Impact of Virtual Machine Granularity on Cloud Computing Workloads Performance

Ping Wang*, Wei Huang, Carlos A. Varela

Department of Computer Science
Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180-3590, USA
Email: {wangp5, huangw5, cvarela}@cs.rpi.edu

***Abstract*—This paper studies the impact of VM granularity on workload performance in cloud computing environments. We use HPL as a representative tightly coupled computational workload and a web server providing content to customers as a representative loosely coupled network intensive workload. The performance evaluation demonstrates VM granularity has a significant impact on the performance of the computational workload. On an 8-CPU machine, the performance obtained from utilizing 8VMs is more than 4 times higher than that given by 4 or 16 VMs for HPL of problem size 4096; whereas on two machines with a total of 12 CPUs 24 VMs gives the best performance for HPL of problem sizes from 256 to 1024. Our results also indicate that the effect of VM granularity on the performance of the web system is not critical. The largest standard deviation of the transaction rates obtained from varying VM granularity is merely 2.89 with a mean value of 21.34. These observations suggest that *VM malleability* strategies where VM granularity is changed dynamically, can be used to improve the performance of tightly coupled computational workloads, whereas VM consolidation for energy savings can be more effectively applied to loosely coupled network intensive workloads.**

***Index Terms*—Cloud Computing; Granularity; Malleability; Performance; Virtual Machine;**

I. Introduction

Cloud computing is an emerging distributed computing paradigm that promises to offer cost-effective scalable on-demand services to users, without the need for large up-front infrastructure investments [1]. A key enabling technology for cloud computing is virtualization. With virtualization, service providers can ensure isolation of multiple user workloads, provision resources in a cost-effective manner by consolidating virtual machines (VMs) onto fewer physical resources when system load is low, and quickly scale up workloads to more physical resources in peak periods. Cloud computing users also benefit from VM abstraction in that they can quickly deploy VM images with preloaded operating systems, software applications, and application data onto cloud environments without being concerned with lower-level physical infrastructure details.

As cloud computing brings significant benefits to service providers and users, it also introduces new challenges. Its pay-per-use business model gives service users such flexibility in purchasing a service that they could try a service with lower cost and turn away from a service whenever they like, which results in fluctuating user workloads. Meanwhile, the available physical resources for one VM image are non-dedicated and dynamic, because service providers usually multiplex multiple VMs onto one single physical machine (PM) and would turn on/off PMs from time to time for the sake of maintenance or energy savings. In such a computing environment, VMs are likely to be either overprovisioned, incurring unnecessary cost, or underprovisioned, which may degrade the quality of service. Thus, in order for the cloud computing paradigm to fulfill its promises, it is necessary to investigate adaptive VM management techniques that enable the VMs to achieve optimal quality of service in spite of fluctuating user workloads and available physical resources. As a step towards adaptive VM management, we tackle the following problem: *given a workload and a set of physical computing resources, what ratio of VMs to physical processors optimizes the workload's performance?*

While existing works on adaptive VM management mainly focus on VM migration [2], [3], [4], the impact of VM granularity on the performance of a cloud environment has not yet been explored. In this paper, we study the impact of VM granularity on two categories of workloads: tightly coupled computational workloads and loosely coupled network intensive workloads. Our performance evaluation indicates that the relationship between VM granularity and workloads' performance is subtle and depends on the nature of the workloads. For tightly coupled computational workloads VM granularity indeed affects the performance significantly and the right VM granularity could enable service providers to improve user satisfaction by reducing the time and cost required for user tasks, serve more users in the same amount of time and hence utilize resources more fully. By contrast, the impact of VM granularity on the performance of loosely coupled network intensive workloads is not critical and service providers could more effectively apply VM consolidation to such workloads for energy savings.

This paper continues as follows. Section II introduces our approach of exploring the impact of VM granularity on workloads' performance. Section III describes the performance evaluation of the tightly coupled computational workload: HPL. Section IV investigates the influence of VM granularity on the performance of our loosely coupled network intensive workload: a VM-based web service workload. Section V discusses our evaluation results and introduces our future work

*Corresponding Author.

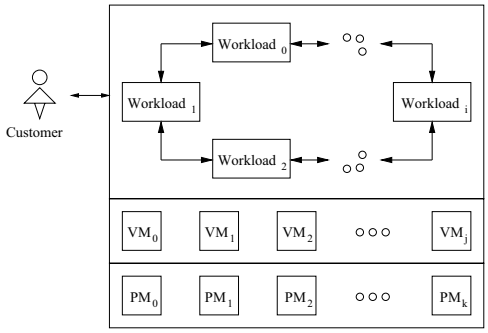

Fig. 1. General architecture of our performance evaluation system: given a workload and a set of physical processors, how many VM instances optimize the workload's performance?

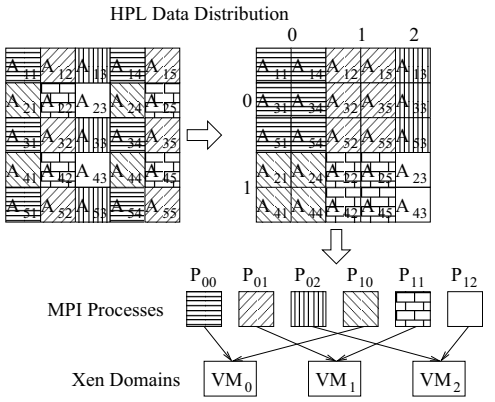

Fig. 2. Architecture of a VM-based computing system.

on VM malleability. Section VI presents related work. Section VII concludes this paper.

II. Approach

This paper aims at exploring the effects of VM granularity on workloads' performance given a set of physical computing resources. With this goal, we design benchmarking experiments to evaluate the performance of one tightly coupled computational application and one loosely coupled network intensive application running on various numbers of VMs. The general architecture of our performance evaluation system is shown in Figure 1.

The representative computational workload we use is HPL 2.0. We run HPL on top of up to 32 guest Xen domains deployed on a 8-CPU PM and up to 48 guest Xen domains deployed on 2 PMs with a total of 12 CPUs. The evaluation results demonstrate that VM granularity has a significant impact on the performance of HPL and it is non-trivial to identify the optimal VM granularity even for our HPL benchmark workload and static physical computing environment.

To form a better understanding of the impact of VM granularity on the HPL performance, we use the profiling tool MPIP-3.2.1 and the system command `top` to collect performance statistics such as MPI time percentage, CPU time percentage per process, and load average. Based on the profiling results, we reach several conclusions. First, the time spent on MPI dominates the overall running time of HPL under the setup of our experiments. Second, the total load average of the VMs can serve as a good indicator of the workload's performance: the lower the total load average, the better the workload's performance.

Next, to study the impact of VM granularity on the performance of network intensive workloads, we deploy a VM-based web system upon up to 28 Xen domains deployed on a 8-CPU PM and benchmark the performance of the web system using the open source http/https benchmarking utility SIEGE 2.69. The experimental results indicate that the effect of VM granularity on the performance of the web system is not critical.

III. Computation Intensive Evaluation

A. Experimental Design We start by investigating how VM granularity affects the performance of tightly coupled computational workloads running on top of VMs. The benchmark we use is HPL 2.0 [5], a benchmark that has been used extensively in the computing industry to measure the floating point computing power of a system.

HPL solves a linear system of order n: $Ax = b$ by first performing LU factorization of the N-by-N+1 coefficient matrix $[Ab] = [[L, U]y]$ and then solve the upper triangular system $Ux = y$ to get the solution x. The N-by-N+1 coefficient matrix is partitioned into NB-by-NB blocks that are cyclically distributed onto a two-dimensional P-by-Q grid of processes. As HPL is run as a MPI job, the processes in the P-by-Q grid are actually MPI processes. The MPI processes are in turn mapped onto a number of VMs in a round-robin way. The parallel HPC computation is tightly coupled, since the computations on the blocks rely on each other and hence the processes to which the blocks are assigned intensively communicate with each other. HPL reports the performance of the underlying distributed system in Gflops (i.e. one billion floating-point operations per second). The architecture of our VM-based computing system is shown in Figure 2.

B. Evaluation Setting We conduct experiments in our Rensselaer cloud computing environment. In the experiments, we deploy VMs on 2 PMs: the first PM is equipped with a quad-processor, dual-core Opteron processor and the second one has a quad-processor, single-core Opteron processor. The Opteron processors run at 2.2GHz, with 64KB L1 cache and 1MB L2 cache. The dual-core PM has 32GB RAM and the single-core PM has 16GB RAM. The communication between the 2 PMs is at 10GB/sec bandwidth and 7usec latency Infiniband, and 1GB/sec bandwidth and 100μsec latency Ethernet.

The hypervisor we use is Xen-3.4.1 and the operating system of the VM instances is Linux 2.6.18.8. The VM instances on the dual-core PM have the default configuration of 1 virtual CPU, 384MB virtual RAM running on 1 physical CPU and 384MB physical RAM, while the VM instances on the single-core PM have the default configuration of 1 virtual CPU, 768MB virtual RAM running on 1 physical CPU and

TABLE I
CONFIGURATION OF VMS DEPLOYED ON 2 PMS

Total # of VMs	On the Dual-core PM		On the Single-core PM	
	# of VMs	# of Virtual/Physical CPUs per VM	# of VMs	# of Virtual/Physical CPUs per VM
3	2	4	1	4
6	4	2	2	2
12-48	8-32	1	4-16	1

TABLE II
HPL PARAMETERS FOR RUNNING VMS ON 1 PM

N	NB	P-by-Q
512	32	4*8
1024	16	4*8
2048	32	4*8
4096	64	4*8

TABLE III
HPL PARAMETERS FOR RUNNING VMS ON 2 PMS

N	NB	P-by-Q
256-8192	64	4*12
8192	16	4*12

768MB physical RAM. When the number of VMs is smaller than the number of the physical CPUs, which means the VMs are unable to use all the physical CPUs given the configuration of one virtual CPU per VM, we increase the number of virtual CPUs per VM so that the VMs can fully utilize the physical CPUs. When the number of the VMs is larger than the number of the physical CPUs, Xen timeshares the CPU time between the VMs. The dual-core PM hosts 2/3 of all the VMs while the single-core PM hosts 1/3, when VMs are deployed on both PMs. Table I lists the configuration of the VMs deployed on 2 PMs. Besides, each PM hosts a unique management VM (i.e. Domain 0) with fixed configuration: 8 virtual CPUs, 2GB virtual RAM running on 8 physical CPUs and 2GB physical RAM on the dual-core PM, and 4 virtual CPUs, 2GB virtual RAM running on 4 physical CPUs and 2GB physical RAM on the single-core PM.

We measure the performance of HPL executing on different numbers of VMs under various HPL parameters. Table II and III list the setting of the three HPL parameters that have significant effects on our evaluation: problem size N, block size NB and process grid P-by-Q. For other configuration parameters, we conform to the tuning guidelines of HPL. For each configuration, we run HPL 10 times and compute the mean, maximum and minimum value over the collected performance values.

C. Results Figure 3 plots the mean, maximum and minimum value of the performance of the HPL application running on top of up to 32 VMs deployed on the 8-CPU PM. We can see that in this experiment running HPL on 8 VMs generates the best performance. Based on the evaluation results, we argue that identifying the VM granularity that gives the optimal performance not only can save users significant amount of time, but also let them accomplish their jobs much more economically in the cloud computing context. We consider the scenario that a user sends a request to execute HPL of problem size 4096 to our VM-based computing system for illustration. Assume the user currently rents 4 VMs to run HPL, then renting 4 more VMs would enable him to accomplish his task with 86% less time, since the Gflops given by 8VMs is 6.09 times higher than that given by 4VMs. Since the cost per hour of renting 4 VMs with 2 CPUs each is approximately the same as that of renting 8 VMs with 1 CPU each, the total cost of the job is proportional to the time it consumes. So utilizing 8 VMs also reduces the total cost of the job by 86%. More interestingly, the user can have his job done with less time and lower cost by renting less VMs. In this case, if the user consolidates his task running upon 16 VMs to 8 VMs, he could finish with 81% less time, because Gflops given by 8VMs is 4.33 times higher than that obtained from running 16VMs. The total cost the job is cut by 81%, since the cost per hour of renting 16 VMs with 0.5 CPU each is roughly the same as that of renting 8 VMs with 1 CPU each.

From the service provider's view, running the workloads at the right VM granularity is beneficial as well. For example, a service provider has a queue of users who request to execute HPL of problem size 4096. Running these HPL tasks on 8 VMs enables each user to finish his job with 86% less time and the provider to serve 6.09 times more users than assigning the HPL tasks on 4 VMs. Therefore, with the right VM granularity the provider can improve user satisfaction by reducing the time and cost required for user tasks, serve more users in the same amount of time and hence utilize resources more fully.

From Figure 3, we also observe that the performance of HPL improves as the problem size grows from 512 to 4096. The reason is that the computation to communication ration increases as the problem size grows and thus HPL can spend more time in computing the linear equation. Furthermore, the difference between the performance gained by having 8 VMs and that obtained from running 4 or 16 VMs becomes more drastic as the problem size increases.

Next, we evaluate the performance of the HPL application under various configurations and VM granularities with the VMs deployed on 2 PMs. Figure 4 shows the performance of the HPL application of smaller problem sizes running on 3 to 48 VMs deployed on 2 PMs and Figure 5 depicts the performance of the HPL application of larger problem sizes. From these figures, we can see that for the problem sizes from 256 to 1024 the optimal VM granularity is 24 VMs on the 2 PMs: the performance achieved is up to 1.40 times higher than that of having 12 VMs and 16.62 times higher than that of utilizing 48 VMs. Meanwhile, for the problem sizes from 2048 to 8192, having 12 VMs on the 2PMs generates the best performance: up to 14.54 times higher than that of running 6 VMs and 1.28 times higher than that of utilizing 24 VMs, except for the configuration of N = 8192 and NB = 64.

The results obtained from our computational evaluation demonstrate that the value of the optimal VM granularity depends on both the available physical resources and the characteristics of the given workload. It is already non-trivial to identify the right VM granularity that optimizes performance even for our HPL benchmark workload and static physical computing environment. Furthermore, a VM-based system that adopts static VM configuration strategies could not continuously achieve optimal performance as its workloads

change. Instead, dynamic configuration strategies are needed to enable a VM-based system to adapt to different workloads.

D. Analysis To form a better understanding of the impact of VM granularity on the HPL performance, we use the profiling tool MPIP-3.2.1[1] and the system command `top` to collect performance statistics such as MPI time percentage, CPU time percentage per process, and load average. We use `top` in batch mode to get its output in files with the delay being 1/20 of the running time of HPL and the iteration limit 10.

MPI Time Percentage: We measure the time spent in MPI when HPL is run on VMs deployed on 2 PMs using MPIP. The mean value of MPI time percentage is 92.45 with a standard deviation of 2.46. This high MPI time percentage can be explained by the fact that in our experiments the size of the linear system is small while the number of the participating processes is relatively large.

CPU Time Percentage per Process: We collect CPU time percentage per process when running HPL on VMs deployed on 2 PMs and observe that the mean value of CPU time percentage per process is 23.49 with a standard deviation of 1.49. This corresponds to that the CPU time of 12 CPUs is divided among 48 processes and consequently each process gets 25% of the CPU time.

Load Average: Unix/Linux operating systems report three load average numbers corresponding to the system load during the last 1, 5, and 15 minutes. We gather the load average per VM for the last one minute from the output files of `top`. Then we divide the load average per VM by the number of CPUs per VM to get the load average per CPU, and multiply the load average per VM with the number of the VMs to obtain the total load average. Figure 6 plots the 3 load average metrics gathered from running HPL on the 8-CPU PM with N = 2048, NB = 32.

To explain the results in Figure 6, we review the configuration of our experiment in Table IV. In addition, we compare the number of processes per CPU and the load average per CPU. It can be seen that the load average per CPU is usually close to the number of processes per CPU. This is because the computation of load average is based on CPU queue length, which in turn is determined by the number of processes per CPU to a large extent. When HPL runs on 1, 2 and 4 VMs, the number of processes per CPU is 4 and the load average per CPU is almost 4. This indicates that the 4 HPL processes are usually in runnable state and waiting for the CPU while only 1 process can be scheduled into the CPU. However, the load average per CPU is only 2.61 even if the number of processes per CPU is 4 when HPL is executed upon 8 VMs, which means the processes spend less time waiting in the CPU queue. This corresponds to our previous observation that 8 VMs on top of the 8-CPU PM is the VM granularity that achieves the best performance. However, the load average per CPU becomes close to the number of processes per CPU again as the number of the underlying VMs continues to grow.

Figure 7 and 8 depict the total load averages of the VMs under different configurations and VM granularities with the VMs deployed on 2 PMs. The two figures show that the optimal VM granularity that generates the lowest total load average is 24 VMs on the 2 PMs for the problem sizes from 256 to 1024, whereas the optimal VM granularity is 12 VMs on the 2 PMs for the problem sizes from 2048 to 8192 except for the configuration of N = 8192 and NB = 64. This is consistent with the observation we obtained from the performance evaluation results and indicates the following relationship between total load average and performance: the lower the former, the higher the latter.

TABLE IV
LOAD AVERAGE PER CPU AGAINST VM GRANULARITY FOR RUNNING HPL ON 1 PM WITH N = 2048, NB = 32

# of VMs	# of Processes per VM	# of Virtual/Physical CPUs per VM	# of Processes per CPU	Load Average per CPU
1	32	8	4	3.91
2	16	4	4	3.87
4	8	2	4	3.88
8	4	1	4	2.61
16	2	1	2	1.86
32	1	1	1	1

However, the VM granularity that generates the lowest total load average cannot be identified easily, because the total load average is the product of two interrelated factors: the load average per VM and the number of VMs. As Figure 6 shows, the load average per VM decreases while the number of VMs increases as we move forward along the x axis. In our experiments, having too few VMs does not give the lowest total load average, but increasing the number of VMs beyond a certain point also incurs unnecessary penalty.

IV. NETWORK INTENSIVE EVALUATION

A. Experimental Design Intensive use of network can be a common feature of many cloud computing applications as one typical scenario of the cloud computing paradigm is that users request services that are hosted in remote data centers. Therefore, we study the effect of VM granularity on the performance of network intensive tasks in this section.

We choose a web system serving content to customers as the representative loosely coupled network intensive workload. Our web system adopts the standard LAMP stack, which is composed of Linux 2.6.18.8, Apache HTTP Server 2.2.14, MySQL 6.0.11, and PHP 5.3.1. To enable our web service to keep up with the ever increasing client requests, the web system is built as a distributed system consisting of multiple cooperating server nodes. However, the underlying server nodes are VMs instead of PMs. For the client end, we use an open source http/https benchmarking utility SIEGE 2.69[2]. In the experiments, we utilize two features of SIEGE. First, SIEGE enables us to stress test the target web server with a

[1] http://mpip.sourceforge.net/

[2] http://www.joedog.org/index/siege-home

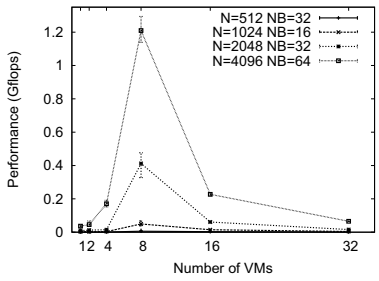

Fig. 3. Performance of HPL running on 1 to 32 VMs deployed on the 8-CPU PM.

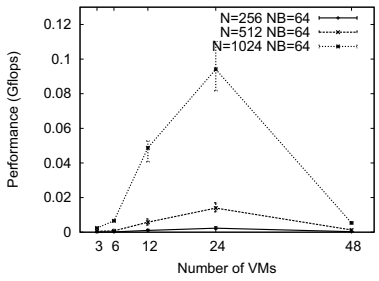

Fig. 4. Performance of HPL of smaller problem sizes running on 3 to 48 VMs deployed on 2 PMs with 12 CPUs in total.

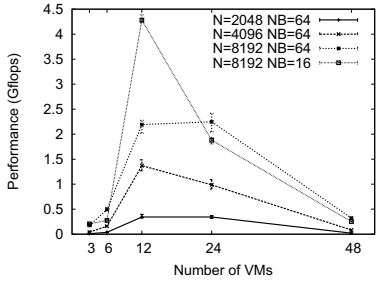

Fig. 5. Performance of HPL of larger problem sizes running on 3 to 48 VMs deployed on 2 PMs with 12 CPUs in total.

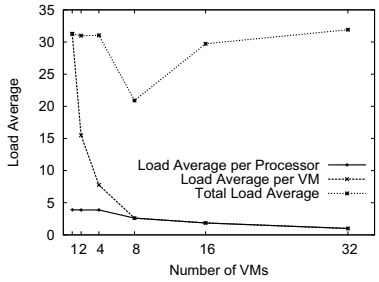

Fig. 6. Load average per CPU, load average per VM, and total load average for running HPL on the 8-CPU PM with N = 2048, NB = 32.

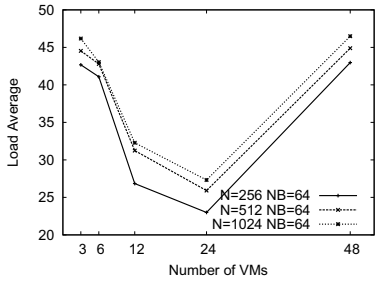

Fig. 7. Total load average against VM granularity for HPL of smaller problem sizes running on 3 to 48 VMs deployed on 2 PMs with 12 CPUs in total.

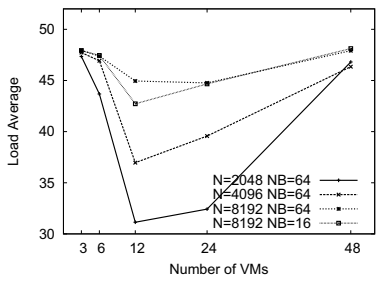

Fig. 8. Total load average against VM granularity for HPL of larger problem sizes running on 3 to 48 VMs deployed on 2 PMs with 12 CPUs in total.

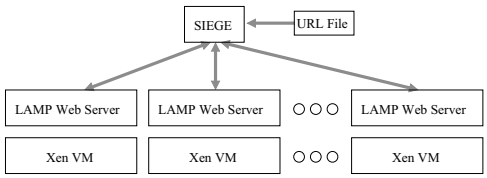

Fig. 9. Architecture of a VM-based distributed web system.

specified number of concurrent simulated users. The stress on the web server generated by a number of simulated users is equal to many times of that generated by the same number of real users, because real users take time to read the web pages they have requested. Secondly, SIEGE can read URLs from a configuration file and hit the URLs sequentially. We use this feature to simulate the situation that clients have the ability to determine the server nodes of the web server to connect to, which is a feasible way of hosting a web server across a set of server nodes [6]. In our experiments the server nodes of the web server do not depend on each other, therefore the web system is loosely coupled. The system architecture of our VM-based web system is depicted in Figure 9.

B. Evaluation Setting The experimental environment is similar with that of the computational experiments, with the following differences. The VMs hosting the web server are deployed on the 8-CPU PM. The client end is hosted on a SUN Blade 1000 machine with 2 processors running at 750MHz and 2GB RAM. The communication between the client end and the VMs is at 100MB/sec bandwidth and 350μsec latency Ethernet. The default configuration of a VM is as follows: 1 virtual CPU, 1GB virtual RAM running on 1 physical CPU and 512MB physical RAM.

The workload of the VM-based web system is generated by having a number of concurrent clients request 512KB data from the web server. We conduct three sets of experiments, corresponding to the following three cases: 1) 100% of the data that our web clients request is from a static file, 2) 50% of the data is from a static file and 50% from the database, and 3) 100% of the data is from the database. The performance metric of the web system is the transaction rate reported by SIEGE and each performance value is averaged over 10 executions.

C. Results Figure 10 shows the transaction rate of our VM-based web system under various numbers of concurrent clients and VM granularities for the three sets of experiments. The figure indicates that the transaction rate fluctuates to a small extent when the granularity of the VMs changes. To be more specific, we present the standard deviations and mean values of the transaction rates corresponded with the different numbers of concurrent clients in Table V. The largest standard deviation of the transaction rates obtained from varying VM granularity is merely 2.89 with a mean value of 21.34. Our explanation for the results is that the bottleneck of the web system is the network bandwidth from the SIEGE client to the web system, not the CPU time or memory space.

D. Analysis Based on the observation that VM granularity is not critical in determining the performance of our VM-

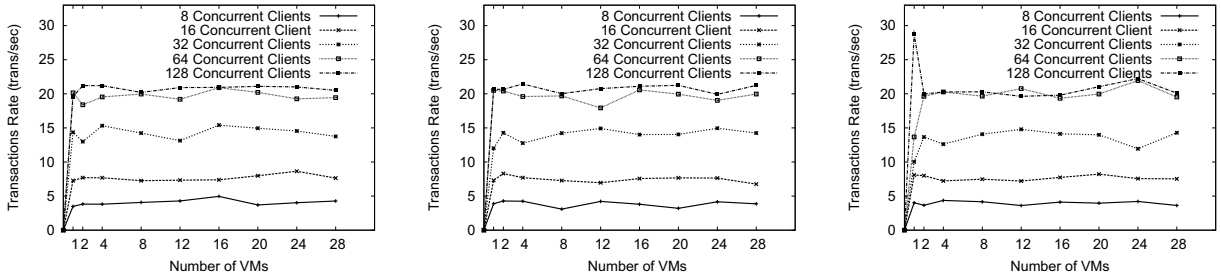

Fig. 10. Transactions rate under various number of concurrent clients and VM granularities for 3 cases: 1) 100% of the data is from a static file, 2) 50% of the data is from a static file and 50% is from the database, and 3) 100% of the data is from the database.

TABLE V
STANDARD DEVIATION AND MEAN OF TRANSACTIONS RATES

# of Concurrent Clients	Standard Deviation/Mean		
	100% from File	50% from File	100% from DB
8	0.43/4.05	0.44/3.86	0.28/3.97
16	0.44/7.66	0.46/7.47	0.36/7.67
32	0.88/14.3	0.96/13.94	1.50/13.28
64	0.73/19.67	0.84/19.74	2.30/19.42
128	0.54/20.74	0.54/20.78	2.89/21.34

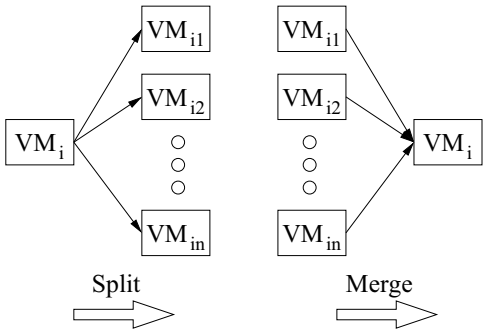

Fig. 11. VM Malleability includes 2 capabilities: Split and Merge.

based web server, we conclude that VM consolidation can be more effectively applied to loosely coupled network intensive workloads. To illustrate this, we consider an example where two PMs PM_A and PM_B located within the same subnet are hosting 10 VMs respectively and these VMs are used as server nodes of a web system. Consolidating the web system by running all of the 20 VMs on PM_A and freeing PM_B will save 50% of the energy cost without degrading performance observably given the bottleneck of the web system is the available bandwidth of the subnet.

V. DISCUSSION AND FUTURE WORK

As our performance evaluation demonstrates that VM granularity affects the performance of tightly coupled computational workloads to a large extent, we realize that it is desirable to equip a cloud computing system with the capability of reconfiguring VM granularity to optimize the performance of tightly coupled computational workloads. However, it is not trivial to identify the optimal VM granularity even for our fixed workload and static physical computing environment. Considering that realistic cloud computing environments are much more complex and dynamic than our experiments illustrate, we claim that malleability, which is the ability to change the granularity of the VM instances dynamically, is needed to enable a VM-based system to adapt to different workloads and changing computing environments. Our VM malleability model is composed of two parts: split and merge. First, one VM can be split into multiple VMs and newly created VMs can be migrated onto other PMs if needed. On the other direction, multiple VMs, which might be located on different PMs, can be merged into one VM on demand.

Following, we sketch our VM malleability model including algorithms for splitting and merging VMs (see Figure 11) and an approach to deal with network and disk resources.

1) Split Algorithm: In the split operation, one source VM is split into N partitions, which are restored to N target VMs on N target hosts. The logical steps that a split operation takes are summarized below.

- Step 1 Reservation and Lock: The algorithm identifies N target hosts for the split operation through sending migration requests and receiving response messages. To guarantee the quality of service provided by the VMs after split, remote hosts are required to own adequate available resources for accepting a migration request. Next, the algorithm gets a lock on the source VM for the split. A lock is granted if the VM is not participating in another split or merge operation.
- Step 2 Suspend: The source VM is suspended and all arriving messages except those related to the split operation are enqueued to be delivered to appropriate target VMs when the split completes.
- Step 3 Split: The algorithm divides the processes of the source VM into N groups so that they can be distributed to N VMs. One complexity involved in process redistribution is to consider the dependency among the processes, e.g. processes may share memory or communicate with signals. To maintain such dependencies, the algorithm checks the dependencies among the processes and adjusts

the grouping to ensure that processes that are dependent on each other are put in the same group. The adjustment might reduce the number of process groups and make the granularity of the split VMs larger.
- Step 4 Save: Afterwards, each process group is saved into one chunk file, which keeps the data, stacks, register contents, communication states and relevant kernel contexts for the processes.
- Step 5 Transfer: The chunk files are transferred to the target hosts. Techniques like pre-copy [4] and post-copy [7] could be used to reduce the downtime of the system caused by this step.
- Step 6 Resume: On each target host, a target VM is created into which the processes in the transferred chunk file are imported. After all the processes resume their execution on the target VMs, the source VM releases its lock and can be removed from its host.

2) Merge Algorithm: In the merge operation, N source VMs, which might be located on different hosts, are merged into one target VM. The logical steps that a merge operation involves are summarized below.

- Step 1 Reservation and Lock: The algorithm selects a remote host that has sufficient resources to serve as the target host for the merge operation and obtains locks on the N source VMs during merge.
- Step 2 Suspend: The algorithm suspends the source VMs and enqueues their incoming messages that are not related to the merge operation. These messages will be delivered to the target VM after the merge completes.
- Step 3 Save: The processes on each source VM are saved in one chunk file.
- Step 4 Transfer: The chunk files are transferred to the target host.
- Step 5 Merge: On the target host, the algorithm creates a target VM, restores the processes from the chunk files and imports the processes into the target VM.
- Step 6 Resume: The target VM resumes the execution of all the processes. The source VMs release their locks and can be removed afterwards.

3) Local Resources: To achieve VM malleability, we also need to consider other resources that are part of a VM besides processes in memory. We will borrow approaches that VM migration uses to maintain network connections and local storage into our malleability model. Network redirection can be implemented with a combination of IP tunneling and Dynamic DNS [8], in which old connections to a migrated VM (which is caused by either a split or merge operation) are redirected to the new IP address through IP tunneling while new communications are directed to the new IP address directly due to the updated Dynamic DNS entry. The tunnel can be revoked after all old connections expire. Our malleability model will initially focus on the VM-based systems that employ a shared storage infrastructure (e.g. SAN or NAS), which can be assumed for most modern cluster environments.

4) Autonomous VM Malleability Rather than taking VM malleability as an alternative to VM migration, it can be employed as a supplementary tool in the repertoire of the autonomous reconfiguration middleware, such as the Internet Operating System (IOS) [9]. The operational aspects of VM malleability involving how to split and merge VMs are solved at the VM level while the strategical aspects of VM malleability (e.g. when to split or merge VMs, how many VMs to split or merge) are handled by middleware. This separation of concerns alleviates the administrator of a VM-based cloud environment from the difficult tasks of determining the system-wide workload and resource usage, and reconfiguring the execution system correspondingly. Instead, autonomous reconfiguration can be achieved by middleware that gathers profiling information about applications and the execution environment and reconfigures the VM-based system via VM migration or malleability.

VI. Related Work

In this section, we discuss the existing research related to our work in three fields: process/component malleability, middleware-driven reconfiguration and virtual machine migration.

A. Process/Component Malleability [10] investigates the impact of process granularity on system-level performance and finds out that while small process granularity generates unnecessary context-switching overhead and increases inter-process communication, large granularity of each process may also yield unsatisfactory performance because the data of the processes do not fit in the upper level of the memory-hierarchy or the processes are too few to be distributed onto the available resources in a balanced way. To enable a parallel application's execution system to run with the right process granularity that utilizes the resources most effectively, process malleability is introduced and implemented as an extension to the process check pointing and migration (PCM) library, a user-level library for iterative MPI applications.

[11] reaches similar observations concerning component granularity and realizes component malleability in the SALSA programming language [12] with SALSA actor being reconfigurable component.

Compared to both process and component malleability, VM malleability has the following advantages. First, while process and component malleability require collaboration from application programmers, VM malleability is transparent to the application layer. Secondly, it is non-trivial to realize malleable processes or components across different programming languages, operating systems and hardware architectures. By contrast, VM malleability is platform independent via the virtualization layer.

B. Middleware-driven Reconfiguration Both process and component malleability are integrated with the Internet Operating System (IOS) [9], a framework for middleware-driven dynamic reconfiguration of distributed computing systems. The reconfiguration mechanisms supported by IOS include: 1) profile application communication patterns, 2) evaluate the

dynamics of the underlying physical resources, and 3) reconfigure processes or components by changing their mappings to physical resources via migration or malleability.

Since middleware-driven reconfiguration has the benefit of freeing application developers or system administrators from dealing with non-functional concerns, the integration of VM malleability with IOS is also one direction of our future work.

C. Virtual Machine Migration VM migration has been explored as an adaptive VM management technique that dynamically changes the mapping between VM instances and physical resources. [4] proposes the pre-copy approach, which iteratively copies the memory of a VM from the source to destination host before releasing the VM at the source and resuming it at the destination. [7] describes the post-copy migration that defers the transfer of a VM's memory until its processor state has been sent to the target host. To realize VM migration across wide area network, [8] designs a system that pre-copies local persistent state to the destination while the VM still runs on the source host. Meanwhile, a user-level block device is used to record and forward the write accesses to the destination to ensure consistency. The system also contains a network redirection scheme that redirects the existing connections of the migrated VM using IP tunneling and advertises its new IP address with Dynamic DNS. Recently, there also has been significant work on live migration which supports continuous device access [13], [14].

While VM migration provides VM-based cloud environments with capabilities such as flexible physical resource reallocation, system workload consolidation, and on-line maintenance, the effectiveness and scalability of VM migration solutions are limited by the granularity of the VM instances.

VII. Conclusion

This paper tackles the problem of identifying the right ratio of VM instances to physical processors that optimizes the workload's performance given a workload and a set of physical computing resources. To evaluate the impact of VM granularity on workloads' performance, we conduct a series of experiments in which HPL is chosen as the representative tightly coupled computational workload and a web server providing content to customers as the representative loosely coupled network intensive workload. Our results demonstrate that for the computational workload, VM granularity has a significant effect on the workload's performance whereas for the network intensive workload the influence of VM granularity is not critical. Based on these two observations, we conclude that: 1) VM malleability, which is the ability to change VM granularity dynamically, is desirable to enable a VM-based system to perform tightly coupled computational workloads more efficiently, and 2) VM consolidation for energy savings can be more effectively applied to loosely coupled network intensive workloads without observable performance degradation.

Acknowledgment

We would like to thank Jonathan Chen and Steven Lindsey for the deployment and maintenance of the Xen virtual layer in our Rensselaer Cloud Computing Environment. Funding for this research is provided in part by NSF CAREER CNS Award No. 0448407 and NSF MRI No. 0420703.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] J. G. Hansen and E. Jul, "Self-migration of operating systems," in *Proceedings of the 11th ACM SIGOPS European workshop*, Leuven, Belgium, September 2004.

[3] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002, pp. 377–390.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI'05: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286.

[5] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. (2008, Sep.) Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. [Online]. Available: http://www.netlib.org/benchmark/hpl/

[6] M. E. Crovella and R. L. Carter, "Dynamic server selection in the internet," in *HPCS '95: Proc. IEEE International Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Mystic, CT, USA, Aug. 1995, pp. 158 – 162.

[7] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. New York, NY, USA: ACM, 2009, pp. 51–60.

[8] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiberg, "Live wide-area migration of virtual machines including local persistent state," in *VEE '07: Proceedings of the 3rd International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*. New York, NY, USA: ACM Press, June 2007, pp. 169–179.

[9] E. M. Kaoutar, D. Travis, K. S. Boleslaw, and A. V. Carlos, "The Internet Operating System: Middleware for adaptive distributed computing," *International Journal of High Performance Computing Applications (IJHPCA), Special Issue on Scheduling Techniques for Large-Scale Distributed Platforms*, vol. 20, no. 4, pp. 467–480, 2006.

[10] ——, "Malleable iterative MPI applications," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 3, pp. 393–413, 2009.

[11] D. Travis, E. M. Kaoutar, and A. V. Carlos, "Malleable components for scalable high performance computing," in *Proceedings of the HPDC'15 Workshop on HPC Grid programming Environments and Components (HPC-GECO/CompFrame)*. Paris, France: IEEE Computer Society, June 2006, pp. 37–44, best paper award.

[12] C. Varela and G. Agha, "Programming dynamically reconfigurable open systems with salsa," *OOPSLA '01: SIGPLAN Notices. ACM Object Oriented Programming Languages, Systems and Applications Intriguing Technology Track Proceedings*, vol. 36, no. 12, pp. 20–34, December 2001.

[13] S. Kumar and K. Schwan, "Netchannel: a vmm-level mechanism for continuous, transparentdevice access during vm migration," in *VEE '08: Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. New York, NY, USA: ACM, 2008, pp. 31–40.

[14] A. Kadav and M. M. Swift, "Live migration of direct-access devices," *SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 95–104, 2009.