Profiling and Predicting Task Execution Time Variation of Consolidated Virtual Machines

Maruf Ahmed, Albert Y. Zomaya
School of Information Technologies, The University of Sydney, Australia
Email: mahm1846@uni.sydney.edu.au, albert.zomaya@sydney.edu.au

***Abstract*—The *task execution time variation* (TETV) due to consolidation of *virtual machines* (vm), is an important issue for data centers. This work, critically examines the nature and impact of performance variation from a new angle. It introduces a simple and feasibly implementable method of using micro and syntactic benchmarks to profile vms. It is called, the *Incremental Consolidation Benchmarking Method* (ICBM). In this method, server resources are systematically incremented, in order to quantitatively profile the effects of consolidation. Resource contention due to basic resources, like CPU, memory and I/O, have been examined separately. Extended experiments have been done on the combinations of those basic resources, too. All experiments have been done and data are collected from real virtualized systems, without using any simulator. The *least square regression* (LSR) is used on the profiled data, in order to predict the TETV of vms. To profile the TETV data, the server has been consolidated with different types and levels of resource loads. The prediction process, introduced here is straightforward and has low overhead, making it suitable to be applied on a wide variety of systems. The experimental results show that, the LSR models can reasonably well predict TETV of vms under different levels of consolidation. The *root mean square error* (RMSE) of prediction for combination of resources, like CPU-Memory, CPU-I/O and Memory-I/O, are within 2.19, 2.47 and 3.08, respectively.**

Keywords–virtualization; consolidation; performance; variation; prediction.

I. INTRODUCTION

The *virtualization* is an essential part of modern data centers. It is required for running many day-to-day operations, like deploying a fault tolerance scheme, or providing *Cloud* services. A *virtual machine* (vm) is a self-contained unit of execution, usually created with the help of a hypervisor running on top of a physical *host*. The vms are immensely important for data centers, as they help to implement the pay-as-you-go model for the Cloud. Usually, a number of vms are run on a host to reduce the operational cost. All the simultaneously running vms of a physical host are collectively known, as the *co-located vms*. The Cloud users can rent vms and have complete control over the rented vms. However, they do not have access to the underlaying physical hardware.

The *consolidation* of vms, is generally done to increase the resource utilization of virtualized servers. However, it imposes a performance penalty, which manifest itself through the *task execution time variation* (TETV) of co-located vms [1]–[3]. This performance variation happens because of resource contention among the vms. It is an obstacle to efficiently scheduling parallel applications on virtualized systems, for several reasons: (i) The variation depends on the server load and resource contention among the vms. The result is that, the same task may take different amount of time to be completed on different vms. At present there is no well accepted method to predict this variation; (ii)To schedule a task of any parallel application, the execution finish time of all the parents tasks must be known. It becomes difficult due to the TETV. Thus, it is an important issue to address, if parallel applications are to be scheduled on virtualized clusters, efficiently.

Most of the previous works on this area fall into two main categories. The first one, is to explore the cost-performance models of parallel applications on public Clouds [4]–[13]. The other one, is virtualized server consolidation benchmarking [14]–[19]. Nonetheless, one can easily identify several weaknesses of those works: (i) They do not explore the resource contention and performance variation of co-located vms explicitly; (ii) Experiments have been done, mainly with parallel applications. Those have complex internal structure of their own, usually represented by a task graph. The virtualization involves so many layers of abstraction and hardware indirection. The complex internal structures of an application can make it difficult to accurately capture the relationship among the co-located vms; (iii) They do not provide the option to control usages of different computing resources, either individually or granularly during experiments. In this case such an ability is very desirable; (iv) Consolidation benchmarks are designed to provide an overall average point of some combination of tests, not details about different levels of consolidation; (v) The consolidation benchmarks are hugely dependent on vendors and their applicability for comparing different virtualization technologies are not well defined. For example, the *VMmark* benchmark is designed for VMware ESX servers; (vi) Most works use complex mathematical optimization tools, which have high overhead. The performance modeling greatly depends on system configuration, and changes to system configuration may require model retraining, which in turn becomes hugely time consuming process due to high overhead; (vii) Many works deal with theoretically derived model of the Cloud and simulation is used for verification. Those simulations often do not take virtualization into consideration, hence does not always portray the reality.

It is clear that, a new design for the consolidation benchmark is required, to address above limitations. The *Incremental Consolidation Benchmarking Method* (ICBM) overcomes above issues, using micro and syntactic benchmarks. Some of the design features of the ICBM are discussed next, in order to explain how does it overcome above issues:

1) Micro and syntactic benchmarks suites have been used instead of parallel applications. This gives the ability to manipulate basic resources, like CPU, memory and I/O, both individually and granularly during experiments. This makes it possible to analyze the effect of consolidation on each resource type more discretely than the previous works;

2) The ICBM, at its core is agnostic to both virtualization technology and system configuration. First and foremost, it is

 ISBN: 978-1-61208-460-2

a methodology that can be applied to any system in general, making it suitable to compare a wide range of systems;

3) The TETV prediction models for combinations of resources have been built from profiled vm data. Separately collected TETV data due to basic resources, like CPU, memory and I/O, have been used to predict TETV for combination of resources, like CPU-Memory, CPU-I/O and Memory-I/O. The prediction results have a good level of accuracy, demonstrating that profiling for every combination of resource load is not necessary. It can potentially save a huge amount of time while profiling a large data center;

5) All experiments have been done on actual virtualized servers rather than using simulators. All results presented here are real system data. The results show that, the ICBM can predict TETV of real virtualized systems quite accurately;

4) Prediction models have been built using the *Least Square Regression* (LSR), which has very low overhead. Use of LSR instead of a complex mathematical tool, makes the training and prediction process much faster. Often, changes in system configuration may require retraining of models, in such cases a low overhead process can save a lot of time;

5) Analysis of profiled data reveals some interesting patterns. For example, it shows that certain types of resource combinations can cause the task execution time of consolidated vms to degrade more rapidly than the others. This indicates that resource contention is one of the main factors, behind the average utilization of virtualized servers being so low.

The micro and syntactic benchmarks suites play an important part in the design of the ICBM. They are important tools for server performance analysis, and a lot of studies have been done on their design. These benchmarks are the result of long time analysis of commercially successful applications. They are inspired by an interesting phenomenon that, the applications spend both 80% of their time and resources, executing just 20% of the code [20]. The micro and syntactic benchmark suites are carefully crafted to mimic such important parts rather than running the entire application. Each benchmark suite consists of several individual applications. These applications are grouped together in a well-thought-out pattern.

The benchmarks suites are used here, to get a finer control on the individual server resource types. Without using these benchmark suites, such controlling of resources is not possible. Experimental results show that, the benchmark suites can cause significant amount of resource contention and TETV on vms. Thus, the benchmark suites can be a set of powerful tools for studying the performance variation of virtualized servers. The experiments that are done here, are very comprehensive and provide some interesting results.

Rest of the paper is structured as follows. The main objectives of experiments are discussed in the Section II, followed by the methodology described in the Section III. Sections III-B and III-A briefly discuss the benchmarks used in the experiments and experiential setup, respectively. Detail results of experiments and prediction are given in the Section IV. Finally, Section V concludes the paper, with an overview of the future work.

II. Problem Description

This section describes the objectives of the experiments. The first objective, is to quantitatively measure the effect of consolidation on the task execution time of vms. The logical view of a virtualized server is shown in the Figure 1a. Here, a typical situation has been considered. Usually, a physical server hosts vms of the same configuration, however number of simultaneously running vms may change over time. Different vms may also have different types of resource requirements. Some of them may be CPU or memory intensive, while others may be multiple resources intensive. As the number of vms increase or decrease on a physical host, it is important to know the TETV of individual vms. The objective here, is to offer a systematic way to record the TETV due to different numbers of vms, and find a way to predict the TETV.

The second objective, is to establish a relationship between TETV due to basic types of resource contention and that of combination of resources. Figure 1b depicts the situation of a server from resource usages point of view. In each graph, the x-axis represents the number of vms simultaneously running on the host. The y-axis represent the task execution time of a vm. As the number of co-located vms increase, the task execution time starts to vary.

The actual amount of variation depends on types and amount of resource loads. For three basic resources types, namely CPU, memory and I/O (Figure 1b(i-iii)), the amount of TETV are excepted to be different. Again for combination of resources, like CPU-memory, CPU-I/O and memory-I/O (Figure 1b(iv-vi)), the TETV would be different, too. Profiling a virtualized system is difficult for many reasons. A server may be running different number of vms, with different amount of loads at different points. That makes it impractical, to profile all vms, for all combination of resources. Establishing a relation, among the TETV due to basic resource types and that of combination of resources, would save a lot of time while profiling a large data center. Next, the procedure used in the ICBM is discusses in details.

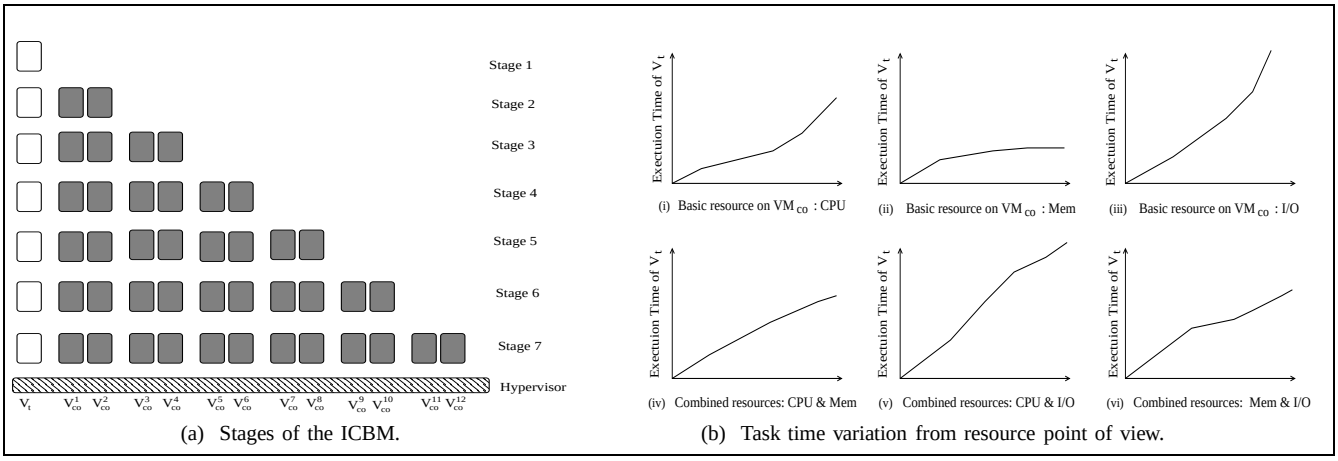
III. Methodology

This section introduces the terminologies and methodology that have been used for rest of the paper. One of the vms of host, is designated as the target vm (v_t), while rest are designated as co-located (v_{co}) vms. The v_t has been used to run different tasks to record their TETV. On the other hand, v_{co} have been used to collectively create resource contention.

In Cloud, the data is generally stored on dedicated nodes over the network. However, before processing, all data is retrieved into the local node. For example, a MapReduce worker node performs all the mapping and reducing steps on local data. In fact, MapReduce nodes rely more on local data, and try to consume as little bandwidth as possible [21]. That way, local I/O contention has much more impact compared to that of network I/O. What is more, in order to address the overall I/O contention issue, it is necessary to address the local I/O contention first [22]. This work provides a quantitative way to measure the I/O contention of vms of a node. The network I/O system consists of several such nodes. Taking this work as a basis, the issue of network I/O contention can be addressed through extended experiments.

A parallel application can have many different data flow paths. As mentioned in the introduction, such an application can be decomposed into a set of tasks. Then, those tasks need to be scheduled on different vms individually. This work lays ground, for understanding the effect of consolidation at vm

 ISBN: 978-1-61208-460-2



(a) Stages of the ICBM. (b) Task time variation from resource point of view.

Figure 1. Experimental setup and objective of ICBM.

level. Without this understanding, it is not possible to formulate the effect on consolidation at a higher level.

The micro and syntactic benchmarks used here, are actually application suites. They are combinations of several applications, patterned together to execute such a way that they put maximum pressure on a particular system resource. For example, the Filebench includes several different categories of tests, including a file-server test [23]. Similarly, the Unixbench consists of many tests, including the System Call Overhead, Pipe-based Context Switching and Execl Throughput test [24]. The Nbench includes, a task allocation algorithm, Huffman compression, IDEA encryption, Neural Net and LU Decomposition test [25]. There are many more test components on those suites, and all of them are designed to mimic the steps of commercial applications. Thus, together they can put the vms under stress like a real application. Also the experimental results from real systems show that, these benchmarks cause the task execution time to vary significantly. Thus, they are well capable of creating resource contention like a real application.

Initially, v_t run one task alone on the host (stage 1 in Figure 1a) and execution finish time is recorded. In successive stages, co-located vms are added in a particular pattern to create resource contention. Manipulating each co-located vm, independently at each step, makes it possible to increase a particular type of resource load at a desired quantity. In the experimental setup, two v_{co} are added at each stage until the server reaches the saturation point. Depending on the configuration, a host can simultaneously run only a certain number of vms, without becoming unresponsive. That is considered to be the saturation point. As co-located vms with different resource types are added at each stage the TETV of all vms are profiled.

Figure 2 provides the pseudo code of ICBM for basic resource types. The ICBM repeats the same steps for each resource type, on co-located vms. Therefore, it is not necessary to explain the steps, for each resource type. Next, these steps are explained for one type of resource load, the CPU load.

Let t be a task whose TETV due to CPU intensive co-located vms is going to be investigated. First, the t is run alone on a single vm (v_t) of the host (stage 1 of Figure 1a), in order to obtain the execution finished time, without any interference from co-located vms. Next, (stage 2) the t is run again, this time along with two simultaneously running co-located vms (v_{co}^1 and v_{co}^2). Both the new vms, run one B_{cpu} type benchmark each, thus only increasing CPU intensive load on the system. This gives the execution time of t on v_t, which is now consolidated with two co-located CPU intensive vms. Afterwards, two more CPU intensive v_{co} are added (stage 3), increasing the number of CPU intensive co-located vms to four. The execution finish time of v_t is profiled again for this setting. This way, two new v_{co} are added at each stage, until the co-located vms stop responding.

```
T ← A set of tasks.
B_cpu ← A set of CPU intensive benchmarks.
B_mem ← A set of memory intensive benchmarks.
B_i/o ← A set of I/O intensive benchmarks.
v_t ← A target vm.
v_co ← A set of co-located vms.
for Each task t ∈ T do
    Add v_t to the host.
    Run t on v_t alone and record the execution finish time.
    for Each benchmark type B ← B_cpu, B_mem, B_i/o do
        while All the vms of host are responding do
            Add two extra v_co with benchmark of type B to host.
            Run v_t simultaneously with the added v_co, and
            record the execution finish time of each.
        end while
        Remove all v_co and free related system resources.
    end for
end for
```

Figure 2. Basic steps of ICBM.

The maximum number of vms, that can be simultaneously run on a host without making it non-responsive, is dependent on the system configuration. In the experiments, the host had one *Intel i7-3770* processor, with four cores and eight hardware threads. Assigning, a logical CPU to each vm, the system could run maximum of fourteen such vms, simultaneously. Adding anymore vm, would made the whole system non-responsive. With one vm designated as v_t, and two more new v_{co} added at each stage, the final stage (stage 7) of the experiment had thirteen simultaneously running vms (one v_t along with twelve v_{co}). The vms are created with least possible subset of CPU resources, so that CPU load can be increased with fine granularity. However, vms with larger number of logical CPU

can be also created, if such is required.

The same steps are repeated for memory intensive v_{co}, too. However, in this case memory intensive benchmarks are run on co-located vms instead of CPU intensive benchmarks. All vms are configured to have 1 GB of RAM. The experiment starts (stage 1) with a single vm (v_t) running the task, whose TETV due to memory intensive co-located vms is going to be investigated. Next, (stage 2) v_t is run again, along with two new co-located vms (v_{co}^1 and v_{co}^2), each running one memory intensive benchmark. Similarly, two more vms (v_{co}^i and v_{co}^{i+1}) are added at each stage until the system reaches a predetermined point of memory load. In this case, adding a new v_{co} makes the host memory load to be increased by 1 GB. It was done so that, the host memory load can be changed granularly. The host has 16 GB of RAM. By restricting the number of vms to thirteen at the final stage, maximum of 13 GB RAM is allocated to the co-located vms, leaving rest for the hypervisor. As with the CPU load, this predetermined load is also not a fixed number. Afterwards, for I/O load the same steps are repeated by adding two I/O intensive benchmarks on two v_{co}, at each stage. Thus, the TETV for three basic resource types (CPU, memory and I/O) are collected.

Next, the procedure is repeated for **resource combinations**. The combinations are made by choosing two resource types at a time, from previously mentioned basic three. Those are CPU-Memory, Memory-I/O and I/O-CPU. Experiments for combination of loads are done exactly the same way. That is, start with one vm (v_t), and at each stage add two co-located vms (v_{co}^i and v_{co}^{i+1}) to increase the load. Difference is that, now two new vms run two different types of benchmarks. Both of them, together create the effect of load combinations. For example, to increase CPU-Memory load, two v_{co} are added at each stage. One (v_{co}^i) runs a CPU intensive benchmark, while the other one (v_{co}^{i+1}) runs a memory intensive benchmark.

Above experiments demonstrate, how the execution time of v_t is varied due to co-located vms (v_{co}^i). However, there is another angle to this problem, that is how the v_{co} are collectively effect the execution times of each other. To examine this, the execution finish times of all the v_{co}^i, are also recorded at each stage. Finally, the whole procedure is repeated without the v_t altogether. That is, all the experimental steps are repeated, by adding only load (on v_{co}^i) at each stage.

Advantage of the ICBM is that, the server load can be changed granularly, for each basic resource type or their combinations. Furthermore, there is no need to know the internal structure of a complex application, which is running on the vm. Establishing a relationship between task execution time and the server load would be helpful for a virtualized data-center in many ways, including designing heuristic algorithms for scheduling, consolidation and live-migration of vms [26][27].

A. Experimental setup

A Dell XPS-8500 system has been used in the experiments. It has one Intel i7-3770 processor and 16GB of RAM. The i7-3770, has four cores and eight hardware threads, each clocked at 3.4 GHz. The host is deployed with Xen 4.2 over Fedora 17 core. Fourteen vms have been setup, one to acts as v_t, while the rest are v_{co}^i. Each vm run Fedora 16, have one logical CPU, 1 GB of RAM and 30 GB virtual disk. The vms are not pinned to the logical cores. Total of 13 GB RAM is allocated to the vms, rest are available for the hypervisor. All the benchmarks are installed on vms beforehand, and a concurrent java application manages all benchmarks and vms from a remote node.

B. Benchmarks used

This section gives an overview of the benchmark suites, used in the experiments. A data center may be running thousands of servers. While designing an algorithm for consolidation or live migration scheme, the internal structures of all the parallel applications may not be known. It is more realistic to characterize the servers by their loads. Micro and syntactic benchmark suites are well suited for such purposes.

Three different categories of benchmarks have been used for three categories of resources, namely CPU, memory and I/O. The *Nbench* [25] and *Unixbench* [24], are the two CPU intensive benchmark suites used here. Two memory benchmark suites used here, are the *Cachebench* [28] and *Stream* [29][30]. Finally, three I/O benchmark suites have been used, they are the *Dbench* [31], *Filebench* [23] and *Iozone* [32]. For each benchmark, several different parameters are need to be set. Owing to the space limitation, it is not possible to list all the parameters here. A fuller discussion about the benchmarks, is out of scope for this paper. Interested readers are encouraged to refer to the citations of respective benchmark suites for details.

IV. RESULTS

The results for prediction are given in the Section IV-C. However, to interpret the prediction results, it is necessary to discuss some observations of the experimental results. It will also help to clarify the training and testing data format, which has been used during the training and prediction phrases.

A. Execution time variations of the target vm (v_t)

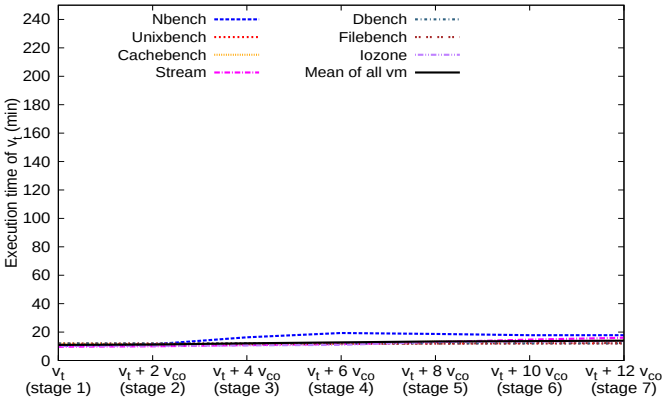
Three graphs of Figure 3, show the TETV of v_t due to three basic types of resource loads. They are being, CPU (Figure 3a), memory (Figure 3b) and I/O (Figure 3c). In each graph, the x-axis shows the total number of vms running on the system, while y-axis presents the task completion time on v_t. First point of the graph, shows the execution time of v_t, when it is run alone on the server. At this stage v_t is free from any interference from co-located vms. As explained in the Section III, two co-located vms (v_{co}) are added to the server at successive stages. In the final stage, there were twelve v_{co}, simultaneously running besides the v_t. In other words, from left to right along the x-axis, the interference from co-located vms increases. The first point of each graph, gives execution time of the task without any interference from co-located vms. On the other hand, the last point gives the execution time with maximum interference from co-located vms. Results clearly show that different types and number of v_{co}, make the task execution time to vary at a different rate.

Figure 3a shows that, the TETV of v_t with the increase of CPU intensive v_{co}^i, is the minimum among all types of workloads. On the other hand, for memory intensive v_{co} (Figure 3b), two CPU intensive tasks (Nbench and Unixbench) on v_t show less variation compared two memory intensive tasks (Cachebench and Stream) under the same situation. As an example, in isolation the Nbench takes 10.1 minute to execute on v_t. With other 12 memory intensive co-located vms (v_{co}) it extends to 12.83 minutes, which is only 27.02% longer. In contrast, the Cachebench under the exact same setup takes 587.58% longer to execute (execution time goes from 11.76

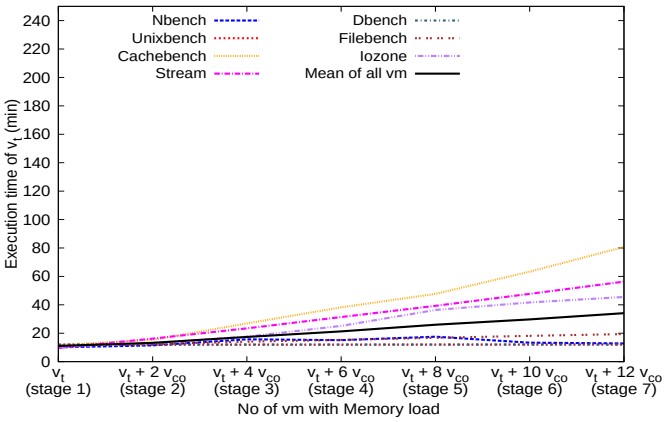
 ISBN: 978-1-61208-460-2

min to 80.68 min). Figure 3c shows the TETV due to I/O intensive v_{co}. Here, CPU intensive tasks do not show much performance degradation, while both the memory and I/O intensive tasks do. For example, the Cachebench (a memory intensive task) have 1057.311% increase in execution time, while Iozone (an I/O intensive task) have 1482.93%.

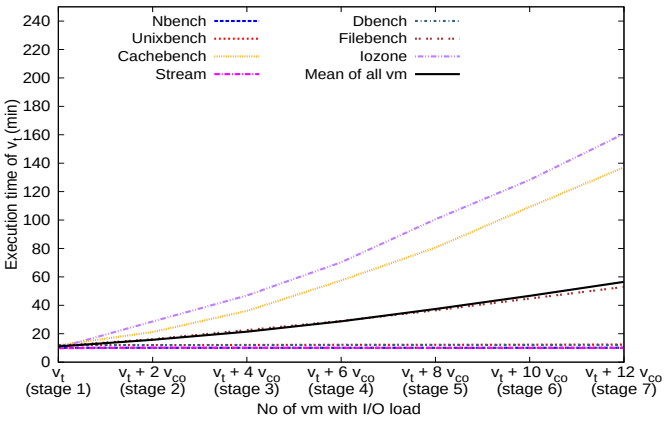
Next, Figure 4 shows the TETV on v_t, when combination of loads have been used on v_{co}. The combinations are being CPU-Memory (Figure 4a), CPU-I/O (Figure 4b) and Memory-I/O (Figure 4c). Figure 4a shows the TETV on v_t due to a



(a) Basic resource type: CPU.

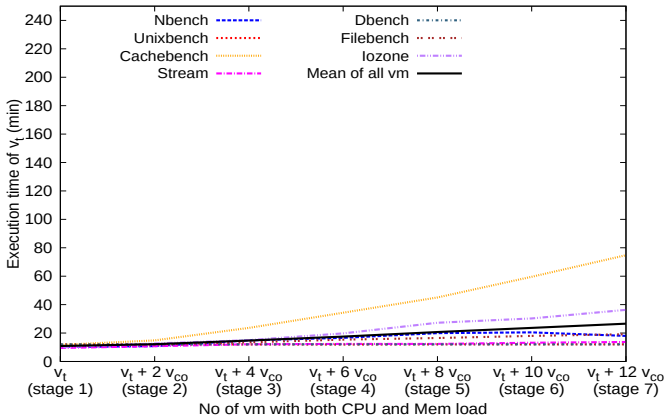


(b) Basic resource type: Memory.

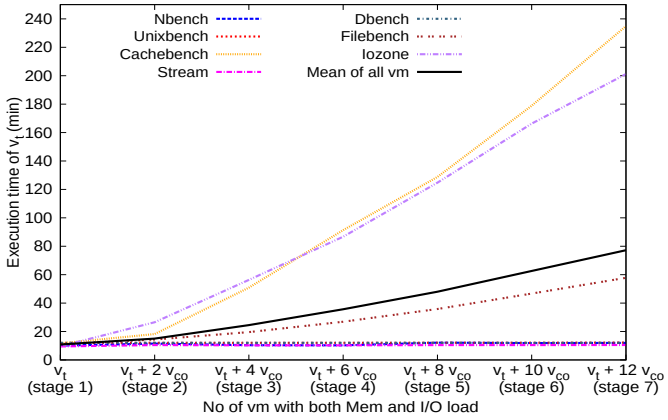


(c) Basic resource type: I/O.

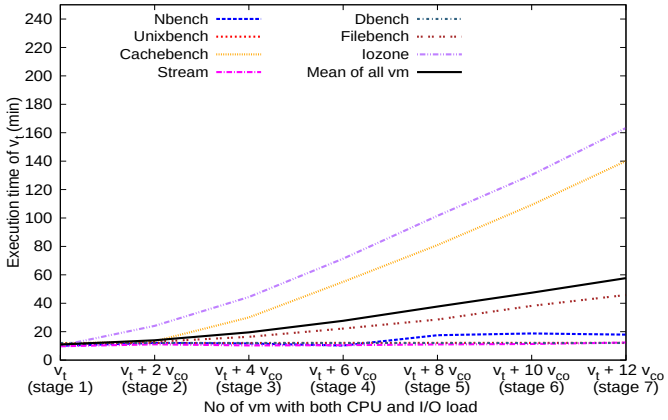
Figure 3. The TETV of v_t due to various basic types of resource load on v_{co}.



(a) Combination of resources: CPU-Memory.



(b) Combination of resources: Memory-I/O.



(c) Combination of resources: CPU-I/O.

Figure 4. The TETV of v_t due to different combinations of resource load on v_{co}.

mix of CPU and memory intensive v_{co}. Here, CPU intensive tasks on v_t show least amount of degradation just as observed previously. Among the memory intensive benchmarks, the Cachebench shows highest rate of performance degradation (542.74%). However, for I/O intensive tasks the effect of CPU-Memory v_{co} combination, is less adverse compared to memory intensive tasks. Figures 4c and 4b, show the performance degradation of v_t, when I/O intensive v_{co} is coupled with CPU and memory intensive v_{co}, respectively.

In both cases, memory and I/O intensive tasks on v_t show comparatively more performance degradation. In the case

 ISBN: 978-1-61208-460-2

of CPU-I/O load (Figure 4c), the Cachebench and Iozone show increase of execution time by 1907.21% and 1991.67%, respectively. Again for Memory-I/O load (Figure 4b), the same two benchmarks show worse performance degradation (1100.09% and 1502.56%, respectively). Table I shows the *standard deviation* (SD) of execution times of all seven tasks on v_t through stage 1 to 7.

TABLE I. STANDARD DEVIATION OF TASK EXECUTION TIMES (MIN).

	Task type	CPU intensive		Mem. intensive		I/O intensive		
	Load type	Nbench	Unixbench	Cachebench	Stream	Dbench	Filebench	Iozone
Basic	CPU	3.66	0.05	0.99	2.42	0.00	0.41	1.51
	Memory	2.58	0.05	25.28	16.89	0.01	2.97	14.18
	I/O	0.01	0.15	46.59	0.12	0.02	15.16	54.70
Combined	CPU-Memory	4.12	0.05	23.51	1.43	0.00	3.06	9.89
	CPU-I/O	0.88	0.08	83.85	0.35	0.01	17.33	71.62
	Memory-I/O	3.82	0.06	49.40	0.82	0.01	13.17	56.62

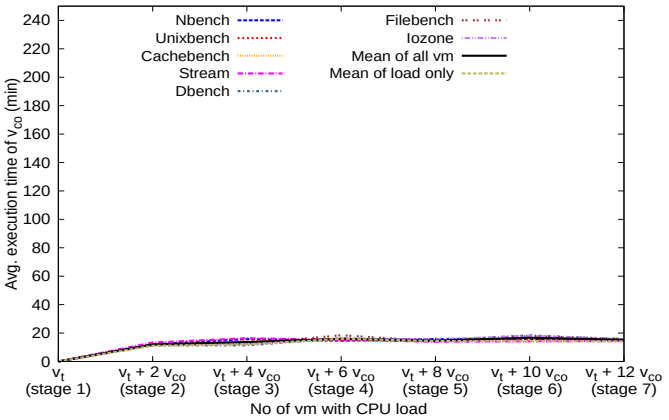
The results of this section show, how the degradation of task execution time for each resource can be presented in a quantitative way. At present, there is no standard way to compare the effect of consolidation, for different types of resources or different classes of servers. The straightforward method presented here, can be used to compare the effect of consolidation on wide varieties of systems.

B. Execution time variations of the co-located vms (v^i_{co})

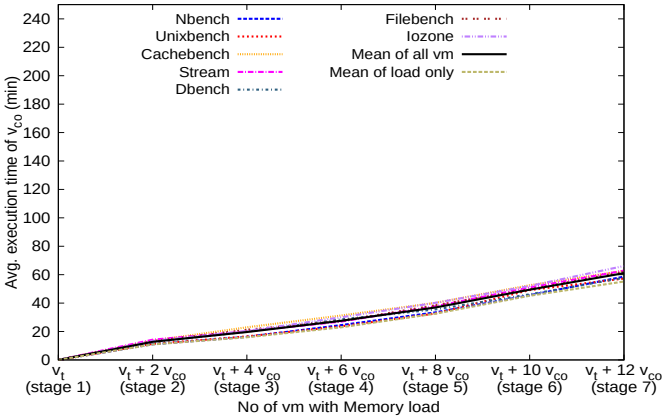
Execution finish time of all the co-located vms, at each stage are also profiled. This data is used for predicting the TETV of v_t. Figures 5 and 6 show the arithmetic mean of execution times of all v^i_{co}, at each stage, separately. During experiments, it was observed that, the arithmetic mean of execution time of all the v^i_{co} follow a pattern, for each resource type. Even though, individual v^i_{co} execution time might not have such characteristic. This, clearly indicates that overall resource usages of all vms, is what ultimately shapes the performance degradation curve during consolidation.

Both in Figures 5 and 6, the first point of each graph is always zero, as there is no v^i_{co} running on the host at stage 1 (see Figure 1a). At stage 2, two v^i_{co} are running, therefore second point is the arithmetic mean of task execution times of two vms (v^1_{co}, v^2_{co}). Similarly, third point is the arithmetic mean of values of four vms (v^1_{co}, v^2_{co}, v^3_{co} and v^4_{co}) and so on.

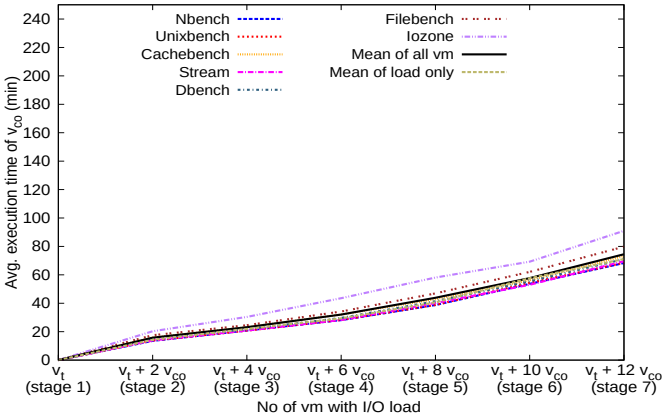
Using above procedure, each subgraph has seven arithmetic mean variation plotting of v_{co}, for seven different tasks running on v_t. Furthermore, arithmetic mean of those seven values are also shown (in black). Increase of different types of workloads causes the arithmetic mean of execution times to change differently. It can be seen that, the variation is the minimum for CPU intensive load (Figure 5a), among all three basic types of resources. Among the three combination of resources, the CPU-Memory (Figure 6a) intensive v_{co} combination shows least amount of performance degradation. On the other hand, the combination of Memory-I/O intensive v_{co} (Figure 6b) have



(a) Basic resource type: CPU.



(b) Basic resource type: Memory.



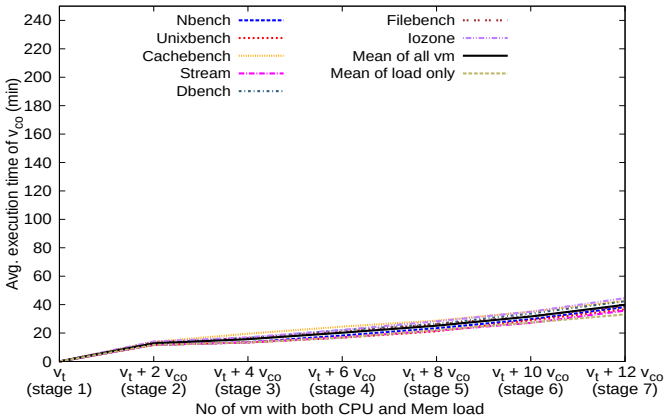
(c) Basic resource type: I/O.

Figure 5. Arithmetic mean of TETV of v_{co} with respect to three basic resource types load changes.

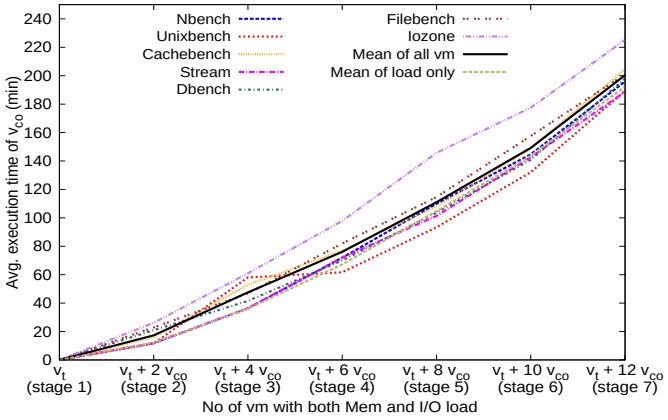
debilitating effect on the system as the arithmetic mean of execution time rises rather quickly, compared to other cases.

In order to obtain above execution time values, each experiment has been repeated at least three times. The order of adding vms has been shuffled, in order to observe their effect. However, no significant difference between the results have been observed. Two vms are added at each stage, only to conduct the experiments in a uniform way. In our observation, the order of adding vms do not change the results ultimately, rather it depends on the cumulative resource usages of vms.

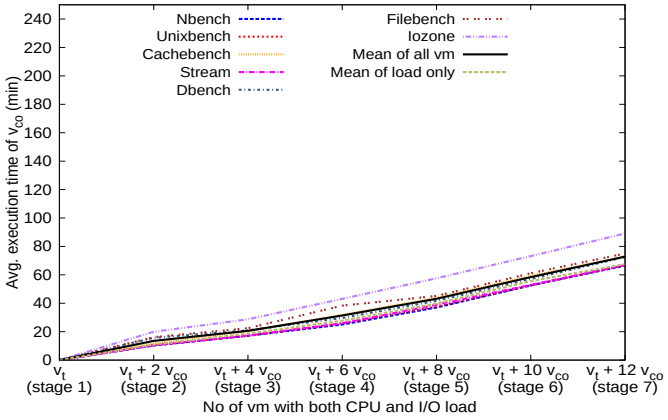
 ISBN: 978-1-61208-460-2



(a) Combination of resources: CPU-Memory.



(b) Combination of resources: Memory-I/O.



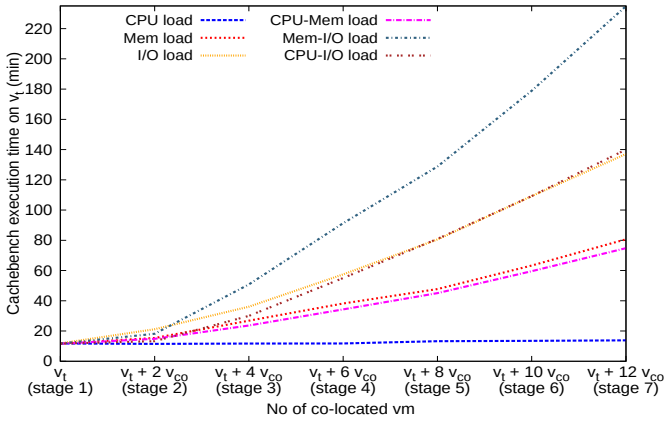
(c) Combination of resources: CPU-I/O.

Figure 6. Arithmetic mean of TETV of v_{co} with respect to three combinations of resource load changes.

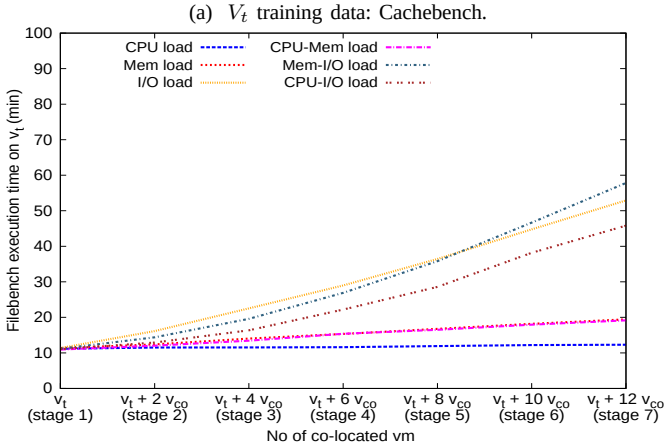
C. Task execution time variation prediction

Four benchmarks have been used as tasks for training. They are the Nbench, Unixbench, Cachebench and Stream. Three other benchmarks have been used for testing. They are the Dbench, Filebench and Iozone. All of the test benchmarks have different levels of performance degradation. Therefore, they can help to evaluate the prediction process better. Training and testing have been done on different sets of data. No training data have been used for testing, and vice versa.

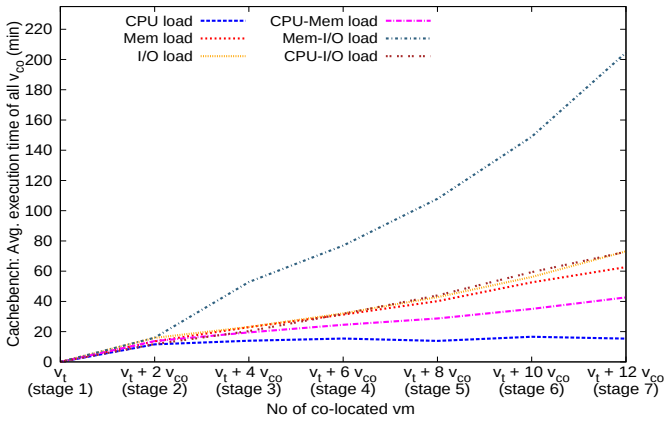
The nine subgraphs of Figure 8, separately show prediction results for three resources of three test tasks on v_t. Each subgraph, contains two sets of predictions, obtained from two separate data sets, which are described next. First set of predictions, are shown in blue on Figure 8. In this case, the TETV data of v_t for basic resource types, has been used to predict TETV of v_t for combination of resources. First, the TETV data of v_t for CPU (Figure 3a), Memory (Figure 3b) and I/O (Figure 3c) intensive v_{co} are recorded, separately. Those are used as inputs. Then, three resource combinations have been used as three separate targets, which are CPU-Memory (Figure 4a), CPU-I/O (Figure 4c) and Memory-I/O (Figure 4b).



(a) V_t training data: Cachebench.

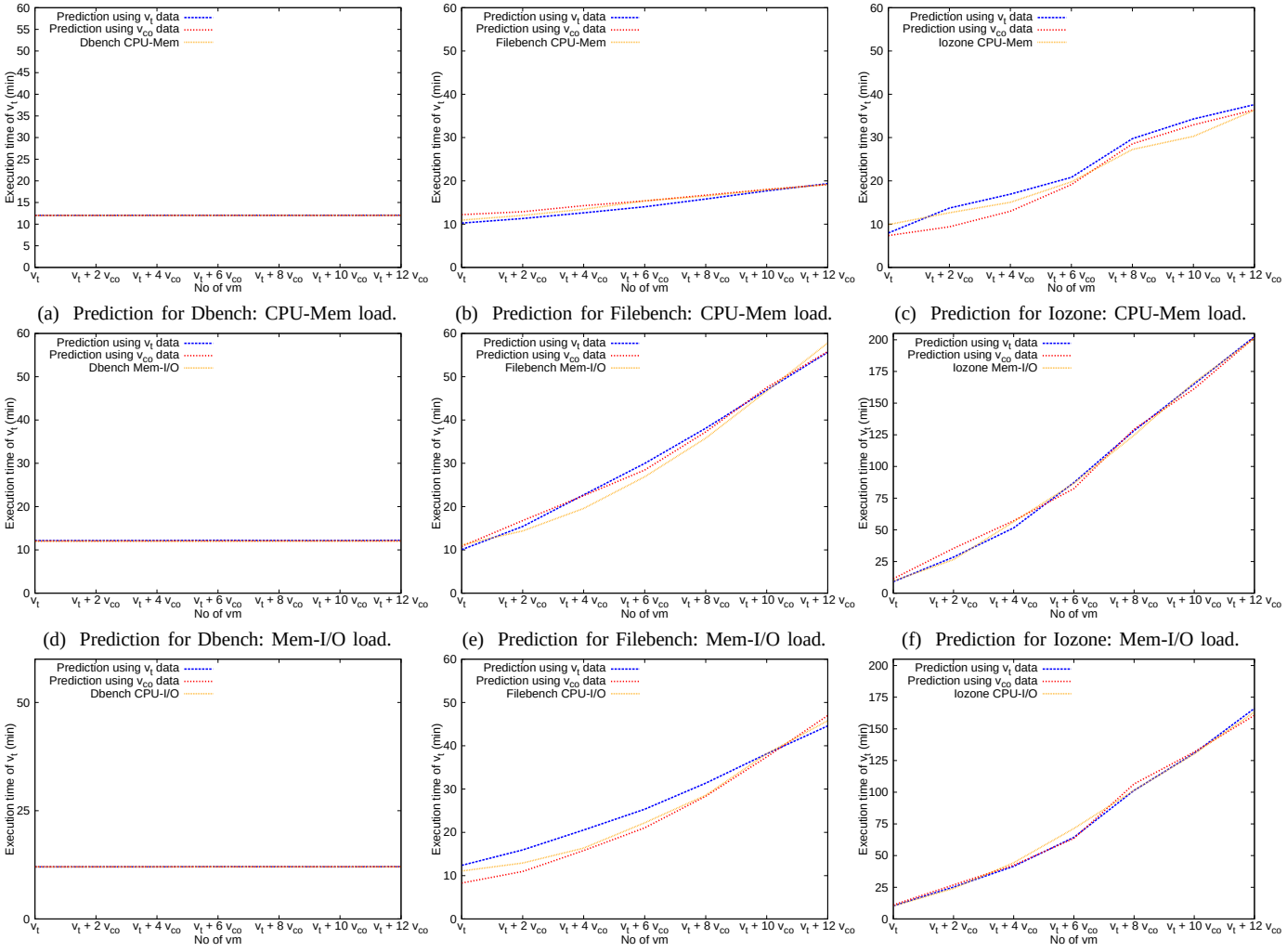


(b) V_t & V_{co} testing data: Filebench.



(c) V_{co} training data: Cachebench.

Figure 7. Examples of input and target data used for both training and testing.



(a) Prediction for Dbench: CPU-Mem load. (b) Prediction for Filebench: CPU-Mem load. (c) Prediction for Iozone: CPU-Mem load.

(d) Prediction for Dbench: Mem-I/O load. (e) Prediction for Filebench: Mem-I/O load. (f) Prediction for Iozone: Mem-I/O load.

(g) Prediction for Dbench: CPU-I/O load. (h) Prediction for Filebench: CPU-I/O load. (i) Prediction for Iozone: CPU-I/O load.

Figure 8. Task execution time prediction from v_t and v_{co} data.

Input & target data example. In above Figures 3 and 4, data is grouped according to resource loads. It is done to demonstrate that, different types of resources influence the task execution time differently. Thus, it is important to study the effect of each resource type separately. However, for training and prediction, input data needs to be grouped benchmark wise. It is required, because during training, the TETV data due to basic resources of a task, needs to be linked to the TETV data due to combination of resources of the same task. Otherwise, training would not be done with a consistent data.

Figure 7a shows an example, of this rearranged input and target data set that is used for training. It combines the TETV data of the Cachebench on v_t, for three basic and three combined types of v_{co} loads from Figures 3 and 4, respectively. All benchmarks data are rearranged similarly. During training, three basic resources data (CPU, Memory and I/O) are used as inputs and three combination of resources data (CPU-Memory, CPU-I/O and Memory-I/O) as targets. From the training data set, three sperate models have been generated, for three sets of target data (CPU-Memory, CPU-I/O and Memory-I/O). All three targets use the same three basic resources data as inputs.

An example of test data set is shown in Figure 7b. It combines six TETV data of the Filebench, from Figures 3 and 4. Prediction results for three resources (CPU, Memory and I/O) of three test benchmarks (Dbench, Filebench and Iozone) are shown in Figure 8. The *root mean square error (RMSE)* for this set of predictions are shown in Table II.

Training with c_{co} data. The second set of predictions (shown in red on Figure 8), are obtained by training the models only with v_{co} data. This demonstrates an interesting phenomenon, that models generated through use of co-located vm (v_{co}) data only, can predict the execution time variations of target vm (v_t), too. During training the CPU (Figure 5a), memory (Figure 5b) and I/O (Figure 5c) data of v_{co} are used as inputs, while CPU-Mem (Figure 6a), Mem-I/O (Figure 6b) and CPU-I/O (Figure 6c) data of v_{co} used as targets. On the other hand, during testing the v_t data are used as both input and target. All testing have been done with the same three benchmarks (Dbench, Filebench and Iozone), that have been used in the previous section for testing. That is, in this case models are generated using only co-located vms data, while testing is done with target vm data.

 ISBN: 978-1-61208-460-2

$$T^t_{cpu-mem,i} = -20.415 + 4.795 * (T^t_{cpu,i})^{0.7} + 0.752 * (T^t_{mem,i})^{0.9} + 0.579 * (T^t_{io,i}) \\ -0.119 * (T^t_{cpu,i})^{0.7} * (T^t_{mem,i})^{0.9} - 0.035 * (T^t_{cpu,i})^{0.7} * (T^t_{io,i}) + 0.002 * (T^t_{mem,i})^{0.9} * (T^t_{io,i}) \quad (1)$$

$$T^t_{cpu-io,i} = -16.393527 + 0.802 * (T^t_{cpu,i}) + 0.282 * (T^t_{mem,i}) + 1.769 * (T^t_{io,i}) \\ -0.020 * (T^t_{cpu,i}) * (T^t_{mem,i}) - 0.023 * (T^t_{cpu,i}) * (T^t_{io,i}) + 0.003 * (T^t_{mem,i}) * (T^t_{io,i}) \quad (2)$$

$$T^t_{mem-io,i} = -16.549 + 2.104 * (T^t_{cpu,i}) + 7.920 * (T^t_{mem,i})^{0.2} - 0.169 * (T^t_{io,i})^{0.96} \\ -0.957 * (T^t_{cpu,i}) * (T^t_{mem,i})^{0.2} + 0.031 * (T^t_{cpu,i}) * (T^t_{io,i})^{0.96} + 0.455 * (T^t_{mem,i})^{0.2} * (T^t_{io,i})^{0.96} \quad (3)$$

Figure 9. TETV models for resource combinations (CPU-Memory, CPU-I/O & Memory-I/O) built from v_t data.

$$T^t_{cpu-mem,i} = -0.009 + 0.190 * (T^{co}_{cpu,i})^{1.2} + 0.220 * (T^{co}_{mem,i})^{1.2} + 0.250 * (T^{co}_{io,i}) \quad (4)$$

$$T^t_{cpu-io,i} = -0.102 + 4.217 * (T^{co}_{cpu,i})^{3.25} - 0.034 * (T^{co}_{mem,i}) + 1.383 * (T^{co}_{io,i})^{0.7} \\ +0.544 * (T^{co}_{cpu,i})^{3.25} * (T^{co}_{mem,i}) - 5.394 * (T^{co}_{cpu,i})^{3.25} * (T^{co}_{io,i})^{0.7} + 0.163 * (T^{co}_{mem,i}) * IO^{0.7} \quad (5)$$

$$T^t_{mem-io,i} = 3.720 * (T^{co}_{cpu,i})^{0.325} + 2.376 * (T^{co}_{mem,i}) - 0.020 * (T^{co}_{io,i})^{0.05} \\ +1.079 * (T^{co}_{cpu,i})^{0.325} * (T^{co}_{mem,i}) - 4.394 * (T^{co}_{cpu,i})^{0.325} * (T^{co}_{io,i})^{0.05} - 3.144 * (T^{co}_{mem,i}) * (T^{co}_{io,i})^{0.05} \quad (6)$$

Figure 10. TETV models for resource combinations (CPU-Memory, CPU-I/O & Memory-I/O) built from v_{co} data.

An **example** of the training data, used here is shown on Figure 7c. It is a collection of six different v_{co} workload TETV data of the Cachebench, from six graphs of Figures 5 and 6. During training, the variation of arithmetic mean of v_{co} TETV due to CPU, memory and I/O are used as inputs. The CPU-Memory, Memory-I/O and CPU-I/O data are used as targets. Three models have been generated in this way, with three v_{co} target data set. However, testing is done with v_t test data like, the example shown in the Figure 7b. Recall that, it was also used for testing in the previous section. Table III shows the RMSE for this set of prediction. It shows that, the prediction results have a good level of accuracy.

Root mean square error (RMSE) for prediction. The RMSE of predictions for above two sets of data, are shown separately on Table II and Table III. To the best knowledge of the authors, there exist no approximation algorithm to estimate the TETV of co-located vms. Therefore, there is no acceptable theoretical bound for the RMSE values [33]. Generally, a lower value of RMSE means better prediction accuracy.

In each case, the prediction is better, when all three basic resources have been used to generate the model rather than two. For example, while predicting the TETV of CPU-Memory load combination, the model build using CPU, memory and I/O (all 3 basic resources) data produces better results, than that generated by using only CPU and memory (2 resources) data.

Parametric model. Lastly, the Figures 9 and 10 show model parameters and coefficients, as they are trained from profiled v_t and v_{co} data, respectively. They demonstrate the relation between input and prediction data formats. Three equations of the Figure 9, are for three resource combination targets. The terms are self-explanatory. For example, $T^t_{cpu-mem,i}$ denotes predicted task execution time of a task on v_t, when it is consolidated with total of i number of CPU and memory intensive co-located vms. Similarly, $T^t_{cpu-io,i}$ and $T^t_{mem-io,i}$ represent the predicted task execution time for CPU-I/O and Memory-I/O load combinations, respectively. The same input parameters have been used for all equations. $T^t_{cpu,i}$ represents the task execution time on v_t when server is consolidated with i number of CPU intensive co-located vms. Similarly, $T^t_{mem,i}$ and $T^t_{io,i}$ represent the task execution time when the server is consolidated with the same number of memory and I/O intensive co-located vms, respectively.

The equations of Figure 10 have the same targets, however inputs are different. Here, arithmetic mean of execution times of co-located vms have been used as inputs. For example, $T^{co}_{cpu,i}$ represents the arithmetic mean of execution times of i number of CPU intensive co-located vms. In this case, arithmetic mean of execution times of co-located vms have been used to predict the TETV of target vm.

TABLE II. ROOT MEAN SQUARE ERROR (RMSE) FOR PREDICTION USING TARGET VM (v_t) DATA.

	CPU-Memory Prediction		CPU-I/O Prediction		Mem-I/O Prediction	
Benchmarks	With all 3 resources: CPU, Mem, I/O	Only 2 resources: CPU, Mem	With all 3 resources: CPU, Mem, I/O	Only 2 resources: CPU, I/O	With all 3 resources: CPU, Mem, I/O	Only 2 resources: Mem, I/O
Dbench	0.036	2.516	0.146	0.506	0.011	0.468
Filebench	0.771	2.628	2.131	11.933	2.605	3.022
Iozone	**2.193**	10.505	**2.475**	21.566	**3.083**	3.588

TABLE III. ROOT MEAN SQUARE ERROR (RMSE) FOR PREDICTION USING CO-LOCATED VM (v_{co}) DATA.

	CPU-Memory Prediction		CPU-I/O Prediction		Mem-I/O Prediction	
Benchmarks	With all 3 resources: CPU, Mem, I/O	Only 2 resources: CPU, Mem	With all 3 resources: CPU, Mem, I/O	Only 2 resources: CPU, I/O	With all 3 resources: CPU, Mem, I/O	Only 2 resources: Mem, I/O
Dbench	0.006	0.317	0.020	17.621	0.014	20.182
Filebench	0.657	4.515	1.841	10.607	1.475	47.797
Iozone	2.083	43.546	4.572	87.725	3.898	59.863

Changes in basic configuration of vms, may require some modification of model parameters. The public Cloud offers vms for renting in certain number of predefined configurations. Therefore, number of such models required in reality, are limited. What is more, since LSR has low overhead, building or rebuilding of a limited number of such models would not be time consuming.

V. Conclusion and Future work

This work addresses an important issue of virtualization, the performance variations due to consolidation. A straightforward methodology has been introduced here, that is practically feasible to implement. It is different from most of the complementary works on virtualization, that employ complex mathematical tools and rely on simulation. In contrast, this work introduces a low overhead prediction process, and results are collected from real virtualized systems.

Micro and syntactic benchmark suites have been used here in a step by step process, to manipulate various vm resources individually. The methodology introduced here is unique, and results prove the effectiveness of this method. Experimental results from real virtualized systems show that, in this way it is possible to predict the TETV of vms quite accurately. It provides a new and quantitative way to explore the mutual performance interference of co-located vms. The results also provide some valuable inside into the nature of resource contention due to consolidation of vms.

The experimental results encourages one to continue working along this direction. For future work, experiments would need to be extended to a wider range of virtualization techniques and server configurations, to derive a more generalized model. More questions related to this method, can be addressed in details in future works.

References

[1] T. Janpan, V. Visoottiviseth, and R. Takano, “A virtual machine consolidation framework for CloudStack platforms,” in ICOIN, 2014, pp. 28–33.

[2] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, and H. Tenhunen, “Utilization Prediction Aware VM Consolidation Approach for Green Cloud Computing,” in CLOUD, 2015, pp. 381–388.

[3] A. Tchana, N. D. Palma, I. Safieddine, D. Hagimont, B. Diot, and N. Vuillerme, “Software Consolidation as an Efficient Energy and Cost Saving Solution for a SaaS/PaaS Cloud Model,” in Euro-Par, 2015, pp. 305–316.

[4] M. Dobber, R. D. van der Mei, and G. Koole, “Effective Prediction of Job Processing Times in a Large-Scale Grid Environment,” in HPDC, 2006, pp. 359–360.

[5] V. E. Taylor, X. Wu, J. Geisler, and R. L. Stevens, “Using Kernel Couplings to Predict Parallel Application Performance.” in HPDC, 2002, pp. 125–134.

[6] W. Gao, Y. Li, H. Lu, T. Wang, and C. Liu, “On Exploiting Dynamic Execution Patterns for Workload Offloading in Mobile Cloud Applications,” in ICNP, 2014, pp. 1–12.

[7] A. Gupta, L. V. Kalé, F. Gioachin, V. March, C. H. Suen, B. Lee, P. Faraboschi, R. Kaufmann, and D. S. Milojicic, “The who, what, why, and how of high performance computing in the cloud,” in CloudCom, Volume 1, 2013, pp. 306–314.

[8] J. Simão and L. Veiga, “Flexible SLAs in the Cloud with a Partial Utility-Driven Scheduling Architecture,” in CloudCom, Volume 1, 2013, pp. 274–281.

[9] S. Xi, M. Xu, C. Lu, L. T. X. Phan, C. D. Gill, O. Sokolsky, and I. Lee, “Real-time multi-core virtual machine scheduling in Xen,” in EMSOFT, 2014, pp. 27:1–27:10.

[10] J. Zhao, J. Tao, L. Wang, and A. Wirooks, “A Toolchain For Profiling Virtual Machines,” in ECMS, 2013, pp. 497–503.

[11] A. Iosup, “IaaS Cloud Benchmarking: Approaches, Challenges, and Experience,” in HotTopiCS, 2013, pp. 1–2.

[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking Cloud Serving Systems with YCSB,” in SoCC, 2010, pp. 143–154.

[13] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, “Profiling Applications for Virtual Machine Placement in Clouds,” in CLOUD, 2011, pp. 660–667.

[14] R. McDougall and J. Anderson, “Virtualization performance: Perspectives and challenges ahead,” SIGOPS Oper. Syst. Rev., vol. 44, no. 4, Dec. 2010, pp. 40–56.

[15] V. Makhija, B. Herndon, P. Smith, E. Zamost, and J. Anderson, “VMmark: A Scalable Benchmark for Virtualized Systems,” VMware, Tech. Rep. TR-2006-002, 2006.

[16] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, “Quantitative comparison of Xen and KVM,” in Xen summit. Berkeley, CA, USA: USENIX association, Jun. 2008.

[17] P. Apparao, R. Iyer, and D. Newell, “Towards Modeling & Analysis of Consolidated CMP Servers,” SIGARCH Comput. Archit. News, vol. 36, no. 2, May 2008, pp. 38–45.

[18] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, “Modeling Virtual Machine Performance: Challenges and Approaches,” SIGMETRICS Perform. Eval. Rev., vol. 37, no. 3, Jan. 2010, pp. 55–60.

[19] H. Jin, W. Cao, P. Yuan, and X. Xie, “VSCBenchmark: Benchmark for Dynamic Server Performance of Virtualization Technology,” in IFMT ’08, 2008, pp. 5:1–5:8.

[20] G. G. Shulmeyer and T. J. McCabe, “Handbook of Software Quality Assurance (3rd Ed.),” G. G. Schulmeyer and J. I. McManus, Eds. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999, ch. The Pareto Principle Applied to Software Quality Assurance, pp. 291–328.

[21] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, 2008, pp. 107–113. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[22] S. Ma, X.-H. Sun, and I. Raicu, “I/O throttling and coordination for MapReduce,” Illinois Institute of Technology, Tech. Rep., 2012.

[23] OpenSolaris Project, “Filebench,” URL: `http://filebench.sourceforge.net/wiki/index.php/Main_Page`, Retrieved: February, 2016.

[24] “Unixbench: BYTE UNIX benchmark suite,” URL: `http://github.com/kdlucas/byte-unixbench`, Retrieved: February, 2016.

[25] C. C. Eglantine, NBench. TypPRESS, 2012, iSBN: 9786136257211.

[26] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, “A Quantitative Study of Virtual Machine Live Migration,” in CAC, 2013, pp. 11:1–11:10.

[27] S. Nathan, P. Kulkarni, and U. Bellur, “Resource Availability Based Performance Benchmarking of Virtual Machine Migrations,” in ICPE, 2013, pp. 387–398.

[28] P. J. Mucci, K. London, and P. J. Mucci, “The CacheBench Report,” URL: `www.earth.lsa.umich.edu/ keken/benchmarks/cachebench.pdf`, Retrieved: February, 2016.

[29] J. D. McCalpin, “Memory Bandwidth and Machine Balance in Current High Performance Computers,” IEEE CS TCCA Newsletter, Dec. 1995, pp. 19–25.

[30] ——, “STREAM: Sustainable Memory Bandwidth in High Performance Computers,” University of Virginia, Charlottesville, Virginia, Tech. Rep., 2007.

[31] A. Tridgell, “The dbench benchmark,” URL: `http://samba.org/ftp/tridge/dbench/README`, 2007, Retrieved: February, 2016.

[32] W. Norcott and D. Capps, “IOzone Filesystem Benchmark,” URL: `www.iozone.org`, Retrieved: February, 2016.

[33] Z. A. Mann, “Allocation of Virtual Machines in Cloud Data Centers&Mdash;A Survey of Problem Models and Optimization Algorithms,” ACM Comput. Surv., vol. 48, no. 1, Aug. 2015, pp. 11:1–11:34.

 ISBN: 978-1-61208-460-2