit and to homomorphically modify it is costly. Thus, whether this technique can be used in practice is doubtful.

IX. ESTIMATING PERFORMANCE

Customer side computation overhead consists of key generation, problem encryption operation, and result verification, which corresponds to the three algorithms KeyGen, ProbEnc, and ResultDec, respectively. Because KeyGen and Result-Dec only require a set of random matrix generation as well as vector-vector and matrix-vector multiplication, the

computation complexity of these two algorithms are upper bounded via O(n2). Thus, it is straight-forward that the most time consuming operations are the matrix-matrix multiplications in problem encryption algorithm ProbEnc. Since m≤n, the time complexity for the customer local computation is thus asymptotically the same as matrix-matrixmultiplication.For cloud server; its only computation overhead is to solve the encrypted LP problem γk as well as generating the result proof ∂, both of which correspond to the algorithmProofGen. If the encrypted LP problem γk belongs to normal case, cloud server just solves it with the dual optimal solution as the result proof ∂, which is usually readily available in the current LP solving algorithms and incurs no additional cost for cloud. If the encrypted problem γk does not have an optimal solution, additional auxiliary LP problems can be solved to provide a proof. Thus, in all the cases, the computation complexity of the cloud server is asymptotically the same as to solve a normal LP problem, which usually requires more than O (n3) time.

X. CONCLUSION & FUTURE GOAL

We have presented the design and implementation of a post-copy technique for live migration of virtual machines. Post-copy is a combination of four key components: demand paging, active pushing, pre-paging, and dynamic self-ballooning. We have implemented and evaluated post-copy on a Xen and Linux based platform and shown that it is able to achieve significant performance improvements over pre-copy based live migration by reducing the number of pages transferred over the network and the total migration time. Thus, in this paper we seek to answer an important research question: *Is the performance of clouds sufficient for scientific computing'?* To this end, we perform a comprehensive performance evaluation of a large computing cloud that is already in production. Our main finding is that the performance and the reliability of the tested cloud are low. Thus, this cloud is insufficient for scientific computing at large, though it still appeals to the scientists that need resources immediately and temporarily. Motivated by this finding, we have analyzed how to improve the current clouds for scientific computing, and identified two research directions which hold each good potential for improving the performance of today's clouds to the level required by scientific Computing.

Regarding Security We Proposed Customers to secretly transform the original LP into some arbitrary one while protecting sensitive input/output information. Fully homomorphic encryption (FHE) scheme, a general result of secure computation outsourcing has been shown viable in theory, where the amputation is represented by an encrypted combinational Boolean circuit that allows to be evaluated with encrypted private inputs we also investigate duality theorem and derive a setoff necessary and sufficient condition for result verification. we show that for any problem γ and its encrypted version γk , solution μ computed by honest cloud server will always be verified successfully. This follows directly from the duality theorem of linear programming. Therefore all conditions derived from duality theorem and auxiliary LP problem construction for result verification is necessary and sufficient. Similar to correctness argument, the soundness of the proposed mechanism follows from the facts that the LP problem γ and γk are equivalent to each other through affine mapping, and all the conditions thereafter for result verification are necessary and sufficient. In near future a goal is set to work around some interesting concepts such as to devise robust algorithms to achieve numerical stability; to explore the sparsity structure of problem for further efficiency improvement; to establish formal security framework; and also to extend our result to non-linear programming computation outsourcing in cloud.

References

[1] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and The art of virtualization. In Proc. of ACM SOSP 2003 (Oct. 2003).

[2] BRADFORD, R., KOTSOVINOS, E., FELDMANN, A., ANDSCHI¨OBERG, H. Live wide-area migration of virtual machines including local persistent state. In Proc. of the International Conference on Virtual Execution Environments (2007), pp. 169–179.

[3] CLARK, C., FRASER, K., HAND, S., HANSEN, J., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual Machines. In Network System Design and Implementation (2005).

[4] CULLY, B., LEFEBVRE, G., AND MEYER, D. Remus: High availability via asynchronous virtual machine replication. In NSDI '07: Networked Systems Design and Implementation (2008).

[5] DENNING, P. J. The working set model for program behavior. Communications of the ACM 11, 5 (1968), 323–333.

[6] DOUGLIS, F. Transparent process migration in the Sprite operating system. Tech. rep., University of California at Berkeley, Berkeley, A, USA, 1990.

[7] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Service, and P. Wong, "Theory and practice in parallel job scheduling," in *JSSPP*, ser. LNCS, vol. 1291. Springer-Verlag, 1997, pp. 1–34.

[8] L. Youseff, R. Wolski, B. C. Gorda, and C. Krintz, "Paravirtualization for HPC systems," in *ISPA Workshops*, ser. LNCS, vol.4331. Springer-Verlag, 2006, pp. 474–486.

[9] E. Deelman, G. Singh, M. Livny, J. B. Berriman, and J. Good, "The cost of doing science on the cloud: the Montage example," in *SC*.IEEE/ACM, 2008, p. 50.

[10] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel,"Amazon S3 for science grids: a viable solution?" in *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, 2008, pp. 55–64.

[11] E. Walker, "Benchmarking Amazon EC2 for HP Scientific Computing,"*Login*, vol. 33, no. 5, pp. 18–23, Nov 2008.

[12] L. Wang, J. Zhan, W. Shi, Y. Liang, and L. Yuan, "In cloud, do mtc or htc service providers benefit from the economies of scale?" in *SC-MTAGS*, 2009.

[13] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters,and D. Epema, "The Grid Workloads Archive," *Future Generation Comp. Syst.*, vol. 24, no. 7, pp. 672–686, 2008.

[14] D. Thain, J. Bent, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny, "Pipeline and batch sharing in grid workloads," in *HPDC*. IEEE, 2003, pp. 152–161

[15] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in Proc. Of CRYPTO'10, Aug. 2010.

[16] Sun Microsystems, Inc., "Building customer trust in cloud computing with transparent security," 2009, online at https://www.sun.com/offers/details/sun transparency.xml..

[17] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in Proc. of ICDCS'10, 2010.[7] J. Li and M. J. Atallah, "Secure and private collaborative linear programming," in Proc.of CollaborateCom, Nov. 2006.

[18] Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing,"2009, online at http://www.cloudsecurityalliance.org.

[19] C. Gentry, "Computing arbitrary functions of encrypted data," Commun. ACM, vol. 53, no. 3, pp. 97–105, 2010.

[20] C. Gentry, "Fully homomorphism encryption using ideal lattices," in Proc of STOC, 2009, pp.169–178.[11] Abhishek Pandey, R.M.Tugnayat and A.K.Tiwari, "Data Security Framework for Cloud Computing Networks", International journal of Computer Engineering & Technology(IJCET), Volume 4, Issue 1, 2012, pp. 178 - 181, ISSN Print: 0976 – 6367, ISSN Online:0976 – 6375.

[21] Gurudatt Kulkarni, Jayant Gambhir and Amruta Dongare, "Security in Cloud Computing",International journal of Computer Engineering & Technology (IJCET), Volume 3, Issue 1,2012, pp. 258 - 265, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375

