Improving server utilization using fast virtual machine migration

C. Isci
J. Liu
B. Abali
J. O. Kephart
J. Kouloheris

Live virtual machine (VM) migration is a technique for transferring an active VM from one physical host to another without disrupting the VM. In principle, live VM migration enables dynamic resource requirements to be matched with available physical resources, leading to better performance and reduced energy consumption. However, in practice, the resource consumption and latency of live VM migration reduce these benefits to much less than their potential. We demonstrate how these overheads can be substantially reduced, enabling live VM migration to fulfill its promise. Specifically, we first experimentally study several factors that contribute to the resource consumption and latency of live VM migration, including workload characteristics, the hypervisor and migration configuration, and the available system and network resources. Then, from the insights gained, we propose an alternative remote direct memory access-based migration technique that significantly reduces VM migration overheads. Finally, via simulation and experiments with real system prototypes, we demonstrate that the reduced VM migration overhead results in significant improvements in resource and energy efficiencies, relative to existing migration techniques.

Introduction

An often-reported advantage of server virtualization is that it enables system administrators to consolidate a set of workloads onto a smaller number of physical servers. This significantly improves system utilization from the low percentages that have been commonly observed in many nonvirtualized data centers [1]. Virtualization and server consolidation are expected to reduce the monetary cost of computation by reducing the amount of hardware and labor required to maintain the hardware, the floor space, and the energy consumption. Most customers who have made the transition to virtualized computing environments have experienced the benefits, although they may have reached a density plateau that reflects the limits of static provisioning. *Density plateau* refers to the new steady state with respect to server densities that is achieved by statically provisioning virtual machines (VMs) on these virtualized servers.

Server consolidation is often treated as a static provisioning operation in which workload peaks are analyzed [2, 3], and then, VMs are placed on the individual physical hosts to minimize the number of servers under maximal load conditions. Here, we define a workload as the operating system and software applications running in a VM. To respond to large-scale workload changes, the VM placements may be periodically revisited, perhaps on a timescale of months or weeks, but these placement frequencies are typically much slower than the natural workload dynamics. We define "placement" as the placement of a VM on a physical server within a management domain. Statistics on production workloads that we [3, 4] and others have collected from nonvirtualized customer systems reveal that the resource-use intensity can be highly dynamic, typically with pronounced daily and weekly cycles and sometimes with higher frequency variations on a scale of minutes. In practice, unused reserve central processing unit (CPU) capacity, called *headroom*, is used to guard against increases in workload intensity. For example, IBM recommends processor headroom of 63%, 40%, 32%, and 25% on 2-, 8-, 16-, and 32-core x86 systems, respectively. The decrease in relative headroom as the number of cores grows is due to the relatively smaller statistical workload fluctuations as the system scales up.

Digital Object Identifier: 10.1147/JRD.2011.2167775

0018-8646/11/$5.00 © 2011 IBM

A potential technology for improving resource utilization in virtualized systems is *live VM migration*, which transfers an active VM from one physical host to another without perceivable interruptions. Bobroff et al. [4] showed, via simulation, that low-latency migration could reduce resource requirements by as much as 50% and service-level agreement violations by up to 20%, and they demonstrated the correlation between resource efficiency and migration latency. Low-latency VM migration and colocation of VMs with complementary demand characteristics effectively creates a single giant server with a small headroom requirement. For example, in a cloud-scale system consisting of thousands of eight-core servers, the 40% headroom requirement may, in principle, be decreased to nearly zero. Furthermore, to save energy, VMs can be consolidated in fewer servers running at their highest capacity (a mode that typically has the highest energy efficiency per unit of computation), with the remaining physical servers put in low power states during periods of low demand [5]. Therefore, understanding key migration characteristics and improving migration performance are crucial for improving resource use and energy efficiency of virtualized systems.

The purpose of this paper is threefold. We experimentally measure migration performance and energy consumption as a function of several factors, use the insights gained from the experiments to design an improved live migration technique, and then measure the resulting improvements in performance and energy consumption. This paper is organized as follows. After providing some background on live migration technology, we characterize the impact of workload characteristics such as memory footprint and usage intensity, and network infrastructure and hypervisor configurations, on migration performance and identify the key performance-limiting factors.

Next, we introduce a new fast live migration technique that we have developed called *FM* (fast migration). FM reduces the observed CPU bottleneck by changing the default hypervisor and networking parameters and by exploiting the remote direct memory access (RDMA) transport found in modern interconnects such as 40-Gb/s InfiniBand**, resulting in speedups in live migration of 1 to 2 orders of magnitude. Finally, focusing on a typical dynamic workload and placing the unused servers in low power states (in a manner that partially resembles the VMware** Distributed Power Manager [6]), we quantify how this speedup can translate into reduced headroom and greater CPU utilization and also demonstrate energy-efficiency improvements of 20% over existing techniques.

Background: characteristics and costs of live migration

Live migration is a key enabler for distributed resource management in virtualized systems because it allows VMs to migrate across physical systems without service disruption [7]. Live migration creates a clear separation between guest user activities and data center management operations, such as provisioning and physical resource allocation. It enables physical system maintenance and remediation without service downtime. Used in conjunction with workload placement and power management middleware, it can support significant performance and power optimization benefits by allowing physical hosts to be evacuated and powered off without service interruption. Here, we first present a brief technical overview of live migration operation and then discuss some of the key workload and system characteristics that can affect migration performance.

Live migration overview

The simplest conceivable VM migration technique is *pure stop and copy*, which entails suspending the VM, transferring its entire memory contents and architectural state to another physical host, and then reinstantiating it there. This approach has the advantage of being highly deterministic and easy to implement, but because of its long *downtime* (the period during which the VM is stopped), this approach is often not preferred for practical applications. In contrast, *live migration* strives to keep the VM downtime to a minimum while also ensuring that the total time required for the overall end-to-end migration (the *migration latency*) stays within reasonable bounds.

A live migration approach called *precopy*-based migration can reduce downtime considerably by transferring memory contents to the destination host while the VM continues to execute on the source host. This is commonly an iterative process. During the period when the active memory of the VM is transferred to the destination, the copy of the VM that is still executing will "dirty" some of the transferred pages by rewriting on them. The hypervisor memory management unit tracks the dirty pages, which are then resent to the destination in subsequent migration iterations. The iterative process continues until a small working set size is reached or until an iteration count limit is reached. At that point, the migration execution changes from the precopy to *downtime* phase, during which the VM is stopped and the remaining active memory of the VM and the architectural state, such as register contents, are transferred to the destination host. Since most of the memory of the VM has been transferred to the destination beforehand, the downtime is typically minimal, except for some pathological cases. After this small downtime, the VM resumes execution at the destination. Current implementations for VMware [8], Xen** [7], and Kernel Virtual Machine (KVM) [9] are all based on *precopy*-based live migration. While precopy is the dominant approach for existing migration implementations, there are also other live migration techniques such as *postcopy*-based migration [10, 11]. Postcopy migration defers memory transfer after the VM is resumed at the destination, and the memory pages can be retrieved on-demand

based on the postresume behavior of the VM. IBM PowerVM* uses both the precopy and postcopy approaches [11].

While the characterization and proposed techniques in this paper are generally hypervisor agnostic, we perform our evaluations and prototype implementation on Linux** KVM [9]. Therefore, here, we also briefly mention the specifics of live migration implementation in KVM. The objective of the existing KVM migration implementation is to minimize the *KVM stop time*, which is the time a VM spends during downtime, possibly at the expense of increased VM migration time. The default "VM stop time objective" in KVM is 30 ms. The VM stop objective is a key parameter of live migration. When increased, some network and storage connections may time out, or various other system outages may occur. When decreased, VM migration may not complete. KVM uses iterative precopy-based migration and a heuristic to determine whether the VM stop time objective can be satisfied.

Cost of migration

While live migration results in very little service disruption (possibly on the scale of tens of milliseconds), it is associated with some costs. Due to iterative scanning, tracking, and transfer of VM pages, additional CPU and network resources are consumed. Therefore, live migration can potentially degrade the performance of the VM that is being migrated, as well as the hosts and network involved in the migration (which, in turn, means that the performance of all of the VMs or other processes on those hosts can be affected) [7–13]. This condition, which we refer to as *brownout*, makes it desirable to minimize the time a VM spends in live migration.

In cloud-scale computing environments, live migration may increase the number of VMs in transit at any given time, thereby increasing the burden on the system infrastructure. Consequently, the gains from agile resource management may diminish as the resource management overheads begin to dominate. Therefore, it is beneficial to improve the VM migration efficiency either by improving the mechanics of VM migration or by improving the physical server or network configurations.

Characterizing migration performance and factors affecting migration

Live migration performance is affected by many factors, which include the following: 1) the VMs themselves, 2) the migration implementation, 3) hypervisor options, and 4) the virtualized infrastructure characteristics such as the servers and the network configuration. Here, we explore these factors and quantify their contribution to overall migration performance with real system experiments based on KVM.

Evaluation framework

We quantify the migration performance on two IBM model HS21 blade servers. We evaluate two KVM implementations with different migration transmit buffer sizes that correspond to the actual canonical KVM configurations of two Linux distributions. We use a 32-bit KVM distribution based on Fedora** Core 10, with a transmit buffer of 1 KB, and compare this to a KVM distribution based on 64-bit Red Hat Enterprise Linux 5.4, with a transmit buffer of 32 KB.

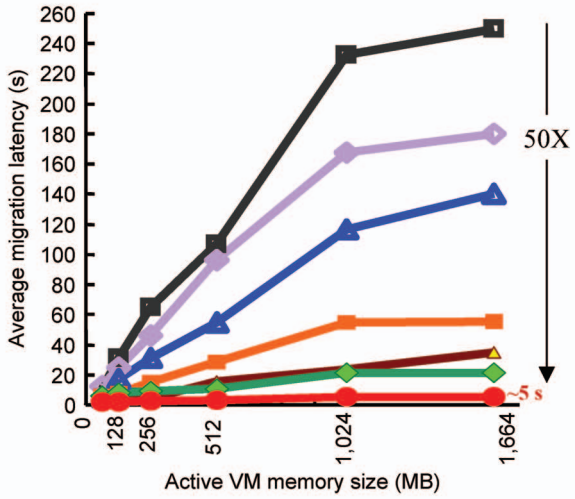
We use a standard 1-Gb/s basic network interface to determine baseline performance. We also use 10-Gb/s Ethernet to demonstrate the performance impact of network bandwidth and larger packet sizes on live migration. RDMA-based migration software is described in the next section.

We used the low-level KVM interface for migration, rather than the higher level support libraries such as *libvirtd*, because the higher level libraries do not provide all of the required migration functionality we needed for our experiments. We developed a centralized controller that interfaces with each KVM host and tracks host inventory and VM resource demand. We also developed individual host agents that communicate with the central controller and between one another for live migration coordination. A live migration is initiated by the centralized controller and afterward carried out by the source and destination hosts. In our implementation, the overall migration proceeds in the following sequence of four steps. First, a "listening" VM starts at the destination. Second, the source host starts the active migration to the destination. Third, the controller periodically polls the source hypervisor to track migration progress. Fourth, once the migration is completed, the source VM is terminated and the VM continues its execution at the destination.

We used two benchmarks to evaluate the workload impact on migration: The first is from Standard Performance Evaluation Corporation (SPEC) CPU2006 suite [14], and the second is based on memory microbenchmarks that we developed. Microbenchmarks allow the controller to precisely monitor and manage the active memory size and page-dirtying characteristics. The memory microbenchmark has an adjustable working set size and a configurable delay to adjust the dirtying rate. We used */sysfs* and *nmon* tools for tracking resource use on each host. The host agents also employ process-level monitoring of individual VM-level resources. We track actual end-to-end migration latency in our experiments, defined as the time difference between the start of active migration (second step) and the first *complete* migration status (end of third step). In our migration characterization experiments, we used the default stop time objective of KVM. However, later in the RDMA evaluations, we also demonstrate the performance impact of using different VM stop time objectives.

Impact of workload characteristics on live migration performance

Memory activity of workloads significantly influences the migration latency. Idle VMs can be migrated in a single or



#	Buf	Netw	MTU	B/W capping
1	1 KB	1 GbE	1,500	KVM default
2	1 KB	10 GbE	1,500	KVM default
3	1 KB	1 GbE	1,500	Unlimited
4	1 KB	10 GbE	1,500	Unlimited
5	32 KB	1 GbE	1,500	Unlimited
6	32 KB	10 GbE	1,500	Unlimited
7	32 KB	10 GbE	8,000	Unlimited

Figure 1

Live migration performance for different hypervisor, hardware, and network configurations at different workload intensities. (See text for an explanation of the acronyms and abbreviations used in the table.)

few iterations since not many memory pages are written between the iterations. Active VMs with a high page-dirtying rate and large active memory may require many more iterations and consequently have much longer migration latency values. (The system configuration impact on migration performance is detailed in the next section.)

We evaluate two dimensions of migration performance. First, we examine how the active memory and the overall writable working set size affect migration latency with a fixed worst case dirtying rate. Second, we evaluate how the rate of change in dirtying rate affects migration latency. **Figure 1** shows the effect of active memory size on migration latency. Each curve represents migration performance under different hypervisor, network, and migration configurations. The horizontal axis represents the amount of active memory used. Note that the active memory size is not the actual configured VM memory size, which is fixed at 2 or 4 GB in these experiments. The performance impact of active memory drastically varies for different migration configurations. Nonetheless, it remains as one of the strongest influencers of overall migration performance. A baseline configuration is shown in the first case (i.e., first row) in the table. As the active memory grows from 64 to 1,536 MB, the migration latency increases from 11 to 246 seconds. This is more than a 20-fold difference in migration latency, observed by the same VM with different workload configurations. For the final configuration (see case 7 in the table), migration latency increases by a factor of three as the workload is increased from its minimal to its maximal value, from less than 2 seconds to slightly above 5 seconds.

Different dirtying rates also result in dramatic and somewhat predictable changes in expected migration latency. For example, for the 1-Gb/s migration network, the migration latency increases from 23 to 180 seconds as the dirtying rate increases from 60 to 200 MB/s. However, the overall latency saturates at this level even as we further increase the dirtying rate because the dirtying rate reaches the line speed around 120 MB/s, after which the transferred pages during precopy cannot keep up with the dirtying rate of the workload. Here, the term "line speed or "wire speed" refers to speeds in which the data is transmitted and processed at the maximum speed allowed by the hardware.

Last, a comparison of idle VM migration latency values demonstrates the impact of overall iterative memory transfer behavior. We configure the same base VM with different memory sizes ranging from 256 MB to 2 GB. Across all the memory sizes, the overall migration latency remains within 2 seconds. This shows that, even with a small amount of active working set, the major contributor to the overall migration latency is the active memory state.

Effect of system configuration on migration performance

Here, we explore some of the key system configuration options and demonstrate the achievable improvements and

observed bottlenecks. In particular, we explore the effect of different hypervisor versions with different migration buffer sizes, different network adapter hardware and network configurations, and different migration configuration parameters. Figure 1 summarizes these different configurations, numbered 1 to 7, depicting progressive combinations of the configuration options. The options are expressed in the figure legend as follows. "Buf" represents the transfer buffer size used in migration implementations of two different hypervisor generations. The low-performance configuration is based on a 1-KB buffer, which limits the migration traffic packet sizes to 1 KB. The 32-KB buffer represents the high-performance hypervisor implementation. In this case, while the buffer can be as large as 32 KB, memory state updates can be still opportunistically sent at 4-KB packet sizes. "Netw" is the network infrastructure used for migration traffic. "1 GbE" is the baseline 1-Gb/s network hardware, and "10 GbE" is the 10-Gb/s hardware. The term "MTU" represents the maximum transmission unit, i.e., the largest packet size that can be transferred over the network without fragmentation.

The last column, "B/W capping," refers to a KVM-specific dynamic migration bandwidth capping policy. The KVM default bandwidth capping policy (see "KVM default" in Figure 1) follows a "slow start," in which the migration bandwidth is gradually increased based on the migration progress. This has a very significant net effect on migration performance, i.e., the bandwidth used and the overall progress follow a staircase pattern, where the majority of elapsed time can be spent in the early steps. In some of the system configurations, we eliminate this B/W capping to enable migration at maximum bandwidth. We refer to this policy as "Unlimited" in Figure 1.

As mentioned, the different latency curves of Figure 1 correspond to the different configurations represented in cases 1 to 7. Here, case 1 represents the baseline configuration with 1-KB buffer, 1-Gb/s network with an MTU of 1,500 bytes, and the default bandwidth capping policy. In contrast, case 7 depicts the case with the most aggressive migration optimizations with 32-KB buffer and 10-Gb/s network with an MTU of 8,000 bytes and with an unrestricted migration bandwidth. The end-to-end performance across the two end cases shows a compelling 50-fold improvement. The remaining cases 2 to 6 show the possible intermediate configuration options.

In Figure 1, we also observe that the baseline buffer size and the KVM default bandwidth management policy exhibit the first order of bottlenecks before the network configuration. Thus, simply optimizing the network infrastructure, without considering all the different constraints having an impact on migration performance, has a limited benefit. For example, the 10-Gb/s hardware configuration in case 2 over the 1-Gb/s base case 1 reduces migration latency by 24% on average. However, a higher (48%) latency improvement is achieved on the 1-Gb/s network (case 3) simply by employing the unlimited bandwidth policy. The combined performance improvement with both unlimited bandwidth policy and larger transfer buffer on the 1-Gb/s network (case 5) leads to a fourfold reduction compared to the base case (case 1). In both cases 3 and 5, the improvements were based on simple hypervisor and migration configurations, without necessitating any hardware change.

As we improve migration performance with a larger transmit buffer and unlimited bandwidth policy, the performance bottlenecks begin to shift from hypervisor and migration implementation to the network infrastructure and configuration (configuration for cases 5 to 7). The network bandwidth becomes the true fundamental bottleneck for the 1-Gb/s network with larger transmit size and unlimited bandwidth (case 5). This shows that the migration bandwidth usage pegs at 1 Gb/s, whereas the other resources remain underutilized. (When we use the term *pegs*, we mean that the migration processes consume all the bandwidth that is available to them, thus saturating the bandwidth usage at 1 Gb/s.) In this case, improving the network infrastructure to a 10-Gb/s network (case 6) has much more significant benefits compared with the prior case. Therefore, while the prior comparison (case 1 versus 2) achieved only a 24% improvement, updating the network infrastructure (case 5 versus 6) leads to a 2.5-fold reduction in overall migration latency. Configuration case 7 presents one last level of improvement via increasing the MTU. With 1-KB transmit buffers and a 1-Gb/s network, the impact of increasing MTU from 1,500 to 8,000 bytes is negligible, as the migration traffic cannot take advantage of the larger MTU. However, with a 10-Gb/s network and a 32-KB buffer, the impact of larger MTU sizes (case 7) also becomes quite significant, further reducing the migration latency of the prior configuration (case 6) with a smaller MTU by a factor of 3.6.

With the impact of different configuration options, we have observed the primary migration bottlenecks shift from bandwidth capping and transmit buffers to network infrastructure and configuration. With the final configuration (case 7), however, the bottleneck shifts to a different resource, i.e., the compute capacity of the host. **Figure 2** shows this CPU saturation for configuration case 7, where the migration bandwidth usage peaks at approximately 4.5 Gb/s as the thread orchestrating the VM migration saturates the processor core on which it is running, pegging it at 100%. At this point, the described configuration options cannot directly mitigate this final bottleneck. The migration performance may be further improved by a different migration implementation with a reduced CPU demand or using multiple migration threads. We discuss our RDMA-based migration implementation for reducing CPU demand in the next section. Another key performance aspect of live migration is the VM stop time objective, which we also explore in the following section.

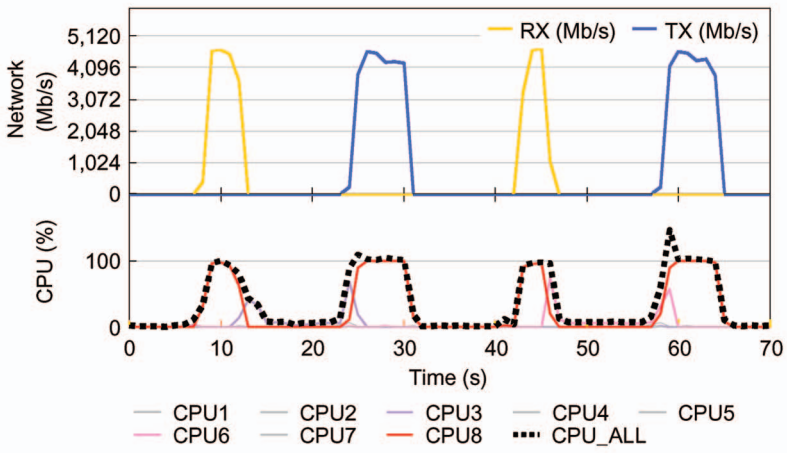


Figure 2

Network and CPU resource use for four consecutive migrations of the same VM, based on a 10-Gb/s network configuration with MTU = 8,000 B, on a hypervisor with a 32-KB transmit buffer and migration bandwidth capping policy. The first and third peaks depict an incoming VM migration (i.e., the host is the destination), and the second and fourth peaks show an outgoing migration (i.e., the host is the source). (RX: receive; TX: transmit.)

Low-latency VM migration using RDMA

By exploiting the observations in the previous section and making use of RDMA, we modified the VM migration code in the Linux KVM to develop a fast migration mechanism that we call *FM*. FM bypasses known bottlenecks in the KVM and Transmission Control Protocol (TCP) software stacks to migrate VMs at near wire speed, 1 to 2 orders of magnitude faster than the existing implementations.

Use of RDMA versus TCP for live migration

The current KVM migration implementation, based on TCP, has a relatively high processing overhead. TCP also requires large message sizes to achieve its peak bandwidth. (It is worth noting that the current migration implementation in KVM [9] was not designed for speed, as noted by its author A. Liguori in a private communication. It will be beneficial to compare a performance-optimized version of future TCP-based migration implementation with the RDMA-based migration. Improving the performance of the TCP stack and TCP-based migration was beyond the scope of this paper.)

High-latency transports have a cascading effect on the migration latency. Not only are the per-page transfer latency and duration of migration iterations longer, but the workload modifies more pages during that increased latency, which increases the number of iterations required to complete the migration. We observed a nonlinear relationship between the migration latency and the network throughput. For some workloads, live migration never completes because the VM stop time objective cannot be satisfied.

We believe it would be worthwhile to explore using the RDMA mechanism found in modern interconnects such as InfiniBand, Ethernet (iWarp**), and RDMA over Ethernet to reduce live migration latency [15]. The potential advantage of RDMA is its relatively low processing overhead due to its elimination of network protocol overheads typically associated with TCP. RDMA transfers application data directly (zero-copy) from source memory to target system memory. Open Fabrics Library isolates the hardware differences between different fabrics (InfiniBand or 10 Gb/s Ethernet). On the other hand, the main drawback of using RDMA operations is that they require user pages to be present in memory during transfer because RDMA network adapters use physical memory addresses. This is called "pinning" or memory registration. (Note that in a typical RDMA operation, the entire memory is not pinned but only the selected pages being transferred at the time.) Two additional drawbacks are that 1) currently, RDMA is mostly

employed in scientific clusters and 2) RDMA-enabled hardware is generally more expensive. Careful implementation and experimentation are needed to establish whether the advantages of RDMA outweigh its disadvantages.

RDMA-based FM implementation

We added an RDMA-assisted option to the existing migration implementation. In our design, migration protocol—headers, markers, trailers, device states—still flow over TCP, unchanged. RDMA is used only for copying VM memory contents since that is the dominant portion of the state of a VM. Note that both TCP and RDMA transfer data over the same physical wire but using different protocols.

Memory pages need to be registered before the RDMA operation. Memory registration serves two purposes: 1) It pins the pages, making them physically present in memory, and 2) it produces physical addresses of the pages that the RDMA adapter needs. Memory registration can consume a significant amount of time, particularly for unmapped pages. Unmapped pages are virtually in the virtual address space of a VM, but they may not have been physically allocated yet. We measured memory registration overheads of various buffer sizes and derived the following approximations: $T_{\text{unmapped}} = N \times 2.4\ \mu\text{s/page} + 29\ \mu\text{s}$, and $T_{\text{mapped}} = N \times 0.14\ \mu\text{s/page} + 27\ \mu\text{s}$, where N is the buffer size in terms of the number of 4-KB pages. These relationships indicate that per-page-registration overhead is increasingly smaller for larger N.

We implemented three different mutually exclusive RDMA-based migration schemes: option A, overlapped memory registration and data transfer; option B, preregistered memory; and option C, skip unmapped guest VM pages (see **Figure 3**).

Option A registers memory in large chunks and attempts to eliminate the latency impact of memory registration by overlapping it with RDMA transfer. VM memory is logically treated in 1-MB chunks for registration purposes only. When a page needs to be registered, the surrounding 1 MB of memory is also registered to amortize the overhead. The registration operation is expected to overlap with the previously initiated data transmissions.

Option B registers all memory pages in advance before VM migration starts. We introduced a new KVM quick emulator (QEMU) monitor command named "*migrate_rdma_prepare*" for this purpose. QEMU is the open-source processor emulator used in KVM. The main drawback of this approach arises from the fact that identifying VMs to migrate in advance may not be feasible in some agile migration management scenarios.

Option C identifies unmapped guest VM pages from the */proc/(pid)/pagemap* interface and transfers a page only when it is backed by a physical memory or a swap device. VM pages that have never been accessed or "ballooned out" by the hypervisor may not physically exist; typically, those pages logically contain all zeros (called a zero page). Option C skips those pages, whereas options A and B transfer them, although they contain no useful information. Option C may be useful in some operational scenarios, where VM memory size is much larger than the mapped memory size.

Changes to the existing KVM migration code were minimal. No changes to the Linux kernel were made. Only ten files in the user-space portion of KVM were altered; these pertained to QEMU and migration. To support RDMA, we introduced 1,009 lines of code in two new files. Overall, we inserted 1,246 new lines of code and deleted 22. We used the Open Fabrics RDMA library and Connection Manager Abstraction for connection setup/removal management across the network. Two command line options were added to the qemu command line. The `-rdma...` options indicate that the target server is able to receive data by RDMA, i.e., `-rdmahost 192.168.1.170 -rdmaport 9999 -incoming tcp:0:8888`. On the source system, migration is started the same way as before, i.e., `migrate tcp:192.168.1.170:8888`.

Experiments: benchmarking setup

We evaluated RDMA-based migration on a 40-Gb/s InfiniBand system and compared it to a TCP-based implementation. Here, we present the main results of our experiments.

Benchmark hardware setup consists of a two IBM 3650-M3 servers interconnected by InfiniBand. The M3 model server has Intel X5670 sockets, a 2.93-GHz processor with six cores and 12 hardware threads. The server has four PCI Express** (Peripheral Component Interconnect Express, PCIe**) Gen2 (5 gigatransfers/second) card slots. A 40-Gb/s InfiniBand card is installed in each server; the InfiniBand card has an 8x PCIe Gen2 connector. The `lspci` command reports the card parameters as follows: MT26428 [ConnectX Virtual Protocol Interconnect PCIe 2.0 5GT/s – InfiniBand quad-data-rate/10 GbE].

The benchmark software setup included Linux version 2.6.30 and QEMU 0.12.3. The guest VM was configured with 3 GB of memory and two virtual CPUs. For the workload, in the guest VMs, we ran the *gobmk* benchmark from the SPEC CPU2006 suite (with the `-rate` 10 option) [14–16]. We chose the gobmk benchmark because it executed a significant amount of store instructions (20%), and the memory-dirtying rate has a significant impact on the migration latency, as explained earlier. However, note that the cache–miss rate due to gobmk is relatively low in comparison to a few other benchmarks in the suite, e.g., mcf, omnetpp, and perlbench. We observed that gobmk occupied approximately 360 MB of memory, and the containing guest VM in total occupied approximately 700 MB (out of 3 GB). We started the VM migration after a benchmark warm-up phase of 90 seconds.

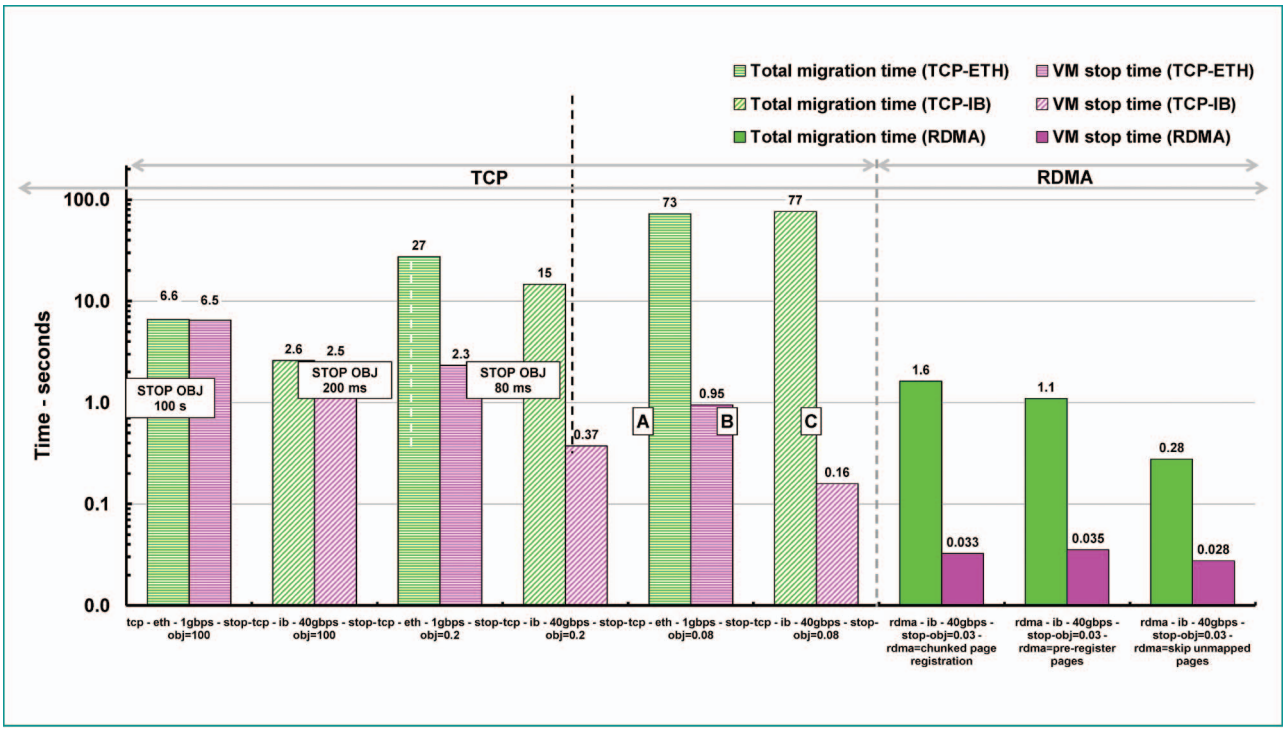


Figure 3

VM migration times and VM stop times for TCP and three different RDMA-based migration methods A, B, and C (see the *x*-axis labels *tcp* and *rdma*). For the TCP-based method, we used three different VM stop time objectives (labeled "stop-obj"). As physical transports, we used Ethernet ("eth-1 gb/s") and InfiniBand ("ib-40 gb/s"). We configured the VM with 3 GB of memory and two virtual processors. The VM was running the SPEC CPU2006 benchmark *gobmk* during its migration.

Experiments: performance results

We have determined the baseline network throughputs on a nonvirtualized Linux system. The TCP bandwidth over the 40-Gb/s InfiniBand network was measured as 8.8 Gb/s with the *iperf* network testing tool. TCP achieves only a fraction of the physical network limit, which is 32 Gb/s for InfiniBand (< 40 Gb/s due to the 10 b/8 b encoding of signals). In comparison, RDMA throughput reaches 26 Gb/s on the same configuration with the *ib_rdma_bw* microbenchmark from the Open Fabric Library. While the measured InfiniBand bandwidth of 26 Gb/s is three times higher than TCP, it is still less than the InfiniBand rate limit of 32 Gb/s. We confirmed that the PCIe Gen2 (8x) connection is limiting the bandwidth at 26 Gb/s.

Figure 3 summarizes the main results of the RDMA-based migration. Each VM migration experiment is represented with an adjacent pair of bars, i.e., total migration time (on the left bar) and the measured VM stop time (on the right bar). The RDMA-based experiment results for options A, B, and C are the three pairs of bars on the right-hand side of the graph. The six bar pairs on the left-hand side are the results of TCP-based migration experiments with two different physical networks, i.e., 1-Gb/s Ethernet and 40-Gb/s InfiniBand. Three different VM stop objectives were used in the TCP experiments, i.e., 100 seconds (which is essentially the stop-and-copy migration technique), 200 ms, and 80 ms. A smaller VM stop objective of 30 ms was used in the RDMA experiments.

The main advantages of our RDMA implementation are apparent when compared with the current TCP-based implementation: 1) RDMA-based migrations complete 1 to 2 orders of magnitude faster than with the current TCP-based implementation, using the same physical network, and 2) measured VM stop times are much shorter with RDMA.

In the TCP-based migration experiments, as we reduce the VM stop time objective from 100 seconds down to, first, 200 ms, and then down to 80 ms, the corresponding migration latency exponentially increases from 6.5 seconds (for the 100-second stop objective) to, first, 15 seconds (for the 200-ms stop objective), and then to 77 seconds (for the 80-ms stop objective). Since the workload is dirtying pages at its own pace, as the VM stop time objective becomes shorter, the migration code and network ability to catch up and move the dirty pages in that shorter time window decreases, and it results in a longer migration latency. In fact, the TCP-based migrations for this benchmark do not complete with a VM stop objective of less than 80 ms.

We also observe that the VM stop objective is a "best effort" parameter, as the measured stop times are sometimes longer than the objective. For example, for the 80-ms stop objective, the measured actual stop time is approximately 160 ms for the 40-Gb/s TCP–InfiniBand configuration and almost 1 second for the 1-Gb/s TCP–Ethernet configuration. Examining these two configurations, we observe that the migration latency values are somewhat similar. However, the faster network generally results in a shorter VM stop time, which is desirable.

Comparing the three different RDMA-based migration implementations, we observe that option A, which overlapped memory registration and communication, is the slowest among the three. However, it is still faster than any TCP-based migration with the same or similar stop objective. Option B migrates faster than A (1.1 versus 1.6 seconds). However, option B requires preregistering memory, which is not accounted for in the 1.1-second value. Option C migrates in 0.28 seconds and is the fastest among all, as it migrates only the mapped pages, about 700 MB out of 3 GB in this experiment.

Note that we did not study the RDMA-based migration in systems that use over-committed memory. Memory overcommitment is a popular technique to increase memory utilization in virtualized systems. The hypervisor reclaims underused pages in VMs and reallocates them to other VMs that need more memory. Memory overcommitment may interfere with the need to pin pages for using RDMA. If the overcommit function swapped pages out to the swap disk, then this will significantly penalize the VM migration performance due to the page-in activity. If the overcommit function simply reclaimed underused pages in a VM, for example, by dropping pages in page caches, then the virtual page in VM essentially becomes an unmapped page. (Here, the term *dropping* refers to deleting a page from a cache or, more precisely, returning the page frame to the free page list.) Thus, for the latter case, we believe that the experimental results of all three options A, B, and C are valid for VMs using overcommitted memory, although we have not verified this experimentally. Future studies should investigate whether memory overcommitment is a barrier to low-latency VM migration.

Improving energy efficiency with low-latency VM migration

The previous two sections detailed the techniques we employed for FM and demonstrated how they greatly reduce migration latency and overhead, improving performance in the process. Here, we explore the extent to which they provide a further benefit, i.e., improved energy savings. The key insight is that reduced latency makes it possible to recover more quickly from unanticipated increases in workload. This allows one to achieve service-level objectives with less headroom and to be more aggressive about powering servers down, and both of these factors directly translate to lower energy consumption.

To illustrate this effect, we conducted the following simulation experiment. We collected a trace of the number of web requests submitted to an IBM website each second for two weeks during the 2009 Australian Open Tennis Championship games. We assumed that the workload was handled by a web server equipped with a load balancer that routed requests to a bank of application servers. We modeled the system with a simple mathematical model of a queue (G/M/k queue) [17] in which the workload distribution was provided by the web request trace. The exponentially distributed service time average was set such that 20 servers could handle the peak load experienced during the games, and the number of servers k was varied according to a simple algorithm with two parameters, i.e., Headroom and minimum time to run (MTTR). The Headroom parameter is the percentage of extra CPU that must be reserved above average demand. MTTR is the minimum time the server must be powered on, as required for reducing risk and hardware wear. The simulator takes as input an assumed latency value, representing the migration latency plus the boot latency (the time required to bring the server from a powered-down state to fully operational).

For each of several assumed total latency values, we run the queuing simulator on the entire two-week trace, measuring individual response times, system utilization, and power consumption. For each latency value, several combinations of Headroom and MTTR are assumed, and the combination that satisfies a specified level of performance (as measured by the fraction of transactions that complete in less than a specified maximum acceptable time) for the least total energy is deemed the optimal combination for that latency. In the experiments reported here, the performance criterion was that the response time should exceed two seconds for no more than 1% of all transactions. To reach this level of performance at total latency of 30 seconds, we found that that the best combination was $(\text{Headroom} = 10\%, \text{MTTR} = 1 \text{ hour})$, whereas for total latency of 180 seconds, it was $(\text{Headroom} = 30\%, \text{MTTR} = 8 \text{ hours})$, and for total latency of 300 seconds, it was $(\text{Headroom} = 60\%, \text{MTTR} = 24 \text{ hours})$. These results quantify how lower latency can substantially reduce the headroom requirement and permit more aggressive power management operations for this particular workload.

Figure 4 summarizes the energy savings for the optimal parameter pairs at several different values of latency. The horizontal line represents the baseline case representing static server consolidation, without dynamic server consolidation or migration. The other curve represents the reduced power consumption with dynamic server consolidation. Specifically, this curve depicts the power consumption for the optimal pair of Headroom and MTTR configurations for each of several assumed values for total latency, denoted in the figure as "power response + migration latency."

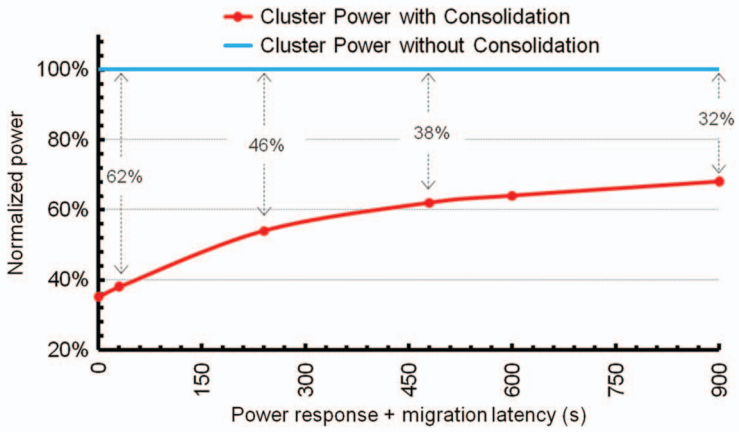


Figure 4

Overall virtualized cluster power and respective power savings with consolidation under different migration and server power-on latency values.

We chose the baseline as static consolidation to demonstrate the achieved improvements with both standard dynamic consolidation without FM and with using fast RDMA-based migration. One conclusion we derive from these experiments relates to the relative energy savings advantage of using FM in dynamic consolidation over standard migration. We highlight this relative improvement in our quantitative results. When the total latency is 900 seconds (15 minutes), there is a substantial energy reduction over static consolidation. This is the regime in which Bobroff et al. performed their experiments [4], where they assumed that the migrations might occur at least 15 minutes apart. Since we were able to obtain a second-by-second trace, we are able to explore what happens for much shorter latency values, finding that there is a significant benefit for further energy savings as the total latency is reduced to less than 5 minutes.

We can estimate the energy-efficiency improvements that would be realized by implementing our FM technique in a system in which the workload intensity variation was similar to that observed at the 2009 Australian Open and in which the VMs were of the size used for the RDMA experiments described in Figure 3 (3 GB, using 700-MB active memory). With the default migration and hypervisor configurations, extrapolation of the results in Figure 3 to the default stop objective of 30 ms suggests that TCP migration over the InfiniBand network would provide a migration time approximately equal to that of the 1-Gb/s Ethernet network, which, from the top curve in Figure 1, is approximately 180 seconds. In contrast, using the best RDMA result (option C) in Figure 3, RDMA would accomplish the same migration in 0.28 seconds. To obtain the total latency, we must add the time it would take to power a server back on from a low-power state. Depending on specifics of the platform and the type of low-power state, we have observed power-on latency values ranging from 30 to 150 seconds or more. If the power-on latency is 150 seconds, then the total latency is 330 seconds for the default hypervisor and network setup and 150 seconds for RDMA. The corresponding normalized power consumption is approximately 60% for the traditional setup and 50% using RDMA. Thus, our FM technique achieves 17% higher energy savings relative to dynamic migration using the traditional settings. On the other hand, if the power latency was just 30 seconds, the total latency is 180 seconds for the traditional case and 30 seconds using RDMA. The corresponding normalized power consumption values are about 52% and 40%, respectively, leading to a 23% improvement in energy savings relative to the traditional default settings. It is very possible that the relative savings could be even larger for larger VMs, but specifics await

further experiments that extend the results reported in Figure 1 to larger VM active memory sizes.

Conclusion

In this paper, we have described cost, performance, and energy benefits of server consolidation. We argued that low-latency VM migration combined with low-latency server power response would significantly reduce the headroom requirements of virtualized systems, therefore leading to efficient server resource and energy utilization. We experimentally characterized the factors having an impact on VM migration latency and its impact on the CPU and network resources. We demonstrated that the active memory size of workloads, the memory-dirtying rate, network configuration, and the VM downtime objective have a significant influence on the migration latency. We then presented an RDMA-assisted VM migration implementation for KVM. We experimentally compared the RDMA-assisted benefits to the existing TCP-based VM migration implementation. While the TCP-based migration may possibly achieve the network throughput of the RDMA-assisted migration, its "tax on" CPU resources is significantly higher than that of RDMA and requires a continuous development effort in order to fully utilize ever-increasing network wire rates. (Here, "tax on" refers to the required resource overhead associated with the migration process.) We also presented the significant potential of energy savings with reduced VM migration latency based on actual data center workloads. Note that the low-latency VM migration technique, e.g., using RDMA, is only a component of effective virtualization management. VM migration management tools and server power state transitions should also exhibit low-latency behavior to realize the promised benefits of low-latency VM migration.

References

1. L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *IEEE Comput.*, vol. 40, no. 12, pp. 33–37, Dec. 2007.
2. X. Meng, C. Isci, J. O. Kephart, L. Zhang, L. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via VM multiplexing," in *Proc. ICAC*, Washington, DC, 2010, pp. 11–20.
3. A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," *Middleware*, vol. 5346, pp. 243–264, 2008.
4. N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. IM*, Munich, Germany, G2007, pp. 119–128.
5. J. Moreira and J. Karidis, "The case for full throttle computing: An alternative to datacenter design strategy," *IEEE Micro*, vol. 30, no. 4, pp. 25–28, Jul./Aug. 2010.
6. VMware Inc. "Resource Management With VMware DRS," in *Whitepaper*, VMware Inc., 2006. [Online]. Available: http://www.vmware.com/pdf/vmware_drs_wp.pdf
7. C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd Conf. NSDI*, 2005, vol. 2, pp. 273–286.
8. M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *USENIX Annual Technical Conf.*, Anaheim, CA, 2005, pp. 391–394.
9. A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The Linux virtual machine monitor," in *Proc. Ottawa Linux Symp.*, Jul. 2007, pp. 225–230.
10. M. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, 2009.
11. W. J. Armstrong, R. L. Arndt, T. R. Marchini, N. Nayar, and W. M. Sauer, "IBM POWER6 partition mobility: Moving virtual servers seamlessly between physical systems," *IBM J. Res. Dev.*, vol. 51, no. 6, pp. 757–762, Nov. 2007.
12. A. Verma, G. Kumar, and R. Koller, "The cost of reconfiguration in a cloud," in *Proc. 11th Int. Middleware Conf. Ind. Track*, 2010, pp. 11–16.
13. W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proc. 1st Int. Conf. CloudCom*, 2009, pp. 254–265.
14. Standard Performance Evaluation Corporation, SPEC CPU2006, gobmk Benchmark Description. [Online]. Available: http://www.spec.org/cpu2006/publications/CPU2006benchmarks.pdf
15. W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High performance virtual machine migration with RDMA over modern interconnects," in *Proc. IEEE Int. CLUSTER*, Austin, TX, 2007, pp. 11–20.
16. T. K. Prakash and L. Peng, "Performance characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor," *ISAST Trans. Comput. Softw. Eng.*, vol. 2, no. 1, pp. 36–41, 2008.
17. L. Kleinrock, *Queuing Systems: Theory*. Hoboken, NJ: Wiley-Interscience, 1975.

Received February 4, 2011; accepted for publication March 9, 2011

Canturk Isci *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (canturk@us.ibm.com).*
Dr. Isci is a Research Staff Member in the Distributed Systems Department at the IBM T. J. Watson Research Center. His research interests are virtualization, data center energy management, and microarchitectural and system-level techniques for workload-adaptive and energy-efficient computing. He received a B.S. degree in electrical engineering from Bilkent University, an M.Sc. degree with distinction in VLSI System Design from University of Westminster, and a Ph.D. degree in computer engineering from Princeton University.

Juixing Liu *Tower Research Capital LLC (jiuxing.liu@gmail.com).*
Dr. Liu works on design and implementation of high-performance trading systems at Tower Research. Previously, he worked at IBM T. J. Watson Research Center and conducted research in areas such as high-performance interconnects and virtual machine technologies. He obtained his Ph.D. degree in computer science and engineering from The Ohio State University. He also holds M.S. and B.S. degrees (both in computer science and engineering) from Shanghai Jiaotong University, China.

Bulent Abali *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (abali@us.ibm.com).*
Dr. Abali is a Research Staff Member in the Systems department. He performs research and development in the areas of servers, storage, networks, systems software, and systems management. His most recent

work includes memory system architectures, memory compression, phase-change memory, and processor and I/O virtualization. He has contributed to numerous IBM products: high-performance computing clusters, and POWER* processor- and x86-based systems hardware and system software. He is an author of 17 patents and 40 technical papers. He received a B.S. degree from Middle East Technical University, and M.S. and Ph.D. degrees from the Ohio State University in electrical engineering.

Jeffrey O. Kephart *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (kephart@us.ibm.com).* Dr. Kephart manages the Agents and Emergent Phenomena team at the IBM T. J. Watson Research Center, as well as a strategic research initiative on Data Center Energy Management. Earlier in his career, Dr. Kephart explored the application of analogies from biology and economics to massively distributed computing systems, particularly in the domains of electronic commerce and anti-virus and anti-spam technology. His team's research efforts on economic software agents and digital immune systems have been widely publicized in such media as *The Wall Street Journal*, *The New York Times*, and *Scientific American*. Dr. Kephart has played a key role in establishing autonomic computing as an academic discipline. He cofounded the International Conference on Autonomic Computing, for which he currently serves as steering committee co-chair. He earned a B.S. degree in electrical engineering from Princeton University and a Ph.D. degree in electrical engineering (with a minor in physics) from Stanford University. He is author or coauthor of more than 20 patents and 125 refereed technical papers. Dr. Kephart is a member of the IEEE and the ACM.

Jack Kouloheris *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (jacklk@us.ibm.com).* Dr. Kouloheris is currently Senior Manager of the Scale Out Software area in IBM Research, where he leads a number of departments working in the area of cloud computing, data center networking, and advanced memory systems. Dr. Kouloheris graduated with a B.S.E.E. degree from the University of Florida in 1982. After graduation, Dr. Kouloheris joined IBM in Boca Raton, Florida, where he worked on chip and board design for what became the IBM AS/400* and 9370 computer systems. In 1986, he returned to school at Stanford University under the auspices of the IBM Resident Study Program and received the M.S.E.E. and Ph.D. degrees in 1987 and 1993, respectively. Dr. Kouloheris returned to IBM at its T. J. Watson Research center in Yorktown Heights, New York. He has worked on a wide variety of projects at IBM Research, including ATM (Asynchronous Transfer Mode) switch design, video compression and decompression algorithms, architectures, chips, and embedded systems design.