An Automated Lightweight Framework for Scheduling and Profiling Parallel Workflows Simultaneously on Multiple Hypervisors

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*—This work presents a lightweight framework for performing automated experiments with the execution time and performance variations of parallel workflows. The execution time variation of tasks due to consolidation is a barrier to efficiently scheduling them on *Virtual Machines* (VMs). In data centers, VMs are usually consolidated to increase resource utilization. However, this causes resource contention and performance degradation among the VMs. To address this issue, it is necessary to perform experiments with large numbers of tasks and schedules. There exists no framework particularly designed for this type of experiment. The proposed framework makes it easy to conduct experiments with large numbers of task execution patterns. Moreover, it is capable of profiling the execution time variation of each task of a workflow. The design principles, implementation issues and trade-offs of the framework are discussed in detail here. The effectiveness of the framework is demonstrated with a data-intensive scientific workflow, which processes the *Galactic Arecibo L-band Feed Array HI* (GALFA-HI) survey data with the *Montage* toolkit. With this framework, experiments have been simultaneously run on three different hypervisors and the execution time variation of each task retrieved. The three hypervisors are the *VMware ESXi* 5.5, *XenServer* 6.5 and *Xen* 4.6. This framework will enable researchers to perform large scale experiments with the execution time variations of parallel tasks on multiple hypervisors and the Cloud.**

***Keywords*–Cloud; virtualization; consolidation; performance; scheduling framework.**

I. Introduction

Virtualization plays an important part for both the data centers and Cloud. Among other advantages, it allows consolidation of *Virtual Machines* (VMs) in data centers. To put it simply, consolidated means running multiple VMs simultaneously on the same server through virtualization. It is a common technique to increase resource utilization, reducing operational cost and energy consumption of data centers. However, the main drawback of consolidation is performance variation, due to resource contention and interferences among the VMs.

More and more applications and workflows are being deployed on the Cloud. However, scheduling of scientific applications and workflows on the Cloud is still problematic because of the task execution time variation. On consolidated servers, the task execution finish time may very unexpectedly, thus it is difficult to determine which applications are suitable to be consolidated for better performance. Recently, many works have focused on this issue [1]–[5].

These works rely on experimental results with consolidated applications, to estimate how they would react to resource contention in general. Thus, they require the running of a large number of experiments, involving scheduling various applications and workflows on VMs. However, there exists no standard framework to manage and run such large scale experiments. This work proposes a framework to easily manage and run large numbers of experiments with complex schedules and resource usage patterns on the Cloud

There are many large scale Cloud management and maintenance software stacks available for modern data centers [6]–[17]. Although they are well-equipped for performing complex maintenance, fault tolerance, and data backup services, they are not adequate for performing experiments with task scheduling and resource usages patterns of VMs for several reasons:

i) These software stacks are mainly designed for providing the Cloud services, not for performing sophisticated experiments with workloads. For example, they have special features for providing fault tolerance, VM replication, migration and high availability of VMs to a data center. The software stacks do not offer any built-in features for performing complex experiments with application scheduling patterns on the Cloud;

ii) They have many modules, and they require a lot of time and effort to master. System administrators require a lot of experience to manage these systems efficiently. On the other hand, most researchers are concerned with a quick and easy setup of experiments. It takes a lot of time to modify a large piece of software even though they do not provide friendly interfaces to conduct scientific experiments easily;

iii) Experiments with scheduling of parallel workflow on VMs often require modification the software stack of the maintenance software. Making such changes to a massive software stack with many modules is a cumbersome process. The proposed framework is designed to bypass the interaction with management software and run complex task scheduling experiments easily on the Cloud.

Recently, the understanding interactions among the VMs and improving the performance of tasks has received a significant amount of attention [1]–[5][18][19]. A simple construct of a framework, which can execute the parallel workflow on VMs residing on multiple servers can make such experimental processes much easier. Some features of the framework and contribution of this paper are briefly stated below:

i) A lightweight framework for profiling execution time variations of parallel workflow on the Cloud has been introduced. It provides a simple interface for conducting complex experiments on VMs and scheduling parallel applications across on multiple hypervisors. The primary objective is to provide an accessible platform to carry out complex experiments on the Cloud.

It can be used independent of any data center management software, thus making the general experimental process easier. There are many open source management software options. However, they have too many components and modules.

ii) They are difficult to setup for complex experiments. This framework is lightweight and easy to handle, making it easier to perform experiments with complex workload patterns.

iii) The framework allows researchers to specify an exact sequence of execution of workload pattern on VMs. A human readable *workload descriptor* file stores all the task patterns. The exact sequence of tasks that is to be executed on VMs is defined in this file. Cloud management software has many layers and hides many complexities from the users. It can be convenient for system administrators, who are only concerned with the outcome. On the other hand, during experiments measuring the impact of execution of each task may be necessary. Extensive experiments will help to understand the VM's behavior under consolidation and identify any anomaly of the schedule more quickly;

iv) Another feature of the framework is the *command descriptor* file. Parallel applications usually consist of several smaller tasks, and various command sets are required to run them. The command descriptor file contains the actual commands, and one mnemonic is issued against each set of commands. Thus, the workload descriptor file remains small and workload patterns are easy to create or modify. The command descriptor file also allows for running complex applications like web servers or database servers. The framework scans the workload file twice. During the first scan, all mnemonics are replaced, and in the second scan actual commands are executed. Thus, adding or modifying real command sets is much easier as they are stored only in one place, in the command descriptor file;

v) The framework can run experimental schedules and resource usage patterns on multiple hypervisors simultaneously. It uses the *Secure Shell* (SSH) to connect to virtualized servers, instead of the API set. The use of SSH ensures flexibility, and any hypervisors can be connected. On the other hand, using multiple API for various hypervisors is a cumbersome process. The SSH gives the ability to connect to any Cloud;

vi) The framework is implemented entirely in Java and can be run on any *operating system* (OS). It can be used as a stand-alone application or plugged-in with any other Java task scheduling program. It is lightweight, completely portable and requires no installation on the system.

To the best of our knowledge, there is no other lightweight framework written in any language, specifically to do experiments with execution time variation of parallel workflows on VMs. This framework is independent of and complementary to Cloud management software. While the management software can be used for providing Cloud services, this framework can be used to run experiments with workload patterns on the Cloud.

The effectiveness of the framework is demonstrated with a real data-intensive workflow, which processes the *Galactic Arecibo L-band Feed Array HI* (GALFA-HI) [20] survey data with the Montage toolkit [21]. The *Incremental Consolidation Benchmarking Method* (ICBM) [22] has been used to analyze the tasks of the workflow. Originally, the ICBM was introduced to analyze the execution time variations of individual tasks on VMs. In this work, it is extended to analyze the tasks of scientific workflow which has not been done previously.

The rest of the paper is organized as follows. Section II describes the problem with an example. Design goals are discussed in Section III, followed by the framework design in Section IV. Section V discusses the workflow and benchmarks used, along with experimental setup. Section VI gives the results of experiments with task execution patterns on three hypervisors. Section VII provides a brief overview and shortcomings of complementary works. A discussion about future work and conclusion are in Section VIII.

II. Problem description

The task execution time variation due to VM consolidation is one of the major problems for the Cloud. It can be even more problematic for parallel applications and scientific workflows, because of having task dependencies. Fig. 1 shows an example of workflow, which processes the *GALFA-HI survey* data [20] using the *Montage* toolkit [21]. It is a data-intensive workflow that creates a mosaic image of a part of the Milky Way galaxy from some data cubes. The data cubes are released at regular intervals, as a part of an ongoing survey. Therefore, this is a widely used workflow in the field of astronomy. It has 16 tasks (t_1 to t_16) on 8 levels (l_1 to l_8). Fig. 2 shows one possible schedule of these tasks on a set of co-located VMs.

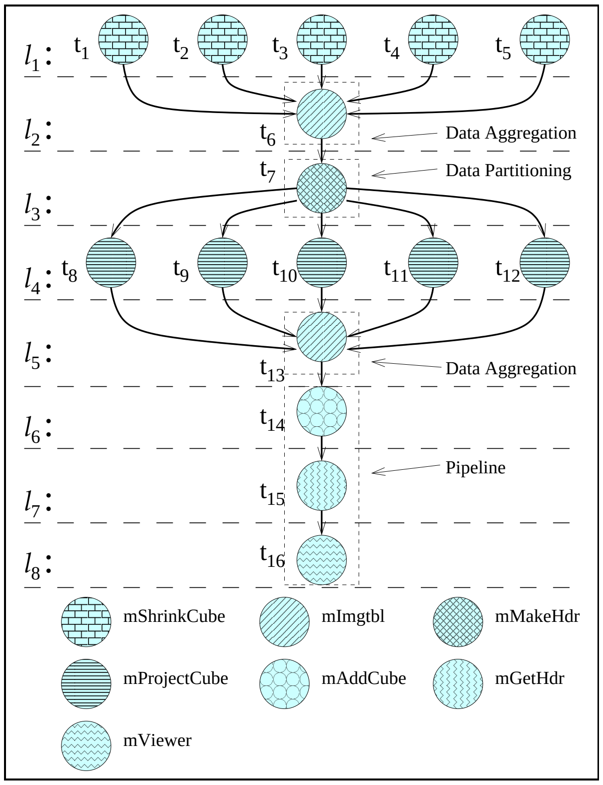


Figure 1. A workflow: GALFA-HI data processing with the Montage toolkit.

In Fig. 2, the tasks of the GALFA-HI workflow (Fig. 1) are scheduled on the VMs of a single server. Here, the server has eight simultaneously running VMs. As the tasks of the workflow have internal dependencies, they need to be scheduled hierarchically. The tasks that can be run simultaneously are grouped together in one level. The tasks of the level below are dependent on tasks of the immediate upper level.

Fig. 2 depicts that the tasks are being executed level by level on the VMs of a single server. There are VMs of three colors on the server. Light blue VMs are where the tasks of GALFA-HI are being executed. In a consolidated server, tasks from other applications are also being executed they are shown in red. Finally, white VMs represent empty VMs, where no tasks are being run at present. The tasks on additional VMs (shown in red) are responsible for resource contention and performance degradation of tasks of GALFA-HI workflow.

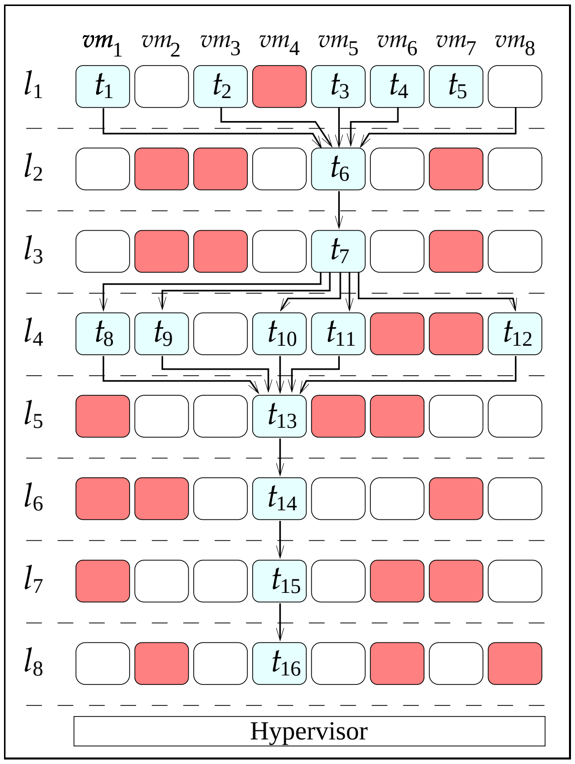

vm_1 vm_2 vm_3 vm_4 vm_5 vm_6 vm_7 vm_8
l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16}
Hypervisor

Figure 2. Scheduling GALFA-HI workflow on VMs.

In this case, performance deterioration of a task can have a cascading effect on the other tasks of the workflow, because of the task dependencies. Furthermore, the performance of tasks of the critical path would directly affect the makespan.

To efficiently schedule workflows on the Cloud it is necessary to take the execution time variations into account. Presently, there is no theoretical solution for this issue. Therefore, most recent works rely on various heuristics [1]–[5]. To design such heuristic solutions, a significant amount of experimental data may be required. This framework makes it easier to carry out large-scale experiments with VM schedules and retrieves data. The obtained data can help to design better heuristics algorithms for the system. One method to obtain such critical task execution time variation data is presented in [22], called the ICBM. This work further shows that the ICBM can be extended to scientific workflows on the Cloud.

A. *ICBM for workflow*

Originally, the ICBM was introduced to retrieve the execution time variations of VMs on consolidated servers [22]. However, the ICBM has not been used with workflows before. This work shows that the concept of ICBM can be applied to parallel workflows, too. The concept of ICBM involves increasing resource usage of a virtualized server, To systematically cause execution time variations on VMs. This means that for a parallel application the same resource usage pattern has to be applied to each task. It is described next.

Fig. 3 shows the steps of ICBM for applying a CPU-intensive resource usages pattern on the GALFA-HI workflow. Initially, only tasks of the workflow are being run on the server. It is shown on Fig. 3a, at this stage tasks from no other application are run on the server. Thus, the execution finish times of tasks of the workflow are obtained, without interferences from VMs belonging to other tasks. Afterward, the workflow is run

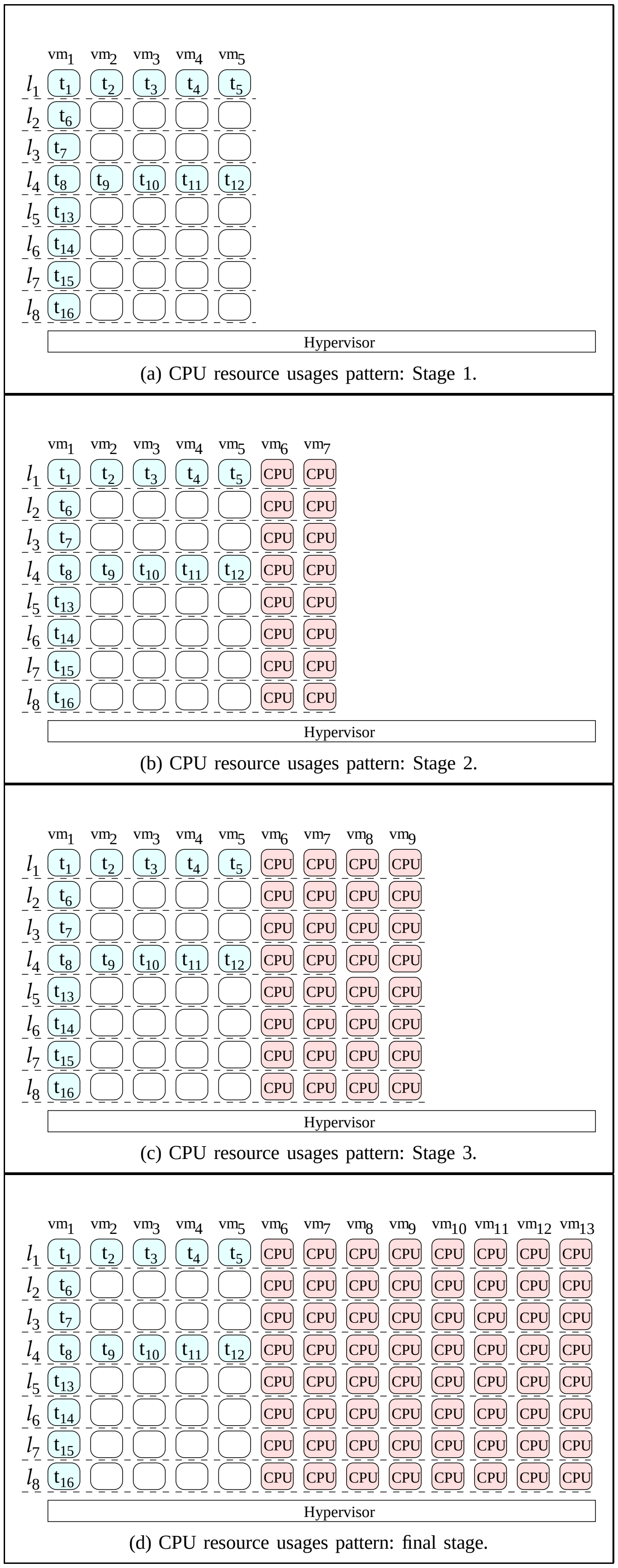
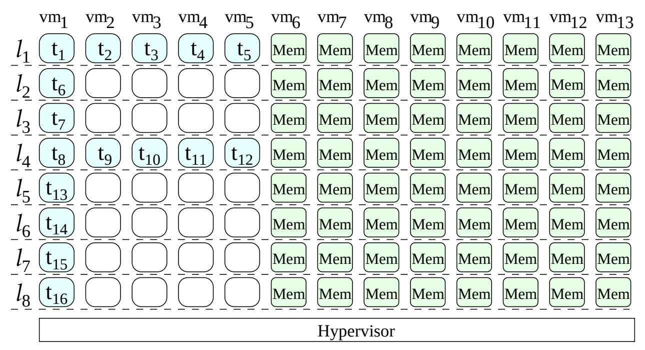

vm_1 vm_2 vm_3 vm_4 vm_5 vm_6 vm_7 vm_8 vm_9 vm_{10} vm_{11} vm_{12} vm_{13}
l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16}
CPU
Hypervisor
(a) CPU resource usages pattern: Stage 1.
(b) CPU resource usages pattern: Stage 2.
(c) CPU resource usages pattern: Stage 3.
(d) CPU resource usages pattern: final stage.

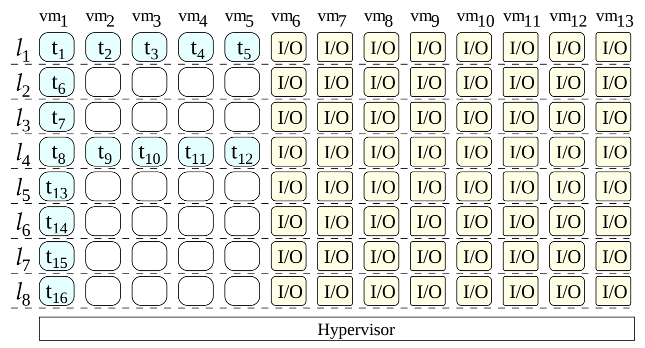
Figure 3. Applying CPU-intensive resource usages pattern on GALFA-HI workflow.

again. However, in this stage, two additional CPU-intensive tasks are executed at each level of execution. This is referred to as stage 2 and shown in Fig. 3b.

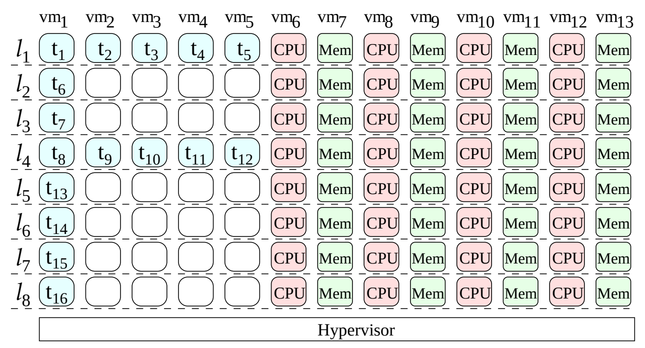
At stage 3, four additional CPU-intensive VMs are being run along with the workflow (Fig. 3c). Thus, the workflow is repeatedly run and CPU-intensive VMs are increased systematically. This process is repeated until all VMs of the server are utilized, and that is the final stage of the experiment. Fig. 3d shows the final stage for this particular server configuration. This server can accommodate a maximum of 13 VMs, and all of them have been used. Tasks of the workflow are occupying five VMs, while the remaining eight are CPU-intensive VMs.


vm1 vm2 vm3 vm4 vm5 vm6 vm7 vm8 vm9 vm10 vm11 vm12 vm13
l1 l2 l3 l4 l5 l6 l7 l8
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16
Mem
Hypervisor

(a) Mem. resource usages pattern: final stage.


vm1 vm2 vm3 vm4 vm5 vm6 vm7 vm8 vm9 vm10 vm11 vm12 vm13
l1 l2 l3 l4 l5 l6 l7 l8
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16
I/O
Hypervisor

(b) I/O resource usages pattern: final stage.


vm1 vm2 vm3 vm4 vm5 vm6 vm7 vm8 vm9 vm10 vm11 vm12 vm13
l1 l2 l3 l4 l5 l6 l7 l8
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16
CPU
Mem
Hypervisor

(c) CPU-Mem. resource usages pattern: final stage.

Figure 4. Various resource usages pattern applied on GALFA-HI workflow.

The ICBM divides experiments into stages so that the tasks of a workflow suffer the least amount of interference at stage 1 (Fig. 3a) while they face the most CPU-intensive resource usage contention at the final stage (Fig. 3d). Then, the entire procedure is repeated for another resource intensive VMs, like memory (Fig. 4a) and I/O (Fig. 4b). Afterward, the steps are repeated for combinations of resources, too. One example of combination of resources is shown in Fig. 4c, it is for CPU-Memory. Here, the process is repeated as described above. However, one CPU-intensive and one memory-intensive VM have been added at each stage, instead of two CPU-intensive ones. Other combinational resource contentions, like CPU-I/O and Memory-I/O, are created in the same process.

From the above discussion, it is clear that experimental procedures like the ICBM require handling large numbers of task schedules. Furthermore, the exact sequence of task executions on VMs and their mutual performance inferences due to consolidation, have to be known precisely. Although, many tasks and resource scheduling software exist, none of them are designed to do experiments with task execution time variations on VMs. They use high-level interfaces and hide almost all scheduling complexities from the user. That may be convenient for average Cloud users, however it is not too beneficial for researchers conducting experiments with resource contention and consolidation. The primary objective of this work is to present a low-level, lightweight framework for experimenting with complex workload patterns automatically. This framework needs to act as both a scheduler and profiler of task execution times and be able to connect to any Cloud. In this work, the design goals, implantation issues and experimental results of the framework are discussed in detail.

III. Motivation and design goals

This section discusses the primary goals and trade-offs considered while designing and implementing the framework.

Easy to perform experiments with workflow: The first priority is to provide an easy interface to perform complex experiments with the workflows on virtualized servers. There exist many complex Cloud management systems and programming paradigms. However, they are not designed for carrying out experiments with VM consolidation. The new framework should be able to perform complex experiments on the Cloud, independent of any management software. This work aims to provide an easy interface to design and carry out experiments with workflows on virtualized servers so that, the performance variation of each task can be profiled independently. The main application of the framework would be to discover the relationship among the execution time variations of consolidated VMs and resource utilization of the server.

Resource usages patterns: Experiments with consolidation are sensitive to VM placements on the server. To capture the effect of consolidation on VMs, it is necessary to create complex workload patterns and execute the tasks accordingly on VMs. Therefore, the proposed framework should provide an easy way to run the tasks according to resource usages patterns, described previously. A human readable file should contain all the workload patterns so that they are easy to create and modify. Researchers would create those files, exactly the way they want the tasks to be executed on the system. Thus, the reaction of the system to resource contentions and consolidation can be examined carefully.

Easy to check the workload patterns: Executing a task of the workflow usually requires several command sets. Managing a lot of commands in one workload pattern file is often problematic. There should be an easy way to rectify any potential error in the workload pattern. One way to achieve

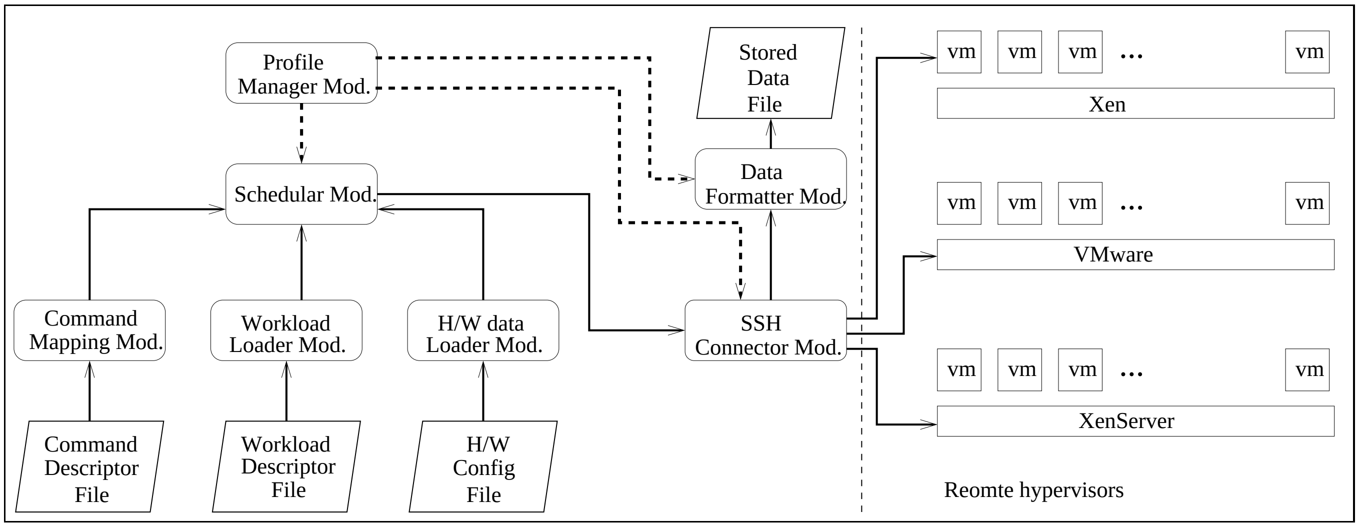


Figure 5. Modules of the framework.

this is not to inscribe full commands in the workload file, rather they are stored in a particular file, separately. Then, the workload file is created only with a short set of mnemonics During runtime, the mnemonics are mapped to actual larger command sets. The process is described in more detail in the implementation section (Section IV).

Connection to any Cloud technology: Modern data centers have a countless number of servers, and various hypervisors are deployed on them. It is necessary for the framework to be able to connect to a large number of VMs running on multiple hypervisors. Therefore, the framework needs a method with small connection overhead, and the ability to run tasks on any Cloud. The implementation section describes how this is achieved.

Easy to deploy: The framework should be easily deployable on a wide variety of systems. There are many operating systems today; therefore the framework should be as universal as possible. It should not be dependent on any Cloud management system or OS, thus, making it possible to initiate experiments from any machine, regardless of the underlying OS. Use of a common framework to perform experiments would give researchers the opportunity to share and collaborate with experimental results more widely.

In this section, motivations and design goals of the framework are described. The next section describes, how those goals are achieved during implementation.

IV. Implementation of the Framework

This section describes the implementation process of the framework to achieve the design goals of the previous section. The framework is divided into seven modules, and each module performs a particular job. All modules are shown in Fig. 5 and described below. Solid lines represent data transfer paths, while dashed lines represent command transfer paths.

The command mapping module: A workflow consists of many tasks, and each task requires a set of commands to execute properly. Inscribing all commands to a workload file is counter-productive for several reasons. It makes the workload file large, and it becomes difficult to inspect the workload patterns. Furthermore, if an error is found in one of the commands, it has to be corrected in all occurrences of the workload file. This pitfall can be avoided by storing all the actual commands in a separate *command descriptor* file.

This file stores a mnemonic against a full set of real commands, then the workload pattern files are created only with these mnemonics. During runtime, first the command mapping module loads all the actual commands to memory, then all mnemonics are replaced with their actual command sets in the workload file. This design choice makes the workload file manageable in size and easier to verify.

The workload loader module: All the experimental resource usage patterns are stored in a *workload descriptor* file, which is a human readable file containing only mnemonics. This file describes, line by line, the dependencies and exact execution sequence of the tasks. Tasks that would be running simultaneously are stored in one line while, the tasks dependent on them are written in the line below. The workload loader module scans the tasks line by line so that they can be executed on the VMs exactly in the order intended on the workload file. This makes it easier to identify how a virtualized system reacts to a particular pattern of resource usages.

The hardware configuration loader module: To execute the sequence of workload patterns correctly, some basic hardware information is required. The necessary hardware configuration of all the VMs and physical host are stored in the *hardware configuration* file. The arrangements of VMs on physical hosts along with their MAC addresses are stored in this file. The *hardware configuration loader* module fetches this data from the file, so that the framework can utilize it to connect and execute workloads on the VMs.

The scheduler module: The *scheduler module* collects information from the above three data loading modules, and allows the tasks to be executed on VMs. At first, memory mapped commands and hardware configuration file are used, to check the consistency of the workload descriptor file. In the case of any inconsistency, the process has to be terminated. After consistency checking, the scheduler issues the necessary commands to VMs through the connecting module, which is described next. It is designed as a separate module, so that it can be modified to implement any custom task scheduling algorithm for VMs if it is required.

The connecting module: Another design goal is to make the framework as universally usable as possible. The framework makes all connections through an SSH implementation in Java, called the JSch [23]. Thus, the entire framework is written in Java and can be run on any OS. It is completely portable and requires no installation. The SSH is chosen over API, to keep the framework lightweight. It allows the framework to connect to multiple hypervisors simultaneously, without having to write codes for multiple API. Furthermore, support for any new hypervisor can be easily added, without code modification.

The data formatting module: Raw data is sent back through the SSH channels; these data need to be formatted to use them with other applications. This module formats and stores the experimental results in output files. The data is analyzed later to discover the relation among resource usages patterns and task execution time variations.

The profile manager module: This is responsible for coordination among all the modules so that they can work seamlessly. The profiler is modular in design so that a module can be customized easily if required. Also, adding new modules for future functionality is much easier in this way.

The next section describes the algorithm for the framework, to demonstrate how those modules work together.

A. Algorithm for the framework

Fig. 6 shows the algorithm for the framework. First, all commands are loaded on the *COMM-LIST* from the command descriptor file. The command loader module does this, by mapping all commands to their corresponding mnemonics in memory (lines 1-2). Then, the workload loader module parse the workload descriptor file, and loads workload pattern on the *WL-LIST* (lines 3-4). The WL-LIST contains a detailed execution plan, for both the parallel application and resource contention patterns. Examples of such patterns are shown in Figs. 3 and 4. Afterward, the hardware configuration data is loaded from the file to *VM-LIST* (lines 5-6). The VM-LIST contains all the data required for connecting to VMs during experiments.

Next, a *for* loop (lines 7-21) processes the WL-LIST, line by line. Recall that the tasks that are to be run simultaneously are written in a single line. Then, an inner *for* loop (lines 8-17) removes one task at a time from the line and checks for consistency against hardware data and commands. The consistent tasks are then stored in a linked list, called the *RUN-LIST*. On the other hand, if a task is not compatible then the application exits. Once all the tasks of a line are processed, the inner *for* loop exits. Then, all the mnemonics of RUN-LIST are replaced with the actual command set, with the help of COMM-LIST (line 15). Once this is done, commands are simultaneously sent to execute all tasks of the RUN-LIST (line 16). The framework then waits for the tasks to finish, and collect the execution time data (line 17). Afterward, the same process is repeated for the next line of WL-LIST, on next iteration of the outer *for* loop. The outer *for* loop exit when all the lines of WL-LIST (entire pattern) have been processed. To experiment with another resource usage pattern, the procedure needs to be restarted from the beginning.

```
1: Load all commands and mnemonics, from the Command Descriptor file
   to $COMM-LIST$.
2: Load workloads from the Workload Descriptor file to $WL-LIST$.
3: Load the VMs configuration from file to $VM-LIST$.
4: for Each line $\mathcal{L}_i \in WL-LIST$ do
5:     for Each task, $t_j \in \mathcal{L}_i$ do
6:         Let, $comm_j \in COMM-LIST$ be the command for $t_j$.
7:         Let, $vm_j \in VM-LIST$ be the VM, where to run $t_j$ .
8:         Check the consistency of $t_j$ against $comm_j$ on $vm_j$.
9:         if $t_j$ is consistent then
10:            Put $t_j$, $comm_j$ and $vm_j$ on $RUN-LIST$.
11:        else
12:            Exit.
13:        end if
14:    end for
15:    Replace all mnemonics of $RUN-LIST$ with actual commands.
16:    Simultaneously send commands to all $vm_j$ of $RUN-LIST$.
17:    Wait for their execution to finish and collect execution time data.
18: end for
```

Figure 6. Algorithm for the framework.

V. Workloads used

Two types of workload have been used in the experiments. The first type is a data-intensive scientific workflow, which is used to observe the execution time variations of tasks under consolidation. The second type is a set of benchmarks suites, used to create resource contention patterns on servers.

A. Scientific workflow: GALFA-HI

The GALFA-HI survey continuously scans the sky for naturally occurring hydrogen atoms [20], and several data cubes have been released so far. Five of those cubes have been processed with the Montage toolkit [21], to create a mosaic image of a part of the Milky Way galaxy. The workflow is shown in Fig. 1 it has 16 tasks and eight levels. It is a data-intensive workflow, which processes about 2 GB of raw data cubes. Experiments measure the execution time variation of tasks in this workflow due to consolidation.

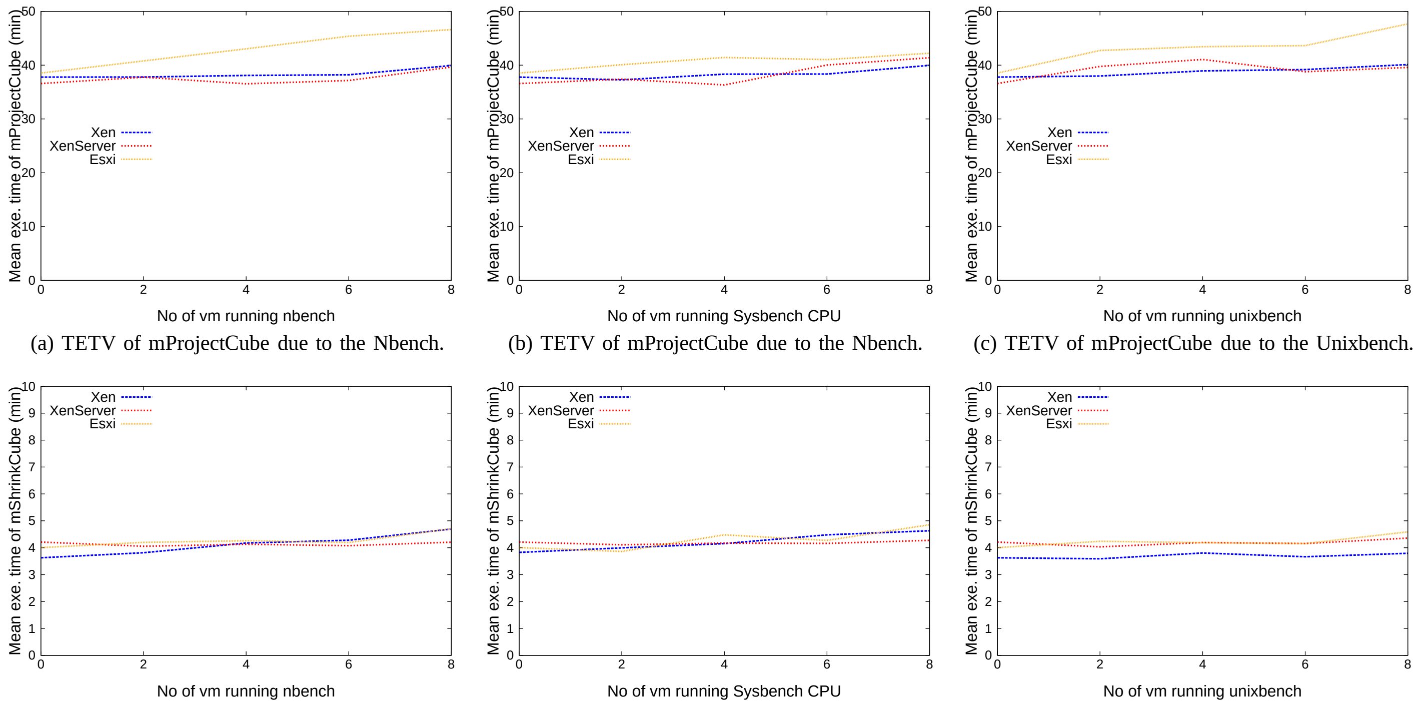
B. Set of benchmark suites

Three sets of benchmark suites have been used to create resource contention patterns on the tasks of the above workflow. They are the sets of CPU, memory and I/O-intensive benchmark suites. Each benchmark suite, in turn, consists of several similar types of tests. Due to space limitation, it is not possible to describe each benchmark suite separately. Next, each set is described in brief.

CPU-intensive benchmarks: Three CPU-intensive benchmarks have been used, they are the *Sysbench CPU* test, *Nbench* and *Unixbench*. The Sysbench CPU test has been widely used with multi-core server [24] and VM workload consolidation experiments [25]. The Nbench is a CPU-intensive benchmark suite, having ten different CPU-intensive tests [26]. The Unixbench is another CPU-intensive benchmark suite, which is used for experiments on Amazon EC2 [27].

Memory-intensive benchmarks: Three memory-intensive benchmarks have been used for creating resource contention patterns. The first is the *Cachebench*, which consists of eight different memory tests [28]. The second is the *Stream*, a syntactic benchmark program for measuring sustainable memory bandwidth [29]. The final one is the *Sysbench memory* test.

I/O-intensive benchmarks: Five I/O-intensive tests have been used to create resource contention patterns. The *Filebench* is an important I/O benchmark suite [30], which can be configured to perform various I/O-intensive tests. Five of them are used, they are the *file-server*, *web-server*, *web-proxy*, *video-server* and *online transaction processing* (OLTP) test.



(a) TETV of mProjectCube due to the Nbench. (b) TETV of mProjectCube due to the Nbench. (c) TETV of mProjectCube due to the Unixbench.

(d) TETV of mShrinkCube due to the Unixbench. (e) TETV of mShrinkCube due to the Sysbench CPU. (f) TETV of mShrinkCube due to the Sysbench CPU.

Figure 7. Task execution time variation (TETV) of the mProjectCube and mShrinkCube functions due to the CPU-intensive workload patterns on VMs.

C. *Experimental setup*

Three Dell XPS-8500 servers of identical hardware configuration had been set up for the experiments. Each server has one Intel i7-3770 processor and 32 GB memory. The i7-3770 has four cores and eight hardware threads, each is clocked at 3.4 GHz. Three different hypervisors are installed on three servers; they are, *VMware ESXi* 5.5, *Citrix XenServer* 6.5 and *Xen* 4.6 on *Centos* 7.

Each hypervisor has 14 VMs of identical configuration. Each VM has one processor, 2 GB of Ram and 50 GB virtual disk. During experiments, the framework connects to all 42 (14×3) VMs on three hypervisors and execute workload patterns simultaneously. The framework itself runs on a remote Dell OptiPlex 9010 machine and connects to hypervisors through the LAN. The results of experiments are given next.

VI. Results

Recall that the GALFA-HI workflow (Fig. 1) has 16 tasks, comprised of seven functions. Average execution times of those seven functions without interferences are shown in Table I. In this case, the tasks are scheduled exactly like that of Fig. 3a. Due to space constraints, it is not possible to discuss execution time variations of all seven functions. Results are shown for only two functions, the *mProjectCube* and *mShrinkCube*. The rest of the functions also show variations similar that of these functions. The results are grouped according to the resources loads for convenience of discussion, for all three hypervisors.

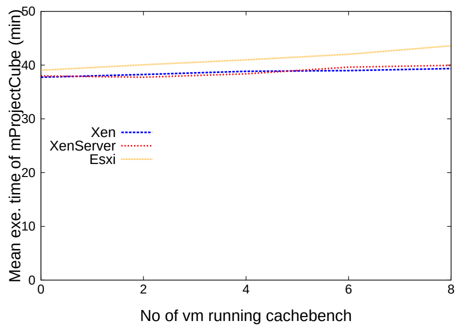
Variations due to CPU-intensive workload: The graphs in Fig. 7 show execution time variations of both the mProjectCube and mShrinkCube functions for CPU-intensive workloads, on three hypervisors. In each graph, the Y-axis represents the execution time variation. The X-axis represents

TABLE I. MEAN EXECUTION TIMES OF TASKS OF GALFA-HI WORKFLOW ON VMS WITHOUT INTERFERENCES (AS SHOWN IN FIG. 3a).

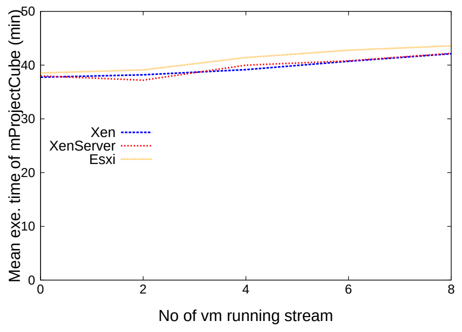
Level	Task	Time (m)
1	mShrinkCube	3.878
2 & 5	mImgtbl	0.02
3	mMakeHdr	0.02
4	mProjectCube	39.774
6	mAddCube	12.32
7	mGetHdr	0.02
8	mViewer	0.04

how many CPU-intensive VMs were running on the server, besides the workflow. The first point of the X-axis is zero, meaning no other VMs were running when the execution time of the function was measured. This execution schedule is shown in Fig. 3a. The next point on X-axis is 2; here two additional CPU-intensive VMs were running at every step of the workflow execution (schedule shown in Fig. 3b). In this way, the workflow is repeatedly executed with increasing number of CPU-intensive VMs. The final point is 8, indicating eight additional CPU-intensive VMs were used, at each step of workflow execution as shown in Fig. 3d.

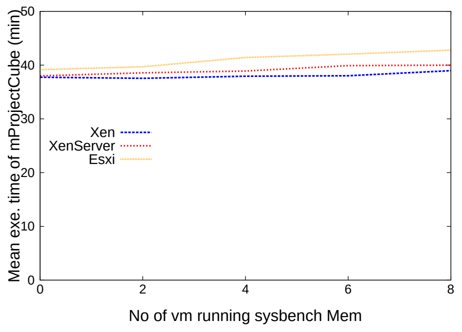
In Fig. 7, from left to right on the X-axis the interference from the number of CPU-intensive VMs increases. The leftmost point is the execution time of a task without any interference from other VMs. The rightmost point is the execution time of the same task with maximum interference. Fig. 7 shows that both the mProjectCube and mShrinkCube tasks show relatively less execution time variation because of CPU-intensive VMs. It applies to all three hypervisors. On ESXi hypervisor, the execution time of mProjectCube function goes from 38.52 minute (the leftmost point on the graph) to 48.13 minute (rightmost point) due to the addition of 8 VMs, each running a Unixbench benchmark suite (Fig. 7c).



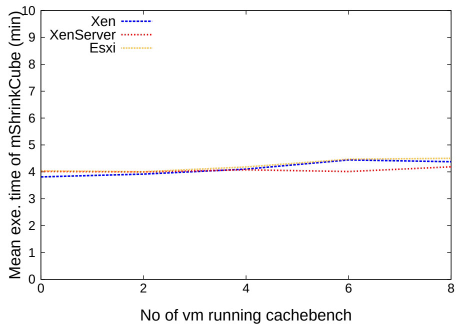
(a) TETV of mProjectCube due to the Cachebench.



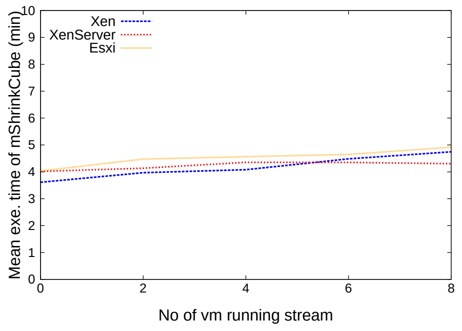
(b) TETV of mProjectCube due to the Cachebench.



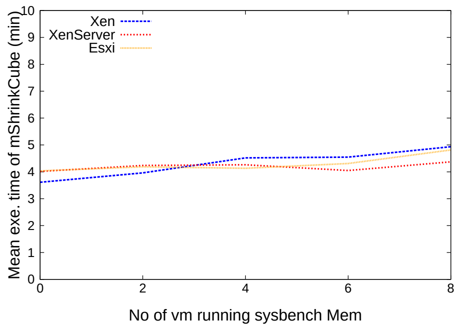
(c) TETV of mProjectCube due to the Sysbench Mem.



(d) TETV of mShrinkCube due to the Sysbench Mem.



(e) TETV of mShrinkCube due to the Stream.



(f) TETV of mShrinkCube due to the Stream.

Figure 8. Task execution time variation (TETV) of the mProjectCube and mShrinkCube functions due to the Memory-intensive workload patterns on VMs.

Therefore, consolidation with eight additional CPU-intensive VMs (in this case the Unixbench) causes 24.94% increase in execution time of the mProjectCube function. It is the highest among three hypervisors. For other hypervisors, the effect of CPU-intensive VMs is minimal. For XenServer, the maximum execution time variation among the tasks is suffered by the mProjectCube function again. It is 13.49% and caused when consolidated with eight VMs running Sysbench CPU tests (Fig. 7b). For Xen, the mProjectCube function also shows the maximum variation among the tasks; it is 6.15%. In this case, eight VMs with Unixbench were consolidated with the function (Fig. 7c).

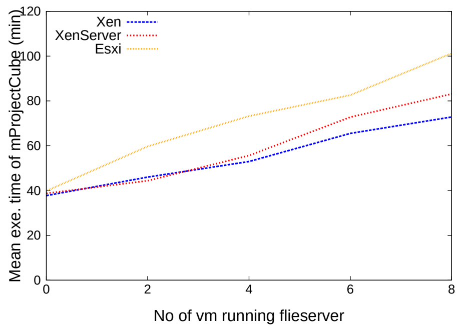
Variations due to memory-intensive workload: Fig. 8 shows the execution time variations of two previous functions, due to the memory-intensive workload on VMs. For all three hypervisors, the maximum execution time variations are shown by the mProjectCube function. In all three cases, it is consolidated with VMs running the Stream benchmark (Fig. 8b). The execution time increase of ESXi, XenServer, and Xen hypervisors are 24.24%, 11.02%, and 11.56%, respectively.

Variations due to I/O-intensive workload: During VM consolidation experiments, the I/O-intensive tasks tend to show a greater degree of resource contention. That is why more I/O-intensive benchmarks have been used in the experiments, compared to other types. Fig. 9 shows the execution time variations of the mProjectCube and mShrinkCube functions, due to consolidation with five different I/O-intensive benchmarks.

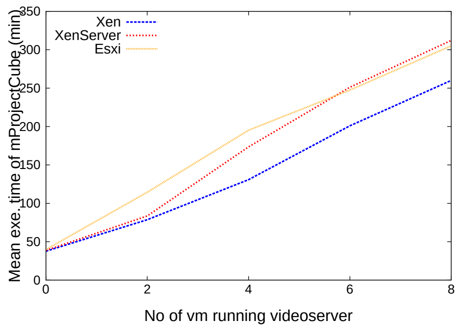
The VMs with video servers cause huge execution time variation for both functions, on all three hypervisors (Fig. 9b). Consolidation with eight VMs with video servers, increases the execution times of mProjectCube function for ESXi, XenServer, and Xen by 683.30%, 705.83%, and 588.96%, respectively. The video servers also have similar effects on the mShrinkCube function, on all three hypervisors (Fig. 9e). The execution time increase of the mShrinkCube function for ESXi, XenServer, and Xen are 901.92%, 774.10%, and 595.34%, respectively. For other I/O-intensive benchmarks, similar results can be obtained, too. For example, Fig. 9a shows the execution time variation of the mProjectCube function due to file-servers on all three hypervisors. Here, execution time increases for ESXi, XenServer, and Xen are 154.39%, 114.78%, and 92.95%, respectively. The file-servers similarly cause execution time variation for the mShrinkCube function, too. Execution time increases for ESXi, XenServer, and Xen are 411.13%, 347.96%, and 343.15%, respectively.

From the presented execution time variation data, it is clear that combination of benchmarks can be used to create resource contention patterns for tasks on VMs. The significance of the above findings is discussed next.

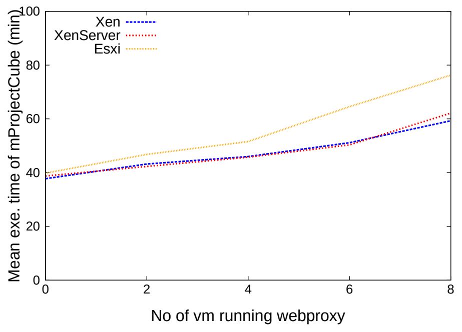
Discussion: The experimental results show that resources like CPU, memory, and I/O, all have dissimilar effects on the task execution time. It is observed for all three hypervisors. From the results, it is clear that execution time variation directly depends on the cumulative resource requirement of the VM of a server. It has been shown previously that, by profiling the execution times of co-located VMs, it is possible to predict the task execution time variations [22]. The resource requirement of the VMs, play a huge part on execution time variations. For example, both the mProjectCube and mShrinkCube functions are I/O-intensive tasks, and they have the maximum variation for I/O-intensive benchmarks. The objective of experiments is to show that the proposed framework can profile the tasks of a scientific workflow for any workload and hypervisor. Thus, it can help to design and carry out experiments, with VM placement and consolidation for scientific workflows.



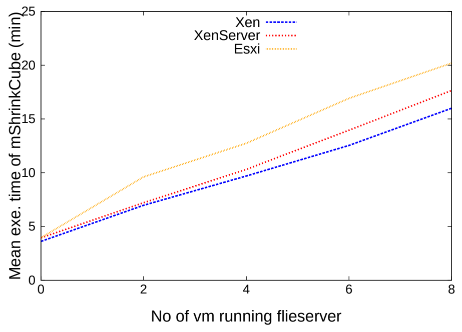
(a) TETV of mProjectCube due to File server.



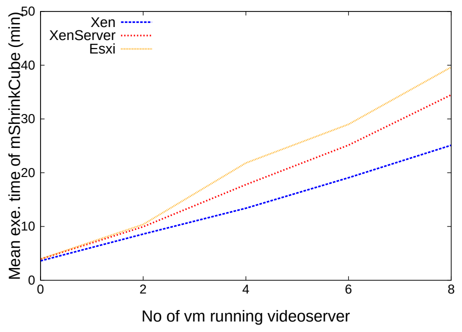
(b) TETV of mProjectCube due to Video server.



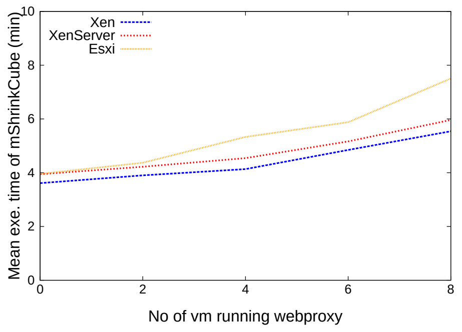
(c) TETV of mProjectCube due to Webproxy server.



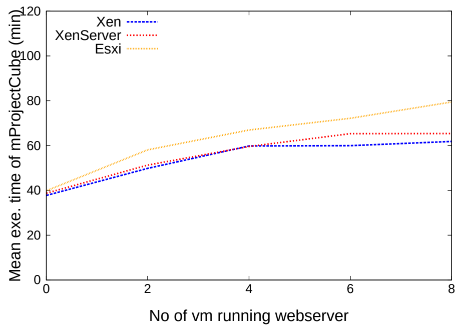
(d) TETV of mShrinkCube due to File server.



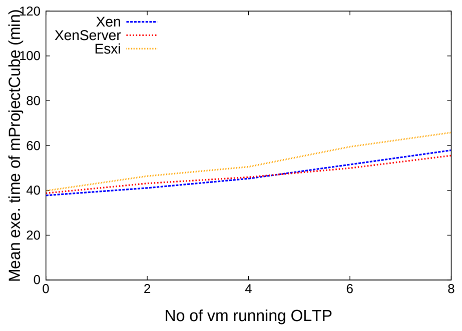
(e) TETV of mShrinkCube due to Video server.



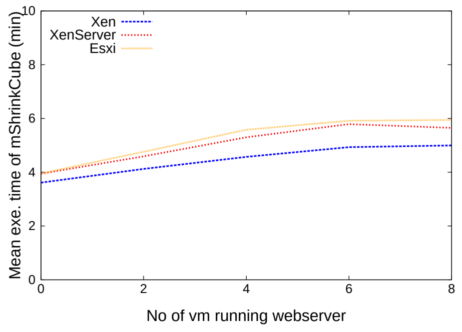
(f) TETV of mShrinkCube due to Webproxy server.



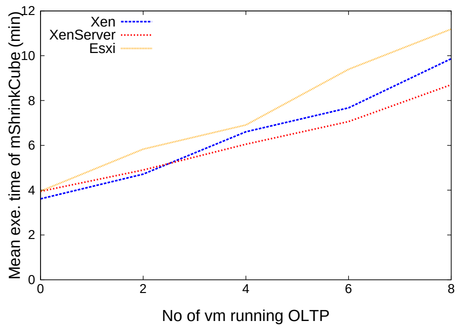
(g) TETV of mProjectCube due to Web server.



(h) TETV of mProjectCube due to OLTP.



(i) TETV of mShrinkCube due to Web server.



(j) TETV of mShrinkCube due to OLTP.

Figure 9. Task execution time variation (TETV) of the mProjectCube and mShrinkCube functions due to the I/O-intensive workload patterns on VMs.

VII. RELATED WORK

Related works can be divided into two broad categories. The first category of works deals with application performance efficiency on the Cloud and VM consolidation [1]–[5]. However, the works do not provide any general framework to do experiments with tasks of parallel applications. In contrast, this work provides a simple and effective framework that can be used for such purposes on the Cloud.

The second category of works are the Cloud management, maintenance and scheduling software [6]–[17]. They can provide many high-level functionalities for the Cloud, like running selected jobs periodically. Many complex operations can be performed with a few commands. However, they hide a lot of operational complexity from the users, and do not allow low-level control over the task execution process. On the other hand, this framework offers an easy interface for executing tasks according to the requirement of the experiment.

Although the works outlined above provide some high-level support for running tasks on the Cloud, none of them combines all the low-level functionality to carry out experiments with VM consolidation. To the best knowledge of the authors, no other previous work has proposed any such framework to perform experiments with workloads on the Cloud.

VIII. Future work and Conclusion

There are a lot of issues related to the Cloud that depend on consolidation, like application performance, energy efficiency, and resource utilization. There are no theoretical solutions available for these problems. Further experiments are required to obtain practical solutions. In future, the framework would be used to setup larger scale of experiments with various scientific workflows and diverse sets of resource usage patterns.

This work presents the design and implementation of a framework for performing experiments with execution time variation of scientific workflows on the Cloud. Profiling of task execution time is required for better understanding of VM consolidation. The framework can apply any resource usage patterns to the tasks of a workflow. It does not compile the input files, rather it behaves like an interpreter. There is no well-accepted theocratical model for task execution variation due to consolidation. Therefore such a framework would help to set up large-scale experiments for achieving a practical solution.

To show the capability of the framework to perform experiments a real life data-intensive workflow and three hypervisors have been used. Resource contention patterns for VMs have been created by combining various types of benchmarks. The framework is lightweight and implemented in Java. It can be run on any OS and can connect to any hypervisor or the Cloud. An extensive set of experiments has been done on three well-known hypervisors, and results are successfully retried, demonstrating that the framework is capable of executing any workflow schedule and resource usage pattern on multiple hypervisors. This framework can be a powerful tool for experimenting with VM consolidation and task execution time variation of workflows.

References

[1] T. Zhu, D. S. Berger, and M. Harchol-Balter, “SNC-Meister: Admitting More Tenants with Tail Latency SLOs,” in *SoCC ’16*, (New York, NY, USA), pp. 374–387, ACM, 2016.

[2] R. Taft, W. Lang, J. Duggan, A. J. Elmore, M. Stonebraker, and D. DeWitt, “STeP: Scalable Tenant Placement for Managing Database-as-a-Service Deployments,” in *SoCC ’16*, (New York, NY, USA), pp. 388–400, ACM, 2016.

[3] R. R. Sambasivan, I. Shafer, J. Mace, B. H. Sigelman, R. Fonseca, and G. R. Ganger, “Principled Workflow-centric Tracing of Distributed Systems,” in *SoCC ’16*, (New York, NY, USA), pp. 401–414, ACM, 2016.

[4] K. Rajan, D. Kakadia, C. Curino, and S. Krishnan, “PerfOrator: Eloquent Performance Models for Resource Optimization,” in *SoCC ’16*, (New York, NY, USA), pp. 415–427, ACM, 2016.

[5] C.-C. Hung, L. Golubchik, and M. Yu, “Scheduling Jobs Across Geo-distributed Datacenters,” in *SoCC ’15*, (New York, NY, USA), pp. 111–124, ACM, 2015.

[6] F. Guthrie, S. Lowe, and K. Coleman, *VMware vSphere Design*. Alameda, CA, USA: SYBEX Inc., 2nd ed., 2013.

[7] M. Liebowitz, C. Kusek, and R. Spies, *VMware vSphere Performance: Designing CPU, Memory, Storage, and Networking for Performance-Intensive Workloads*. Alameda, CA, USA: SYBEX Inc., 1st ed., 2014.

[8] D. E. Williams, *Virtualization with Xen(Tm): Including XenEnterprise, XenServer, and XenExpress: Including XenEnterprise, XenServer, and XenExpress*. Syngress Publishing, 2007.

[9] G. Ahmed, *Implementing Citrix XenServer Quickstarter*. Packt Publishing, 2013.

[10] N. Sabharwal and R. Shankar, *Apache CloudStack cloud computing: leverage the power of CloudStack and learn to extend the CloudStack environment*. Community experience distilled, Birmingham: Packt Publ., 2013.

[11] K. Jackson, *OpenStack Cloud Computing Cookbook*. Packt Publishing, 2012.

[12] A. Paradowski, L. Liu, and B. Yuan, “Benchmarking the Performance of OpenStack and CloudStack,” in *ISORC ’14*, (Washington, DC, USA), pp. 405–412, IEEE CS, 2014.

[13] S. A. Baset, “Open Source Cloud Technologies,” in *SoCC ’12*, (New York, NY, USA), pp. 28:1–28:2, ACM, 2012.

[14] P. Sempolinski and D. Thain, “A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus,” in *CLOUDCOM ’10*, (Washington, DC, USA), pp. 417–426, IEEE Computer Society, 2010.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus Open-Source Cloud-Computing System,” in *CCGRID ’09*, (Washington, DC, USA), pp. 124–131, IEEE Computer Society, 2009.

[16] S. Pousty and K. Miller, *Getting Started with OpenShift*. O’Reilly Media, Inc., 1st ed., 2014.

[17] D. Bernstein, “Cloud Foundry Aims to Become the OpenStack of PaaS,” *IEEE Cloud Computing*, vol. 1, no. 2, pp. 57–60, 2014.

[18] M. Silva, M. R. Hines, D. Gallo, Q. Liu, K. D. Ryu, and D. d. Silva, “CloudBench: Experiment Automation for Cloud Environments,” in *IC2E ’13*, (Washington, DC, USA), pp. 302–311, IEEE CS, 2013.

[19] C. Delimitrou, D. Sanchez, and C. Kozyrakis, “Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters,” in *SoCC ’15*, (New York, NY, USA), pp. 97–110, ACM, 2015.

[20] J. E. G. Peek, C. Heiles, K. A. Douglas, M.-Y. Lee, J. Grcevich, S. Stanimirovi, M. E. Putman, E. J. Korpela, S. J. Gibson, A. Begum, D. Saul, T. Robishaw, and M. Kro, “The GALFA-HI Survey: Data Release 1,” *The Astrophysical J. Supplement Series*, vol. 194, no. 2, p. 20, 2011.

[21] G. B. Berriman, J. Good, B. Rusholme, and T. Robitaille, “The Next Generation of the Montage Image Mopsaic Engine,” in *American Astronomical Society Meeting Abstracts*, vol. 227 of *American Astronomical Society Meeting Abstracts*, p. 348.13, Jan. 2016.

[22] M. Ahmed and A. Y. Zomaya, “Profiling and Predicting Task Execution Time Variation of Consolidated Virtual Machines,” in *CLOUD COMPUTING ’16*, pp. 103–112, IARIA, 2016.

[23] JCraft, Inc., “JSch - Java Secure Channel.” URL: `http://www.jcraft.com/jsch/`. Retrieved: May, 2016.

[24] H. Park, S. Baek, J. Choi, D. Lee, and S. H. Noh, “Regularities Considered Harmful: Forcing Randomness to Memory Accesses to Reduce Row Buffer Conflicts for Multi-core, Multi-bank Systems,” *SIGPLAN Not.*, vol. 48, pp. 181–192, Mar. 2013.

[25] J. Ouyang, J. R. Lange, and H. Zheng, “Shoot4U: Using VMM Assists to Optimize TLB Operations on Preempted vCPUs,” in *VEE ’16*, (New York, NY, USA), pp. 17–23, ACM, 2016.

[26] C. C. Eglantine, *NBench*. TypPRESS, 2012. ISBN: 9786136257211.

[27] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, “Exploiting Hardware Heterogeneity Within the Same Instance Type of Amazon EC2,” in *HotCloud ’12*, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2012.

[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*, pp. 19–25, Dec. 1995.

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