DynamicCloudSim: Simulating Heterogeneity in Computational Clouds

Marc Bux[*]
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
buxmarcn@informatik.hu-berlin.de

Ulf Leser
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
leser@informatik.hu-berlin.de

ABSTRACT
Simulation has become a commonly employed first step in evaluating novel approaches towards resource allocation and task scheduling on distributed architectures. However, existing simulators fall short in their modeling of the instability common to shared computational infrastructure, such as public clouds. In this work, we present DynamicCloudSim which extends the popular simulation toolkit CloudSim with several factors of instability, including inhomogeneity and dynamic changes of performance at runtime as well as failures during task execution. As a use case and validation of the introduced functionality, we simulate the impact of instability on scientific workflow scheduling by assessing and comparing the performance of four schedulers in the course of several experiments. Results indicate that our model seems to adequately capture the most important aspects of cloud performance instability, though a validation on real hardware is still pending. The source code of DynamicCloudSim and the examined schedulers is available at https://code.google.com/p/dynamiccloudsim/.

Keywords
Cloud Computing, Simulation, Instability, Heterogeneity, Scientific Workflows, Scheduling

1. INTRODUCTION
Over the last decade, cloud computing emerged as a form of distributed computing, in which computational resources can be provisioned on-demand over the Internet [1]. In the Infrastructure-as-a-service (IaaS) model of cloud computing, computational resources of any scale can be rented in the form of virtual machines (VMs) from commercial cloud providers like Amazon or Microsoft [2]. The convenience of its pay-as-you-go model along with the aggressive promotion by its providers has led to an exponential growth in the usage of cloud computing over the last years (see Figure 1).

Tailoring highly scalable applications to make efficient use of cloud resources requires developers to be aware of both the performance of rented cloud infrastructure and the requirements of the to-be-deployed application. These characteristics are hard to quantify and vary depending on the application and cloud provider. Since benchmarking a given application on cloud infrastructure of large scale repeatedly under various experimental conditions is both tedious and expensive, simulation constitutes a convenient and affordable way of evaluation prior to implementation and execution on real hardware [3–5].

Unfortunately, available cloud simulation toolkits like CloudSim [6] do not adequately capture inhomogeneity and dynamic performance changes inherent to non-uniform and shared infrastructures like computational clouds. The effect of these factors of uncertainty and instability is not negligible and has been repeatedly observed to strongly influence the runtime of a given application on commercial clouds such as Amazon's Elastic Compute Cloud (EC2) (e.g., [7–13]).

In this work, we present DynamicCloudSim, an extension to CloudSim which provides an array of capabilities to model the instability inherent to computational clouds and similar distributed infrastructures. We showcase the applicability of DynamicCloudSim in a series of experiments involving the scheduling of a computationally intensive scientific workflow which has been repeatedly used for evaluation purposes. We believe scientific workflow scheduling to be a suitable use case for demonstrating the capabilities of a cloud simulation framework for two reasons: (1) Scheduling computationally intensive scientific workflows on distributed and potentially shared architectures presents many opportunities for optimizing robustness to instability [14]; (2) Simulation has been repeatedly made use of for evaluating scientific workflow schedulers (e.g., [15, 16]).

Scientific workflows are directed, acyclic graphs (DAGs), in which nodes correspond to data processing tasks and edges constitute data dependencies between these tasks. They have recently gained attention as a flexible programming paradigm for modeling, representing, and executing complex computations and analysis pipelines in many different areas of scientific research [17]. A considerable number of scientific workflow management systems (SWfMS) has been developed, including Taverna [18], Kepler [19], and Pegasus [20]. A comprehensive survey on SWfMS has been presented by Yu and Buyya [21]. Many of these systems provide the functionality to design workflows in a drag-and-drop user interface, share workflows with other users in public repositories (e.g., myExperiment [22]), or execute workflows on appropriate computational architectures.

[*]To whom correspondence should be addressed

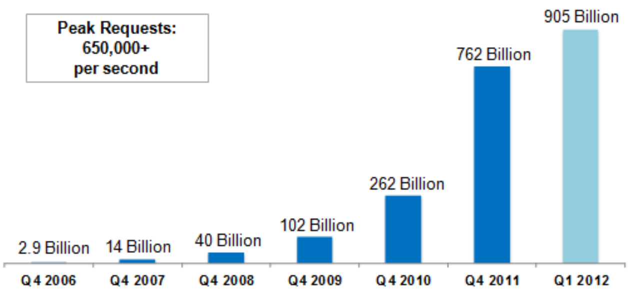

Figure 1: Total number of objects stored in the Amazon Simple Storage Service (Amazon S3) since its introduction in 2006. The exponential growth of the largest cloud provider's data storage solution mirrors the trend of compute clouds increasing in size and popularity. Image has been published on the Amazon Web Services Blog in April 2012[1].

A variety of sophisticated algorithms for scheduling scientific workflows on shared infrastructures such as computational clouds have been developed (e.g., [23–25]). As an application example for DynamicCloudSim, we compare the performance of several established scientific workflow schedulers at different levels of instability. Since some of the investigated schedulers have been developed to handle heterogeneity, dynamic performance changes, and failure, we expect our experiments to replicate the advertised strengths of the different workflow schedulers. Results from an extensive number of simulation runs confirm these expectations, underlining the importance of elaborate scheduling mechanisms when executing workflows on shared computational infrastructure.

The remainder of this document is structured in the following way: The CloudSim framework is described in Section 2, whereas the features it has been extended with are described in Section 3. The setup of the scientific workflow scheduling experiments showcasing the capabilities of DynamicCloudSim is outlined in Section 4. The results of these experiments are presented and discussed in Section 5. Related Work is summarized in Section 6. Finally, a conclusion and an outlook to future work is given in Section 7.

2. CLOUDSIM

CloudSim is an extension of the GridSim [26] framework for simulation of resource provisioning and scheduling algorithms on cloud computing infrastructure developed by Calheiros et al. [6] at the University of Melbourne's CLOUDS Laboratory. It provides capabilities to perform simulations of assigning and executing a given workload on a cloud computing infrastructure under different experimental conditions. CloudSim for instance has been used to (1) measure the effects of a power-aware VM provisioning and migration algorithm on datacenter operating costs for real-time cloud applications [3], (2) evaluate a cost-minimizing algorithm of VM allocation for cloud service providers, which takes into account a fluctuating user base and heterogeneity of cloud VMs [4], (3) develop and showcase a scheduling mechanism for assigning tasks of different categories – yet without data dependencies – to the available VMs [5].

[1]http://tinyurl.com/6uh8n24

CloudSim operates event-based, i.e., all components of the simulation maintain a message queue and generate messages, which they pass along to other entities. A CloudSim simulation can instantiate several *datacenters*, each of which is comprised of *storage* servers and physical *host* machines, which in turn host multiple *VMs* executing several *tasks* (named *cloudlets* in CloudSim). For a detailed overview, refer to Figure 2. A datacenter is characterized by its policy of assigning requested VMs to host machines (with the default strategy being to always choose the host with the least cores in use). Each datacenter can be configured to charge different costs for storage, VM usage, and data transfer.

The computational requirements and capabilities of hosts, VMs, and tasks are captured in four performance measures: MIPS (million instructions per second per core), bandwidth, memory, and local file storage. Furthermore, each host has its own policy which defines how its computational resources are to be distributed among allocated VMs, i.e., whether VMs operate on shared or distinctly separated resources and whether over-subscription of resources is allowed. Similarly, each VM comes with a scheduling policy specifying how its resources are to be distributed between tasks. On top of this architecture, an application-specific datacenter broker supervises the simulation, requesting the (de-)allocation of VMs from the datacenter and assigning tasks to VMs.

One of the key aspects of CloudSim is that it is easily extensible. Several extensions have been presented, including (1) NetworkCloudSim [27], which introduces sophisticated network modeling and inter-task communication, (2) EMUSIM [28], which uses emulation to determine the performance requirements and runtime characteristics of an application and feeds this information to CloudSim for more accurate simulation, or (3) CloudMIG [29], which facilitates the migration of software systems to the cloud by contrasting different cloud deployment options based on the simulation of a code model in CloudSim.

3. DYNAMICCLOUDSIM

CloudSim assumes provisioned virtual machines to be predictable and stable in their performance: Hosts and VMs are configured with a fixed amount of MIPS and bandwidth and VMs are assigned to the host with the most available MIPS. On actual cloud infrastructure like Amazon EC2, these assumptions do not hold. While most IaaS cloud vendors guarantee a certain processor clock speed, memory capacity, and local storage for each provisioned VM, the actual performance of a given VM is subject to the underlying physical hardware as well as the usage of shared resources by other VMs assigned to the same host machine. In this section, we outline the extensions we have made to the CloudSim core framework as well as the rationale behind them.

3.1 File I/O

In CloudSim, the amount of time required to execute a given task on a VM depends solely on the task's length (in MI) and the VM's processing power (in MIPS). Additionally, the external bandwidth (in KB/s) of VMs and their host machines can be specified, but neither have an impact on the runtime of a task. Moreover, many data-intensive tasks are neither computational- nor communication-intensive, but primarily I/O-bound. Especially in database applications, a substantial amount of tasks involves reading or writing large amounts of data to local or network storage [7].

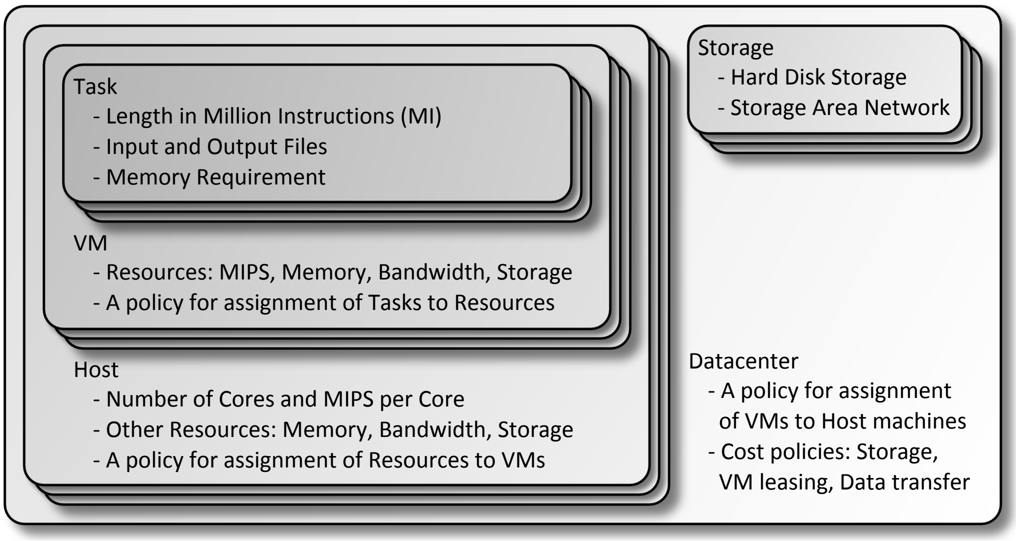
Task
- Length in Million Instructions (MI)
- Input and Output Files
- Memory Requirement
VM
- Resources: MIPS, Memory, Bandwidth, Storage
- A policy for assignment of Tasks to Resources
Host
- Number of Cores and MIPS per Core
- Other Resources: Memory, Bandwidth, Storage
- A policy for assignment of Resources to VMs
Storage
- Hard Disk Storage
- Storage Area Network
Datacenter
- A policy for assignment of VMs to Host machines
- Cost policies: Storage, VM leasing, Data transfer

Figure 2: The architecture of the CloudSim framework.

DynamicCloudSim introduces external bandwidth as a requirement of tasks and file I/O as an additional performance characteristic of tasks, VMs and hosts. It takes into account all performance requirements of a task when determining how long it takes to execute the task on a given VM. Hence, DynamicCloudSim allows the simulation of executing different kinds of tasks (CPU-, I/O-, bandwidth-bound) on VMs with different performance characteristics.

3.2 The Need for Introducing Instability

In a performance analysis spanning multiple Amazon EC2 datacenters, Dejun et al. [7] observed occasional severe performance drops in virtual machines, which would cause the response time of running tasks to greatly increase. Furthermore, they reported the response time of CPU- and I/O-intensive application to vary by a factor of up to four on VMs of equal configuration. Notably, they detected no significant correlation between CPU and I/O performance of VMs. Zaharia et al. [8] found the I/O throughput of "small"-sized VM instances in EC2 to vary between roughly 25 and 60 MB per second, depending on the amount of co-located VMs running I/O-heavy tasks.

In a similar evaluation of Amazon EC2, Jackson et al. [9] detected different physical CPUs underlying similar VMs: Intel Xeon E5430 2.66 GHz, AMD Opteron 270 2 GHz, and AMD Opteron 2218 HE 2.6 GHz. They also observed network bandwidth and latency to depend on the physical hardware of the provisioned VMs. When executing a communication-intensive task on 50 VMs, the overall communication time varied between three and five hours over seven runs, depending on the network architecture underlying the provisioned VMs. In the course of their experiments, they also had to restart about one out of ten runs due to the occurrence of failures. Similar observations have been made in other studies on the performance of cloud infrastructure [10, 11].

Another comprehensive analysis of the performance variability in Amazon EC2 was conducted by Schad et al. [12]. Once per hour over a time period of two months they measured the CPU, I/O, and network performance of newly provisioned VMs in Amazon EC2 using established microbenchmarks. Performance was found to vary considerably and generally fall into two bands, depending on whether the VM would run on Intel Xeon or AMD Opteron infrastructure. The variance in performance of individual VMs was also shown to strongly influence the runtime of a real-world MapReduce application on a virtual cluster consisting of 50 EC2 VMs. A further interesting observation of this study was that the performance of a VM depends on the hour of the day and day of the week. Iosup et al. [13] made similar observations when analyzing more than 250,000 real-world performance traces of commercial clouds.

Evidently, the performance of computational cloud infrastructure is subject to different factors of instability:

1. Heterogeneous physical hardware underlying the provisioned VMs (*Het*)
2. Dynamic changes of performance at runtime (*DCR*)
3. Straggler VMs and failed task executions (*SaF*)

In the remainder of this section, we describe in detail how DynamicCloudSim attempts to capture these factors of instability.

3.3 Heterogeneity

Similar to Amazon EC2, the provisioning of resources to virtual machines in DynamicCloudSim is based on compute units instead of fixed performance measures. Different host machines provide a different amount of computing power per provisioned compute unit, effectuating in heterogeneity (*Het*) among VM performance. Furthermore, in contrast to CloudSim, DynamicCloudSim does not assign new VMs to the host with the most available resources, but to a random machine within the datacenter. Hence, VMs of equal configuration are likely to be assigned to different types of physical machines providing varying amounts of computational resources. Similar to a commercial cloud like Amazon EC2, the user is oblivious to the hardware underlying the provisioned VMs and has no control over the VM allocation

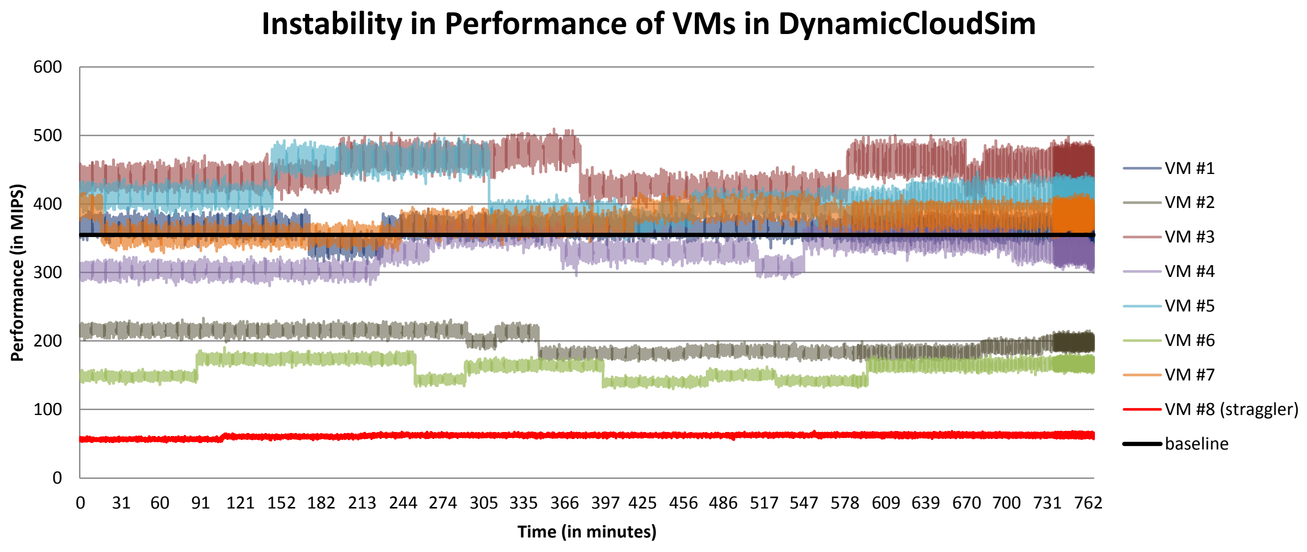

Figure 3: CPU performance of a configuration of eight VMs (including one straggler) running for 12 hours with default parameters in DynamicCloudSim. The black line represents how the VMs' performance would have looked like in basic CloudSim.

policy. In Section 4, we describe how we set up a simulation environment resembling the inhomogeneous hardware configuration of Amazon EC2

In addition to the heterogeneity achieved by assigning VMs to different types of hosts, DynamicCloudSim also provides the functionality to randomize the individual performance of a VM. If this feature is enabled, the performance characteristics (CPU, I/O, and bandwidth) of a newly allocated VM are sampled from a normal distribution instead of using the default values defined by the VM's host machine. The mean of this normal distribution is set to the host's default value of the performance characteristic and the relative standard deviation (RSD) can be defined by the user, depending on the desired level of heterogeneity. Schad et al. [12] found several of the performance measurements of VMs in Amazon EC2 – particularly random disk I/O and network bandwidth – to be normally distributed. Hence, while we plan to introduce sampling from different kinds of distributions for a future version of DynamicCloudSim, we found the normal distribution to be a good starting point.

Dejun et al. [7] reported average runtimes between 200 and 900 ms – with a mean and standard deviation of roughly 500, respectively 200 – for executing a CPU-intensive task on 30 VMs across six Amazon EC2 datacenters. Based on these measurements, we set the default value for the RSD parameter responsible for CPU performance heterogeneity to 0.4. In the same way, we determined a default value for I/O heterogeneity of 0.15. Based on similar measurements taken by Jackson et al. [9] for communication-intensive tasks on Amazon EC2, we set the default value for network bandwidth heterogeneity to 0.2. These values are backed up by the performance measurements of Schad et al. [12], who observed an RSD of 0.35 between processor types in EC2 as well as RSD values of 0.2 and 0.19 for disk I/O and network performance.

3.4 Dynamic Changes at Runtime

So far we only modeled heterogeneity, which represents permanent variance in performance of VMs due to differences in underlying hardware. Another important concept of instability inherent to cloud computing are dynamic changes of performance at runtime *(DCR)* due to sharing of common resources with other VMs and users, as for instance reported by Dejun et al. [7]. DynamicCloudSim attempts to capture two aspects of this concept: (1) Long-term changes in a VM's performance due to a certain event, e.g., the co-allocation of a different VM with high resource utilization on the same host. (2) Uncertainty or noise, which models short-term alterations in a VM's performance.

To simulate the effects of long-term changes, DynamicCloudSim samples from an exponential distribution with a given rate parameter to determine the time of the next performance change. The exponential distribution is frequently used to model the time between state changes in continuous processes. In DynamicCloudSim, the rate parameter is defined by the user and corresponds to the average number of performance changes per hour. In light of the observations made by Schad et al. [12] and Iosup et al. [13], who found that the performance of a VM can change on an hourly basis, we assume the performance of a VM to change about once every two hours by default.

Whenever a change of one of the performance characteristics has been induced on a VM, the new value for the given characteristic is sampled from a normal distribution. The mean of this distribution is set to the baseline value of the given characteristic for this VM, i.e., the value that has been assigned to the VM at allocation time. The RSD of the distribution is once again set by the user. Higher values in both the rate parameter of the exponential distribution and the standard deviation of the normal distribution correspond to higher levels of dynamics.

Uncertainty (or noise) is the lowest tier of dynamic performance changes in DynamicCloudSim. It is modeled by introducing slight aberrations to a VM's performance whenever a task is assigned to it. As with heterogeneity and dynamics, this is achieved by sampling from a normal distribution with user-defined RSD parameter.

On Amazon EC2, Dejun et al. [7] observed relative standard deviations in performance between 0.019 and 0.068 for CPU-intensive tasks and between 0.001 and 0.711 for I/O-intensive tasks. We set the default values for the RSD parameter of long-term performance changes to the third quartile of these distributions, i.e., to 0.054 for CPU performance and 0.033 for I/O performance. Similarly, we set the default RSD value for the noise parameter to the first quartile, i.e., to 0.028 for CPU and 0.007 for I/O.

3.5 Stragglers and Failures

In massively parallel applications on distributed computational infrastructure, fault-tolerant design becomes increasingly important [30]. For the purpose of simulating fault-tolerant approaches to scheduling, DynamicCloudSim introduces straggler VMs and failures (*SaF*) during task execution. Stragglers are virtual machines exhibiting constantly poor performance [8]. In DynamicCloudSim, the probability of a VM being a straggler can be specified by the user along with the coefficient that determines how much the performance of a straggler is diminished.

We propose default values of 0.015, respectively 0.5 for the straggler likelihood and performance coefficient parameters. These values are based on the findings of Zaharia et al. [8], who encountered three stragglers with performance diminished by 50 % or more among 200 provisioned VMs in their experiments. The effect of these parameters is exemplarily shown in Figure 3, which illustrates all of the introduced factors of instability (Het, DCR, SaF) in combination for the CPU performance of eight VMs, including one straggler, in an experiment of 12 hours in DynamicCloudSim.

Failures during task execution are another factor of instability inherent to distributed computing. DynamicCloudSim is currently confined to a basic method of failure generation: Whenever a task is assigned to a VM and its execution time is computed, DynamicCloudSim determines whether the task is bound to succeed or fail. This decision is based on the average rate of failures specified by the user. The default value for the rate of failed task executions is set to 0.002, based on the observations of Jackson et al. [9], who had to restart one out of ten experiments on 50 VMs due to the occurrence of a failure on one VM.

There are various reasons for failed task execution, such as temporary performance breakdowns within a VM or the inability to access input data or write output data. Usually, such perturbations are not immediately recognized, hence resulting in severely increased runtimes. Consequently, in DynamicCloudSim the runtime of a failed task execution is determined by multiplying the task's execution time with a user-defined coefficient. The introduction of more sophisticated failure models on different levels (VM, storage, task) of workflow execution is left for future work (see Section 7).

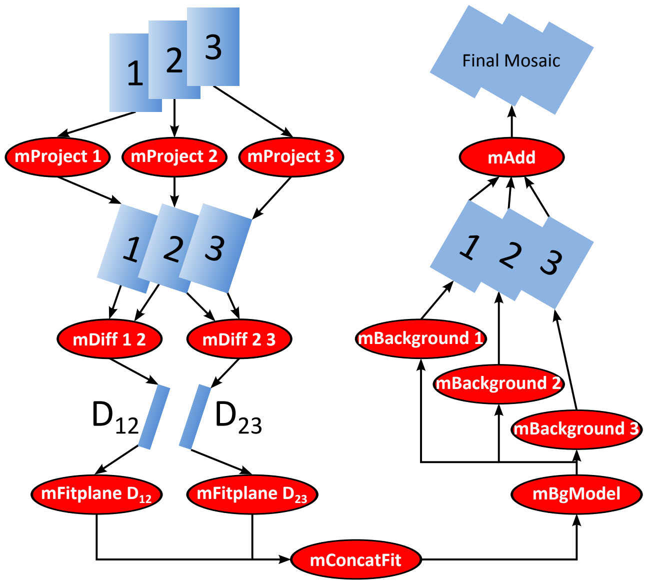

Figure 4: A schematic instance of the Montage workflow, in which three input images are processed to generate a mosaic [31].

4. EXPERIMENTAL VALIDATION

To showcase a possible application of DynamicCloudSim, we simulate the execution of a computationally intensive workflow using different mechanisms of scheduling and different levels of instability in the computational infrastructure. We expect the schedulers to differ in their robustness to instability, which should be reflected in diverging workflow execution times. In this section, we outline the evaluation workflow, the experimental settings, and the schedulers which we used in our experiments.

4.1 The Montage Workflow

We constructed an evaluation workflow using the Montage toolkit [31]. Montage is able to generate workflows for assembling high-resolution mosaics of regions of the sky from raw input data. It has been repeatedly utilized for evaluating scheduling mechanisms or computational infrastructures for scientific workflow execution (e.g., [32–35]). See Figure 4 for a schematic visualization of the Montage workflow and Figure 5 for an example of output generated by a Montage workflow.

In our experiments, we used a Montage workflow which builds a large-scale (twelve square degree) mosaic of the m17 region of the sky. This workflow consists of 43,318 tasks reading and writing 534 GB of data in total, of which 10 GB are input and output files which have to be uploaded to and downloaded from the computational infrastructure.

4.2 Experimental Settings

The 43,318 task Montage workflow was executed on a single core of a Dell PowerEdge R910 with four Intel Xeon E7-4870 processors (2.4 GHz, 10 cores) and 1 TB memory, which served as the reference machine of our experiments. Network file transfer, local disk I/O and the runtime of each task in user-mode were captured and written to a trace file.

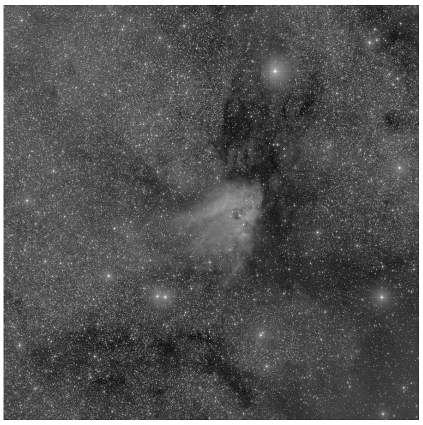

Figure 5: A one square degree mosaic of the m17 region of the sky. The image has been generated by executing the corresponding Montage workflow.

We parsed the Montage workflow and the trace it generated on the Xeon E7-4870 machine in DynamicCloudSim. The 43,318 tasks were assigned performance requirements according to the trace file, i.e., a CPU workload corresponding to the execution time in milliseconds, an I/O workload equal to the file sizes of the task's input and output files, and a network workload according to the external data transfer caused by the task. When executing the workflow in CloudSim, all data dependencies were monitored. Thus, a task could not commence until all of its predecessor tasks had finished execution.

In our simulations, we attempt to mirror the computational environment of Amazon EC2. Hence, we obtained SPECfp® 2006 benchmark results for Intel Xeon E5430 2.66 GHz, AMD Opteron 270 2.0 GHz, and AMD Opteron 2218 HE 2.6 GHz, which Jackson et al. [9] detected in their evaluation of Amazon EC2 as underlying hardware. SPECfp® 2006 is the floating point component of the SPEC® CPU2006 benchmark suite. It provides a measure of how fast a single-threaded task with many floating point operations is completed on one CPU core. An overview of the benchmark results is displayed in Table 1.

A CloudSim datacenter was initialized with 500 host machines: 100 Xeon E5430, 200 Opteron 2218, and 200 Opteron 270. Since the Xeon E5430 has twice as many cores as the AMD machines, each type of machine contributes to the datacenter with an equal amount of cores and thus compute units. The CPU performance of each core of these machines was set to the ratio of the machine's SPECfp® 2006 score to the reference machine's score. For instance, the CPU performance of Xeon E5430 machines was set to 355, effectuating in a runtime of 28,169 milliseconds for a task that took 10,000 milliseconds on the Xeon E7-4870 reference machine.

We assume all of the data associated with the workflow – input, intermediate, and output – to be saved on shared storage such as Amazon S3. Different measurements of network throughput within Amazon EC2 and S3 ranging from 10 to 60 MB/s have been reported (e.g., [9, 36, 37]). We therefore set the default I/O throughput of virtual machines to 20 MB/s. The external bandwidth of virtual machines was set to 0.25 MB/s, based on the remote access performance of S3 reported by Palankar et al. [11] and Pelletingeas [37].

In the course of the experiments, we incrementally raised the level of instability in DynamicCloudSim. In total, we conducted four experiments, in which we measured the effect of heterogeneity (Het), dynamic performance changes at runtime (DCR), and straggler VMs and faulty task executions (SaF):

1. **Het**: We measure the effect of heterogeneous computational infrastructure on different approaches to workflow scheduling. To this end, the relative standard deviation (RSD) parameters responsible for inhomogeneity are incrementally set to 0, 0.125, 0.25, 0.375, and 0.5 (for CPU, I/O, and network performance). The simulation of dynamic performance changes at runtime (DCR) as well as straggler VMs and failed tasks (SaF) is omitted.

2. **DCR**: We examine how dynamic changes in the computational infrastructure affect workflow scheduling. Therefore, the RSD parameters responsible for long-term changes in the performance of a VM was varied between 0, 0.125, 0.25, 0.375, and 0.5. At the same time, the rate of performance changes is fixed at 0.5 and the RSD parameters for noise are set to 0.025 across all runs.

3. **SaF**: We determine the effect of straggler resources and failures during task execution. For this reason, the likelihoods of a VM being a straggler and of a task to fail are set to 0, 0.00625, 0.0125, 0.01875, and 0.025. The performance coefficient of straggler resources is set to 0.1 and the factor by which the runtime of a task increases in the case of a failure is set to 20.

4. **Extreme parameters**: In this setting, we utilize 1.5 times the maximum values for heterogeneity, dynamics and stragglers / failures from experiments 2, 3, and 4 (i.e., RSD parameters of 0.75 for Het and DCR; straggler and failure likelihoods of 0.0375 for SaF) to determine the effect of combining all introduced factors of instability at a very high level.

4.3 Scientific Workflow Schedulers

Scheduling a scientific workflow denotes the process of mapping the workflow's tasks onto the available computational resources [24]. Most scheduling policies are developed with the aim to minimize overall workflow execution time. However, certain scenarios call for different scheduling objectives, e.g., the optimization of monetary cost or data security [21]. In general, we differentiate between static and adaptive schedulers [14]. Here we consider (1) a static round robin scheduler, (2) the HEFT scheduling heuristic [23], (3) a greedy task queue, and (4) the LATE algorithm [8]. All of the schedulers outlined in this section were implemented in DynamicCloudSim.

4.3.1 Static Schedulers

In static scheduling, a schedule is assembled prior to execution and then strictly abided at runtime. The round robin scheduler is often used as a baseline implementation of a static scheduler (e.g., [20]). It constructs a schedule by traversing the workflow from the beginning to the end, assigning tasks to computational resources in turn. This way, each resource will end up with roughly the same amount of tasks, independent of its computational capabilities or the

Table 1: CFP2006 benchmark results

machine	cores	SPECfp® 2006 base score	percentage of reference	URL
Intel Xeon E7-4870 2.4 GHz	10	51.0	100 %	http://tinyurl.com/d3oghak
Intel Xeon E5430 2.66 GHz	8	18.1	35.5 %	http://tinyurl.com/bckaqow
AMD Opteron 2218 2.6 GHz	4	12.6	24.7 %	http://tinyurl.com/ajqj3n3
AMD Opteron 270 2.0 GHz	4	8.89	17.4 %	http://tinyurl.com/aug9xcq

tasks' workload. We expect the static round robin scheduler to perform well in homogeneous and stable computational infrastructures. Adding heterogeneity (Het), dynamic changes at runtime (DCR) or stragglers and failures (SaF) to the experiment should heavily diminish its performance.

A number of more sophisticated mechanisms for static workflow scheduling on heterogeneous computational architectures have been proposed (e.g., [16, 24, 25]). Among the more influential scheduling algorithms is the Heterogeneous Earliest Finishing Time (HEFT) heuristic, which has been developed by Topcuoglu et al. [23]. Similar to most sophisticated static schedulers, HEFT requires runtime estimates for the execution of each task on each computational resource, which are difficult to obtain.

The HEFT heuristic traverses the workflow from the end to the beginning, computing the upward rank of each task as the estimated time to overall workflow completion at the onset of this task. The computation of a given task's upward rank incorporates estimates for both the runtimes and data transfer times of the given task as well as the upward ranks of all successor tasks. The static schedule is then assembled by assigning each task in decreasing order of upward ranks a time slot on a computational resource, such that the task's scheduled finish time is minimized.

In our experiment, HEFT is provided with accurate runtime estimates based on the execution time of each task on each CloudSim VM at the time of its allocation. Hence, we expect the HEFT scheduler to perform well in both homogeneous and heterogeneous infrastructure. However, we expect poor performance if dynamic changes (DCR) or failures in the computational infrastructure (SaF) are introduced and runtime estimates become inaccurate.

Both the static round robin scheduler and the HEFT scheduler are available in the SWfMS Pegasus [20]. The SWfMS Kepler [19] also supports static scheduling by means of its SDF director.

4.3.2 Adaptive Schedulers

In adaptive scheduling, scheduling decisions are made on-the-fly at the time of execution. The most intuitive approach to adaptive scheduling is a greedy task queue. Here, tasks are assigned to computational resources in first-come-first-serve manner at runtime. Whenever a resource has an available task slot, it fetches a task from a queue of tasks ready for execution. Task queues have been implemented in a variety of SWfMS, including Taverna [18] and Kepler [19]. The default scheduler of Hadoop [38] also employs a greedy queue. In our experiment, we expect a task queue scheduler to outperform static schedulers when dynamic changes (DCR, SaF) in the computational infrastructure are introduced.

The LATE (Longest Approximate Time to End) scheduler developed by Zaharia et al. [8] constitutes a well-established alteration of the default task queue. By speculatively replicating tasks progressing slower than expected, LATE exhibits increased robustness to the effects of straggler resources and failed task execution (SaF). LATE keeps track of the runtime and progress of all running tasks. By default, 10 % of the task slots on resources performing above average are assigned speculative copies of tasks which are estimated to finish farthest into the future and have progressed at a rate below average. Intuitively, this approach maximizes the likeliness for a speculative copy of a task to overtake its original. LATE evidently follows a rationale similar to that of HEFT, since both scheduling heuristics prioritize the assignment of tasks with longest times to finish to well-performing computational resources.

LATE was implemented as an extension of Hadoop's default scheduling algorithm. For Hadoop's Sort benchmark executed on 800 virtual machines of an Amazon EC2 test cluster, the LATE scheduler has been shown to outperform the default scheduler of Hadoop by 27 % on average [8]. In our experiments, we expect LATE to be robust even in settings with straggler resources and high rates of failures during task execution (SaF). However, due to 10 % of the computational resources being reserved for speculative task execution, LATE should perform slightly inferior to a greedy queue on homogeneous and stable computational infrastructure.

5. RESULTS AND DISCUSSION

For each configuration, the 43,318 task Montage workflow described in Section 4.1 was executed 100 times on eight virtual machines and the average runtime was determined. The results of the four experiments outlined in Section 4.2 are displayed in Figures 6 and 7. Over the course of the entire experiments, average runtimes of the Montage workflow between 296 and 13,195 minutes have been observed. Clearly, the instability parameters provided by DynamicCloudSim have a considerable impact on execution time.

In the experiment simulating the effect of heterogeneous resources (Het), all schedulers except the static round robin scheduler exhibit robustness to even the highest levels of variance (see Figure 6a). The reasons for this observation are that HEFT has been designed specifically with inhomogeneous computational resources in mind and queue-based schedulers like the greedy scheduler and LATE are able to adapt to the computational infrastructure. All three schedulers effectively assign less tasks to slower resources. Conversely, the static round robin scheduler is oblivious to the computational infrastructure and simply assigns an equal amount of tasks to each resource, which results in faster resources having to idly wait for slower resources to finish.

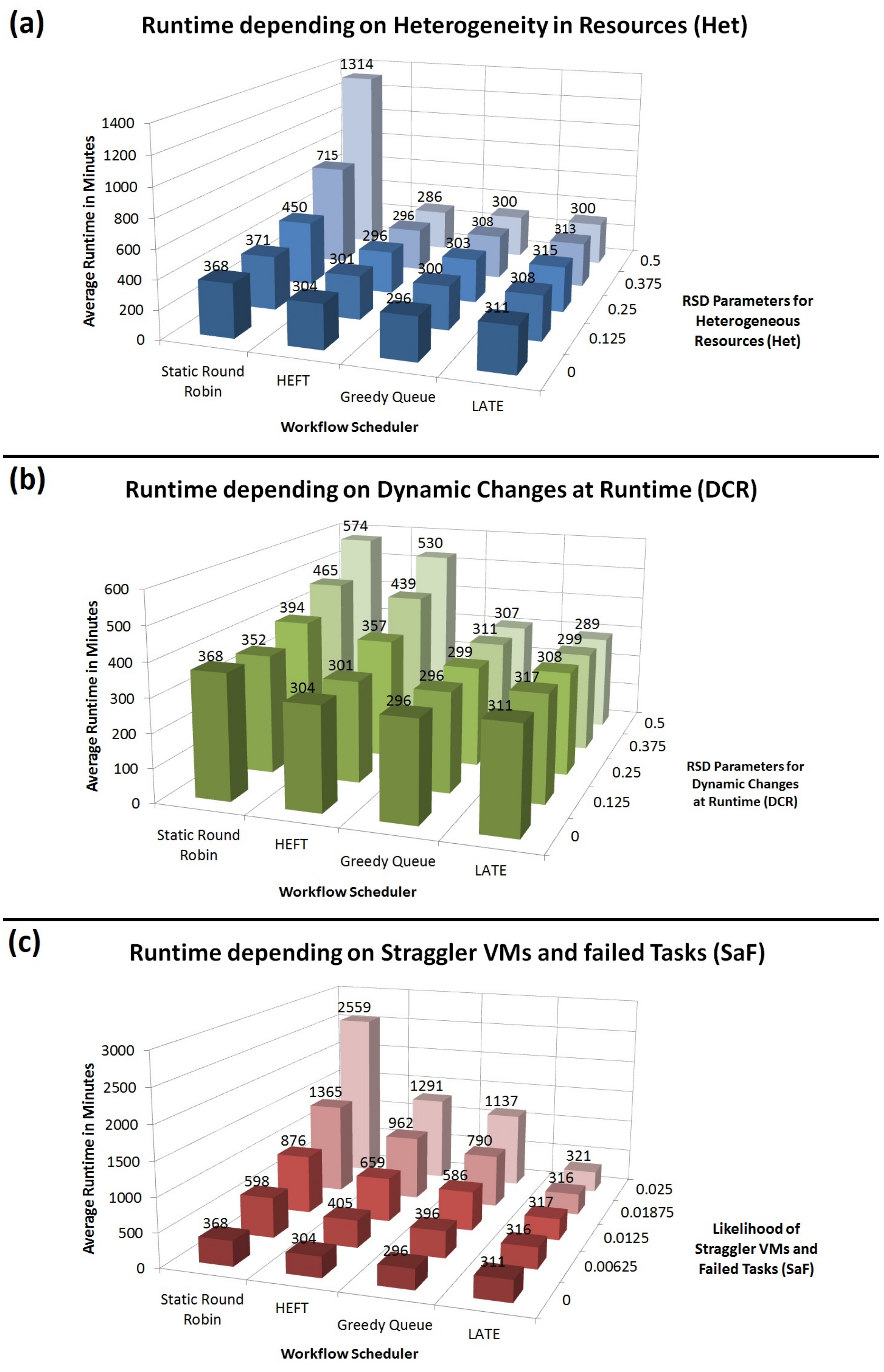

Figure 6: Effects of (a) heterogeneity (Het), (b) dynamic changes of performance at runtime (DCR), and (c) straggler VMs and failed tasks (SaF) on execution time of the Montage workflow using different schedulers.

Since LATE always reserves 10 % of the available resources for speculative scheduling, we would expect runtimes slightly below a greedy queue, which we did not observe. The reason for this might be that HEFT and LATE have a slight edge over the greedy queue-based scheduler: If there is a computationally intensive task which blocks the execution of all successor tasks – of which there is one in the Montage workflow called mBgModel –, HEFT and LATE are able to assign it to a well-suited computational resource, instead of simply assigning it to the first available resource. HEFT does this by consulting the accurate runtime estimates of all task-resource-assignments it has been provided with. LATE simply starts a speculative copy of the task on a compute node performing above average.

Finding only the static round robin scheduler to perform subpar in this experimental setting confirmed our expectations outlined in Section 4.3. Evidently, DynamicCloudSim is able to simulate the effect of inhomogeneous resources. Since heterogeneity is commonly encountered in distributed architectures like computational clouds, this is a very desirable property which will continue to be important going forward and has not been sufficiently supported by other cloud simulation toolkits.

The second experiment measured how dynamic changes in the performance of VMs (DCR) affect the runtime of the Montage workflow achieved by the four scheduling mechanisms (see Figure 6b). It confirms our expectations of static schedulers like static round robin and HEFT not being able to handle dynamic changes. The major shortcoming of static schedulers lies in the fact that they assemble a schedule prior to workflow execution, which is then strictly abided to. Therefore, changes in the runtime environment make even elaborate static schedules suboptimal.

HEFT provided with accurate runtime estimates constitutes one of the most sophisticated static scheduling policies available, since it takes into account the suitability of a given task for each resource. The only scenarios, in which HEFT should perform substantially worse than a greedy task queue, should be ones in which all tasks have equal performance requirements (which is not the case in Montage) or the runtime estimates are inaccurate, e.g., due to alterations at runtime. Hence, the findings of the second experiment are a strong indicator of DynamicCloudSim being able to simulate dynamic changes in the performance of resources.

The third experiment examined how the appearance of straggler VMs (i.e., virtual machines with only 10 % of a regular machine's performance) and failed task executions (SaF) influence the performance of the four examined workflow schedulers. Figure 6c confirms the robustness of the LATE scheduler even for high amounts of failures and stragglers. In contrast, the performance of all other schedulers diminished quickly in the face of failure. This is not surprising, since if critical tasks are assigned to straggler VMs or encounter a failure during execution, overall workflow execution time can increases substantially. Speculative replication of tasks with a low progress rate alleviates this problem.

In the fourth experiment, we examined how all three of the introduced factors of instability combined and taken to extremely high levels (RSD parameters of 0.75 for Het and DCR; likelihood parameters of 0.0375 for SaF) influenced

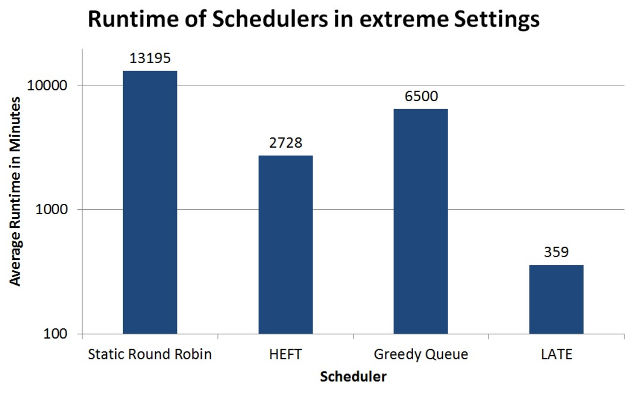

Figure 7: Execution time (in log scale) of the Montage workflow in DynamicCloudSim in extreme cases of instability.

the workflow execution time. The results of this experiment are shown in Figure 7. Once again, LATE is the only scheduler to exhibit robustness to even extreme parameter configurations. Furthermore and in contrast to the findings in the third experiment, the HEFT scheduler substantially outperforms the greedy job queue.

The combination of all factors, i.e., dynamic changes at runtime to inhomogeneous compute resources which can also be stragglers or subject to faulty task execution, can lead to cases, in which the execution of the Montage workflow task mBgModel (as described above) can take extremely long. This is more problematic for a greedy task queue, which assigns a task to the first available computational resource, which might be a straggler. In contrast, HEFT is at least able to handle heterogeneous and straggler resources by means of its accurate runtime estimates.

The last two experiments illustrated the severe effect of straggler VMs and failed tasks executions (SaF) on workflow runtime, confirming previous reports on the importance of fault-tolerant design in computationally intensive applications (e.g., [30, 35]). While the simulation was able to replicate the advertised strengths of LATE, we acknowledge that more sophisticated failure models would be a desirable enhancement over the current implementation of DynamicCloudSim (see Section 7).

All in all, the experiments clearly confirmed the expectations described in Section 4.3. While a validation on actual cloud infrastructure is still pending, the results serve as an indicator of DynamicCloudSim being well-suited to simulate the dynamics and instability inherent to computational clouds. Furthermore, the simulations underline the importance of adaptive scheduling of scientific workflows in shared and distributed computational infrastructures like public clouds.

6. RELATED WORK

Merdan et al. [39] and Hirales-Carbajal et al. [40] developed simulation environments specifically for comparing different approaches to workflow scheduling on computational grids. They also provide examples of possible experimental setups, yet omit the execution of these experiments. Our work differs from these publications in three ways: Firstly, by extending a universal simulation framework like Cloud-

Table 2: Features of CloudSim, WorkflowSim, and DynamicCloudSim

Feature	CloudSim	WorkflowSim	DynamicCloudSim
performance characteristics MIPS, bandwidth, memory	✓	✓	✓
performance characteristic file I/O			✓
runtime of a task depending on values other than MIPS			✓
modeling of data dependencies	✓	✓	✓
workflow parsing		✓	✓
implementation of workflow schedulers		✓	✓
modeling of delays at different layers of a SWfMS		✓	
support for task clustering		✓	
different VMs on different hosts	✓	✓	✓
random assignment of new VM to a host			✓
resource allocation based on compute units			✓
dynamic changes of VM performance at runtime			✓
modeling of failures during task execution		✓	(✓)
introduction of straggler VMs			✓

Sim, DynamicCloudSim is not limited to the field of scientific workflows, but can be utilized for simulation of any cloud application. Secondly, our work puts a strong emphasis on instabilities in the computational infrastructure, which is important to achieve realistic results. Thirdly, we conduct an experimental validation of the changes added to the simulation toolkit.

Chen and Deelman [35] recently presented WorkflowSim as another extension to CloudSim. WorkflowSim is tightly bound to the SWfMS Pegasus [20] and adds to CloudSim (1) the workflow engine underlying Pegasus and DagMan [41], (2) an elaborate model of node failures, (3) a model of delays occurring in the various levels of the Pegasus stack (e.g., queue delays, pre/post-processing delays, data transfer delays), and (4) the implementations of several workflow schedulers implemented in Pegasus (e.g., greedy task queue, HEFT [23], Min-Min, and Max-Min [24]). Parameters are directly learned from traces of real executions. WorkflowSim follows a quite different approach than DynamicCloudSim: WorkflowSim models delays in the Pegasus workflow stack and is thus tightly coupled to Pegasus. It has no notion of heterogeneous hardware or variance in available resources. In contrast, DynamicCloudSim directly models instability and heterogeneity in the environment in which a workflow (or any other collection of computationally intensive tasks) is executed and is thus independent of a concrete system. See Table 2 for a comparison of features available in CloudSim, WorkflowSim, and DynamicCloudSim.

Donassolo et al. [42] altered the SimGrid framework [43], another popular grid simulator besides GridSim [26] to increase the scalability of simulation runs. These improvements allow SimGrid to simulate the computation of workloads on tens or hundreds of thousands of heterogeneous and possibly volatile machines as encountered in volunteer computing. For the same reasons of increasing scalability, Ostermann et al. [44] also developed GroudSim, another toolkit for simulating the execution of scientific applications in a computational grid or cloud. Scalability has been previously reported to be an issue of GridSim [45] – and thus also CloudSim – for numbers of worker machines above 10,000. However, due to cost limitations we expect the number of VMs provisioned for executing computationally intensive tasks in a computational cloud to usually be below 10,000. Hence, we designed DynamicCloudSim as an extension of CloudSim, which has the key advantage of it being usable in conjunction with many of the valuable extensions of CloudSim, such as EMUSIM [28] or WorkflowSim [35].

7. CONCLUSION AND FUTURE WORK

We presented DynamicCloudSim as an extension to CloudSim, a popular simulator for evaluating resource allocation and scheduling strategies on distributed computational architectures. We enhanced CloudSim's model of cloud computing infrastructure by introducing models for (1) inhomogeneity in the performance of computational resources, (2) uncertainty in and dynamic changes to the performance of VMs, and (3) straggler VMs and failures during task execution. We showed that applying these models to scientific workflow execution using four established scheduling algorithms replicated the known strengths and shortcomings of these schedulers, which underlined the importance of adaptivity in scheduling of scientific workflows on shared and distributed computational infrastructure.

In the experiments described in this paper, each virtual machine only processed one task at a time. Furthermore, issues of data locality were not incorporated yet, since we assumed all files (input, intermediate, and output) to be read from and written to shared network storage, such as Amazon S3. In future work, we would like to revisit the experiments, adding additional task slots per virtual machine and investigating the influence of storing files locally on each VM.

Another area of future research involves the integration of DynamicCloudSim with WorkflowSim [35] to harness the combined functionality of both CloudSim extensions. For instance, the elaborate and multi-layered failure models of WorkflowSim could further enhance the model of instability introduced by DynamicCloudSim.

Most importantly, we intend to showcase that DynamicCloudSim is able to adequately model the behavior of computational clouds by comparing our findings against traces of workflow execution on actual hardware. Furthermore, we plan to incorporate workflows other than Montage into this extended analysis.

Funding

Marc Bux is funded by the Deutsche Forschungsgemeinschaft through graduate school SOAMED (GRK 1651). We further acknowledge support from the European Commission through the BiobankCloud project.

References

[1] P. Mell, T. Grance, *The NIST Definition of Cloud Computing*, National Institute of Standards and Technology (2009).

[2] I. Foster, Y. Zhao, I. Raicu, S. Lu (2008), *Cloud Computing and Grid Computing 360-Degree Compared*, in: Proceedings of the 1st Workshop on Grid Computing Environments, Austin, Texas, pp. 1–10.

[3] A. Beloglazov, R. Buyya (2012), *Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers*, Concurrency and Computation: Practice and Experience 24(13):1397–1420.

[4] L. Wu, S. K. Garg, R. Buyya (2011), *SLA-Based Resource Allocation for Software as a Service Provider in Cloud Computing Environments*, in: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ieee, Newport Beach, California, USA (2011), pp. 195–204.

[5] S. Sadhasivam, N. Nagaveni, R. Jayarani, R. V. Ram (2009), *Design and Implementation of an Efficient Two-level Scheduler for Cloud Computing Environment*, in: Proceedings of the 2009 International Conference on Advances in Recent Technologies in Communication and Computing, Kottayam, India, pp. 884–886.

[6] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya (2011), *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software - Practice and Experience 41(1):23–50.

[7] J. Dejun, G. Pierre, C.-H. Chi (2009), *EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications*, in: Proceedings of the 7th International Conference on Service Oriented Computing, Stockholm, Sweden, pp. 197–207.

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, I. Stoica (2008), *Improving MapReduce Performance in Heterogeneous Environments*, in: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, San Diego, USA, pp. 29–42.

[9] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright (2010), *Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud*, in: Proceedings of the 2nd International Conference on Cloud Computing Technology and Science, Indianapolis, USA, pp. 159–168.

[10] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, *An Early Performance Analysis of Cloud Computing Services for Scientific Computing*, TU Delft (2008).

[11] M. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel (2008), *Amazon S3 for Science Grids: a Viable Solution?*, in: Proceedings of the 1st Workshop on Data-aware Distributed Computing, Boston, USA, pp. 55–64.

[12] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz (2010), *Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance*, Proceedings of the VLDB Endowment 3(1):460–471.

[13] A. Iosup, N. Yigitbasi, D. Epema (2011), *On the Performance Variability of Production Cloud Services*, in: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, California, USA, pp. 104–113.

[14] M. Bux, U. Leser (2013), *Parallelization in Scientific Workflow Management Systems*, Computing Research Repository (CoRR).

[15] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswarans, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund (2001), *A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*, Journal of Parallel and Distributed Computing 61:810–837.

[16] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy (2005), *Task Scheduling Strategies for Workflow-based Applications in Grids*, in: Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid, volume 2, Cardiff, UK, pp. 759–767.

[17] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers (2007), *Examining the Challenges of Scientific Workflows*, IEEE Computer 40(12):24–32.

[18] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li (2004), *Taverna: a tool for the composition and enactment of bioinformatics workflows*, Bioinformatics 20(17):3045–3054.

[19] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao (2006), *Scientific Workflow Management and the Kepler System*, Concurrency and Computation: Practice and Experience 18(10):1039–1065.

[20] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, D. S. Katz (2005), *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*, Scientific Programming 13(3):219–237.

[21] J. Yu, R. Buyya (2005), *A Taxonomy of Workflow Management Systems for Grid Computing*, Journal of Grid Computing 3:171–200.

[22] C. Goble, D. de Roure (2007), *myExperiment: Social Networking for Workflow-using e-Scientists*, in: Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science, Monterey, USA, pp. 1–2.

[23] H. Topcuoglu, S. Hariri, M. Wu (2002), *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*, IEEE Transactions on Parallel and Distributed Systems 13(3):260–274.

[24] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, L. Johnsson (2005), *Scheduling Strategies for Mapping Application Workflows onto the Grid*, in: Proceedings on the 14th IEEE International Symposium on High Performance Distributed Computing, Durham, USA, pp. 125–134.

[25] J. Yu, R. Buyya (2006), *A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms*, in: Proceedings of the 1st Workshop on Workflows in Support of Large-Scale Science, Paris, France.

[26] R. Buyya, M. Murshed (2002), *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, Concurrency and Computation: Practice and Experience 14(13-15):1175–1220.

[27] S. K. Garg, R. Buyya (2011), *NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations*, in: Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing, Ieee, Melbourne, Australia (2011), pp. 105–113.

[28] R. N. Calheiros, M. A. S. Netto, C. A. F. De Rose, R. Buyya (2013), *EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications*, Software - Practice and Experience 43:595–612.

[29] S. Frey, W. Hasselbring (2011), *The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications*, International Journal on Advances in Software 4(3 and 4):342–353.

[30] B. Schroeder, G. A. Gibson (2006), *A large-scale study of failures in high-performance-computing systems*, in: Proceedings of the 36th International Conference on Dependable Systems and Networks, Philadelphia, USA, pp. 249–258.

[31] G. B. Berriman, E. Deelman, J. Good, J. Jacob, D. S. Katz, C. Kesselman, A. Laity, T. A. Prince, G. Singh, M.-h. Su (2004), *Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand*, in: Proceedings of the SPIE Conference on Astronomical Telescopes and Instrumentation, volume 5493, Glasgow, Scotland, pp. 221–232.

[32] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good (2008), *The Cost of Doing Science on the Cloud: The Montage Example*, in: Proceedings of the 2008 Conference on Supercomputing, Ieee, Austin, Texas (2008).

[33] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good (2008), *On the Use of Cloud Computing for Scientific Workflows*, in: Proceedings of the 4th IEEE International Conference on eScience, Indianapolis, USA, pp. 640–645.

[34] K. Lee, N. W. Paton, R. Sakellariou, E. Deelman, A. A. A. Fernandes, G. Mehta (2009), *Adaptive Workflow Processing and Execution in Pegasus*, Concurrency and Computation: Practice and Experience 21(16):1965–1981.

[35] W. Chen, E. Deelman (2012), *WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments*, in: Proceedings of the 8th IEEE International Conference on eScience, Chicago, USA, pp. 1–8.

[36] S. L. Garfinkel, *An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS*, Technical Report TR-08-07, School for Engineering and Applied Sciences, Harvard University, MA (2007).

[37] C. Pelletingeas, Performance Evaluation of Virtualization with Cloud Computing, Master of engineering thesis, Edinburgh Napier University (2010).

[38] T. White (2012), *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., Sebastopol, USA, 3rd edition.

[39] M. Merdan, T. Moser, D. Wahyudin, S. Biffl, P. Vrba (2008), *Simulation of workflow scheduling strategies using the MAST test management system*, in: Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision, Hanoi, Vietnam, pp. 1172–1177.

[40] A. Hirales-Carbajal, A. Tchernykh, T. Röblitz, R. Yahyapour (2010), *A Grid Simulation Framework to Study Advance Scheduling Strategies for Complex Workflow Applications*, in: Proceedings of the 24th IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Atlanta, USA.

[41] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger (2007), *Workflow Management in Condor*, in: I. J. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer, New York, USA, 1st edition, pp. 357–375.

[42] B. Donassolo, H. Casanova, A. Legrand, P. Velho (2010), *Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid*, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, USA, pp. 605–612.

[43] H. Casanova, A. Legrand, M. Quinson (2008), *SimGrid: A Generic Framework for Large-Scale Distributed Experiments*, in: Proceedings of the Tenth International Conference on Computer Modeling and Simulation, Cambridge, UK, pp. 126–131.

[44] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer (2010), *GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds*, in: CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing in conjunction with EuroPAR 2010, Ischia, Italy.

[45] W. Depoorter, N. D. Moor, K. Vanmechelen, J. Broeckhove (2008), *Scalability of Grid Simulators: An Evaluation*, in: Proceedings of the 14th international Euro-Par conference on Parallel Processing, Las Palmas de Gran Canaria, Spain, pp. 544–553.