PriDynSim: A Simulator for Dynamic Priority Based I/O Scheduling for Cloud Applications

Nitisha Jain*, Nikolay Grozev†, J. Lakshmi*, Rajkumar Buyya†
*Supercomputer Education and Research Center,
Indian Institute of Science, Bangalore, India
{sercnitisha@ssl.serc.iisc.in, jlakshmi@serc.iisc.in}

†Cloud Computing and Distributed Systems (CLOUDS) Laboratory,
Department of Computing and Information Systems,
The University of Melbourne, Parkville, Australia
{ngrozev@student.unimelb.edu.au, rbuyya@unimelb.edu.au}

***Abstract*—Cloud computing facilitates flexible on-demand provisioning of IT resources over the Internet. It has proven to be very advantageous for a wide range of industries in the emerging markets by allowing them to match the infrastructure capabilities of their already established competitors. However, Cloud computing also poses several unique challenges in terms of resource allocation and scheduling. In particular, I/O bound applications have been mostly neglected in the research efforts in the area. Simulation tools play a key role in the evaluation of new resource management policies by providing an affordable and replicable testing environment. In this paper, we propose a novel simulation framework *PriDynSim* for priority based I/O policies, which manage the available resources to ensure adequate performance in the presence of deadline constraints. In a case study, we demonstrate how *PriDynSim* can be used to evaluate a resource management policy, which guarantees Quality of Service (QoS) for I/O bound applications running in a typical Cloud environment.**

***Index Terms*—Simulation, I/O policy, priority scheduling, performance QoS.**

I. Introduction

Cloud computing has recently emerged as a disruptive IT model for renting and using computational resources on demand. It enables users to flexibly and dynamically manage IT resources and thus has gained widespread popularity. It offers on-demand availability and scalability without the cost of maintenance for businesses, regardless of their size. This flexibility is especially important for enterprises in the emerging markets, as it allows them to focus on their core lines of business, rather than invest in developing and managing IT infrastructure. If it is adopted timely, Cloud computing can give such organizations a key competitive advantage over their competitors.

In order to leverage the full advantages of Cloud computing environments, there is a need to progressively explore new architectures and policies to solve the issues associated with the present Cloud provisioning model. While the area of CPU resource provisioning and scheduling in clouds has gained significant interest over the last few years, few works have investigated I/O resource management policies. One of the major impediments for the research in the area is the lack of affordable testing environments. Simulation tools provide a good alternative to testing in large scale and expensive testbeds, by providing a controlled environment for hypotheses evaluation. The availability of cloud performance simulators like *CloudSim* [1] has already encouraged the research in the area of cloud resource management and these are widely used in both academia and industry. Many organizations in the emerging market economies can not afford expensive testbeds and thus simulation environments like *CloudSim* are invaluable. They allow such enterprises to quickly evaluate novel resource management techniques, thus fostering their innovation and putting them at par with more resourceful competitors. *CloudSim* provides a flexible and customizable platform for the modeling of Cloud data-centers, services, brokers, virtualized servers as well as network topologies and also energy-aware computation resources. Unfortunately, *CloudSim* lacks proper support and representation of disk I/O operations and policies for allocation of I/O resources to concurrent I/O workloads. The advent of a suitable I/O performance model extending the widely adopted *CloudSim* simulator holds the potential to further advance the research in I/O resource management.

Recently, *CloudSim* extensions modelling I/O operations alongside CPU instructions for each job (called 'cloudlet' in CloudSim) have been proposed [2]. However, the policies for provisioning of disk resources in terms of I/O instructions per second or IOPS for the I/O operations have not been focused upon, with the default scheduling behavior being fair-sharing i.e. resources are equally assigned to all the cloudlets that are contending for disk access at a given time irrespective of their requirements. In [3], it has been demonstrated that multiple concurrent I/O applications in a Cloud setup need to be assigned resources in accordance with their requirements (in terms of deadlines for completion) to ensure that their performance is acceptable to the users. Hence, there is a need for an I/O resource scheduler, which can assign resources to applications based on their specific characteristics and requirements.

In this paper, we introduce a generic *CloudSim* based

simulator named *PriDynSim* for priority based I/O resource scheduling. It facilitates the evaluation of policies for dynamic I/O resource scheduling across co-located heterogeneous applications. By using *PriDynSim*, Cloud researchers can evaluate I/O resource scheduling algorithms which consider the requirements of a wide spectrum of I/O applications to guarantee their performance Quality of Service (QoS). This could be done in a well known simulation environment, without using expensive testbeds. By incorporating the representation of latencies and deadlines of applications in the simulation environment, *PriDynSim* can facilitate future research efforts in the development of time-constrained scheduling policies which can be tested and validated using the proposed framework. More specifically, the *key contributions* of this paper are:

- A performance model for priority based I/O resource scheduling;
- Design and implementation of the *PriDynSim* simulator, which follows the above model;
- A case study demonstrating the functionality of *PriDynSim* for modeling of a priority based I/O policy.

The rest of the paper is organized as follows: Section II discusses the related work in this domain and section III contains detailed description of the design of proposed *PriDynSim* simulator in relation with the *CloudSim* architecture. Section IV explains the implementation details of *PriDynSim* with help of algorithms. In section V, we discuss a case study undertaken to validate the functionality of the proposed framework and show experimental results. Section VI concludes the paper with a brief discussion of the future directions.

II. Related Work

A number of simulation toolkits are available for Cloud environments out of which *CloudSim* is the most closely related to our work. However, *CloudSim* lacks support for representation of deadline driven I/O workloads and the priority based scheduling policies that are required for such workloads. Previously, many attempts have been made at extending *CloudSim* to add new functionality to it based on different requirements. For e.g., *WorkflowSim* [4] introduces support for workflow simulations and scheduling algorithms, *DynamicCloudSim* [5] extends *CloudSim* to handle heterogeneity of applications and dynamic changes to the performance, and *CloudReports* [6] provides a graphical user interface for simulating techniques for power optimization in Cloud computing environments. Others have proposed changes for specific use cases such as *SmartFed* [7] for the handling of federated Cloud environments with multiple providers and *CloudMIG* [8] which facilitates the migration of enterprise software systems to the Cloud. Li et. al. proposed an enhanced platform *DartCsim++* [9] that addresses limitations of *CloudSim* by supporting the simulation of network and power models at the same time. None of these works based on *CloudSim* have addressed the concerns about the performance modeling of concurrently executing latency-sensitive I/O workloads which is a very common scenario in typical real-life Cloud setups.

Comparing with other Cloud simulators, *GreenCloud* [10] and *MDCSim* [11] focus on energy consumption and power optimization for multi-tier Cloud data-centers without giving due consideration to application performance unlike our work. *ICanCloud* [12] is a simulation tool comparable in functionality and design to *CloudSim* which enables simulation on larger scale spanning multiple machines but it also lacks any support for I/O operations or priority scheduling policies. To the best of our knowledge, none of the available simulation platforms have attempted to design priority based I/O scheduling policies to address the performance concerns for I/O intensive workloads in a Cloud environment like *PriDynSim*.

III. PriDynSim Architecture

A. Proposed Extensions

CloudSim was previously extended to support I/O operations alongside CPU instructions when modeling jobs (assigned to cloudlets) [2]. The extension was made on three levels - Host, VM and Cloudlet, where at each level the base entity was extended to include representation for disks and I/O operations. In this work, we have proposed a new simulator which further extends the *CloudSim* framework mainly at the Cloudlet level to represent the job characteristics in terms of the deadlines of the jobs, their start times and the disk IOPS being assigned to them at a given point during the simulation. We have also designed a default intelligent resource scheduling policy in *PriDynSim* for the allocation of disk resources among the contending cloudlets running on the same VM by extending the current time-based scheduler. The *PriDynSim* scheduler ensures better performance as compared with the basic fair-sharing policy currently used by *CloudSim*. By taking cognizance of the deadlines associated with the cloudlets, our scheduler calculates the estimated latencies of completion and assigns suitable priority to the cloudlets with regard to the assignment of disk IOPS. The cloudlet being assigned the highest priority will be allocated the desired IOPS needed to complete its execution within the deadline value, provided the required IOPS is within the system constraints i.e. required IOPS is less than the maximum value of IOPS that can be assigned to any cloudlet on the VM.

To implement this, the main extensions to the existing framework are listed as follows:

1) *HddCloudletEx* : The Cloudlet having support for representing CPU and I/O operations has been further extended to include the following fields :
   a) *Deadline*: Represents the time by which the cloudlet is expected to finish completion.
   b) *Start Time*: The time at which the cloudlet is assigned the disk IOPS for executing I/O operations.
   c) *IOPS*: The value of the disk IOPS assigned to the cloudlet at a given time.

   The values of deadline and start time are input from the Cloud user. The default value of IOPS for a cloudlet is set to zero until the cloudlet is assigned IOPS as per system configuration.

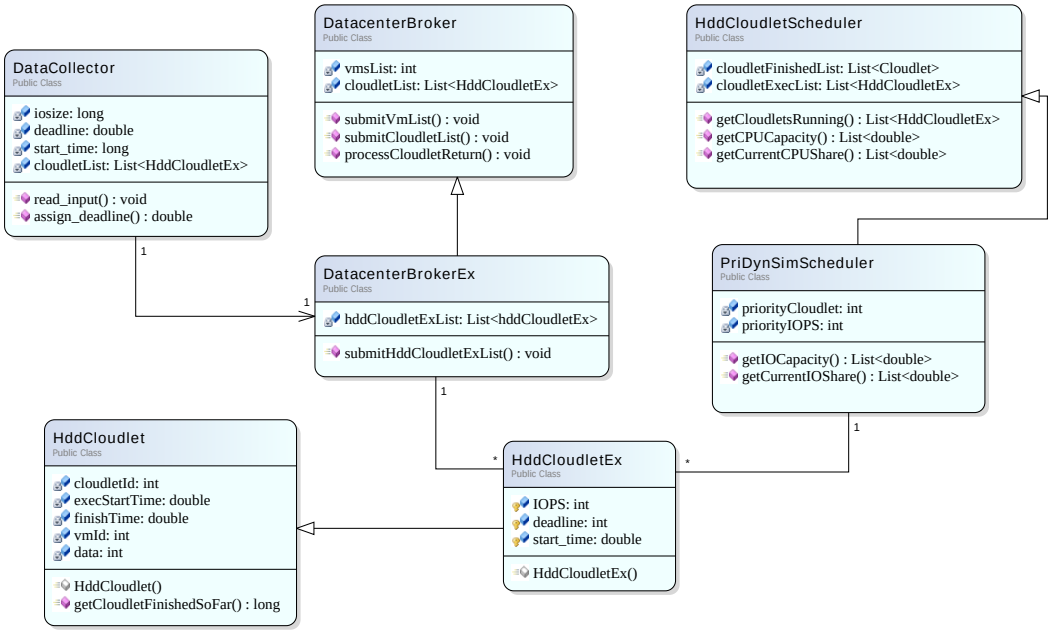

Figure 1: PriDynSim Class Diagram

2) *PriDynSim Scheduler*: The existing time shared scheduler *HddCloudletScheduler* that assigns the disk resources to the cloudlets in a fair and equal fashion has been replaced by a new scheduler which measures the requirements of the cloudlets as per their I/O operations and deadline values, and assigns IOPS to satisfy the deadlines of the workloads of the cloudlets.
3) *DatacenterBrokerEx*: This is an extension of the existing *DatacenterBroker* entity. A *broker* in *CloudSim* terminology acts on behalf of the user for creation and destruction of VMs and submission of cloudlets to the VMs. The extended *DatacenterBrokerEx* entity can handle the submission of *HddCloudletEx* entities that include representation for I/O operations and their deadlines.

To illustrate the relationship between the various classes of the *PriDynSim* framework, we take the help of a class diagram as shown in Figure 1. The base *HddCloudlet* class having support for I/O data representation is extended to *HddCloudletEx* as explained above. A separate *DataCollector* class is responsible for reading the input values of *iosize* and *start_time* for the cloudlets from the user. This class also assigns suitable deadlines to the requests as explained in section V-A. This information is passed on to the *DatacenterBrokerEx* class that is derived from the base class *DatacenterBroker* and can handle the submission of the list of newly defined *HddCloudletEx* instances to the simulation environment for execution. After submission, the allocation of disk IOPS to the cloudlets is performed by the *PriDynSim* scheduler class which in turn is derived from the base class *HddCloudletScheduler*. The class keeps track of the resources assigned to all the cloudlets and gives suitable priorities to the cloudlets based on their latency requirements to satisfy their deadlines.

We next discuss the interaction between the various entities of the *PriDynSim* framework for a simulation session with help of an interaction diagram.

B. Functionality

Figure 2 shows the various entities that are part of the *PriDynSim* framework and their interactions with each other. The *Session_Manager* entity represents a set of jobs or cloudlets that are read as input from the user to be executed on *CloudSim* in parallel. For every cloudlet in the session, the *Data_Collector* entity collects the information about the job such as the number of I/O operations, the deadline and the start time of the jobs and sends the list of cloudlets to the *DatacenterBrokerEx*. The *DatacenterBrokerEx* assigns the cloudlets to the VMs and then starts the execution of the jobs based on their start time as specified by the user.

The *CloudSim* framework is an event based simulator. Cloudlet life-cycle actualities like start, completion, and failure are represented as events in simulation time. Initially, an event occurs every time a new job is started and at every event the resources allocated to the cloudlets are assessed and reassigned based on the requirements. In the proposed framework, the *PriDynSim* scheduler consists of two entities that work in a collaborative manner, the *Latency_Estimator* and the *IOPS_Manager.* At every event, the *Latency_Estimator* calculates the estimated time of completion or the latencies of all the cloudlets according to the number of I/O operations and the IOPS assigned to the cloudlets. Comparing the latency values with the deadlines of the respective cloudlets, it finds out the cloudlets which are estimated to violate their deadlines. If there are multiple such cloudlets, then the one with the earliest deadline is chosen as the *Priority* Cloudlet which is given the highest disk priority so as to finish it before its deadline.

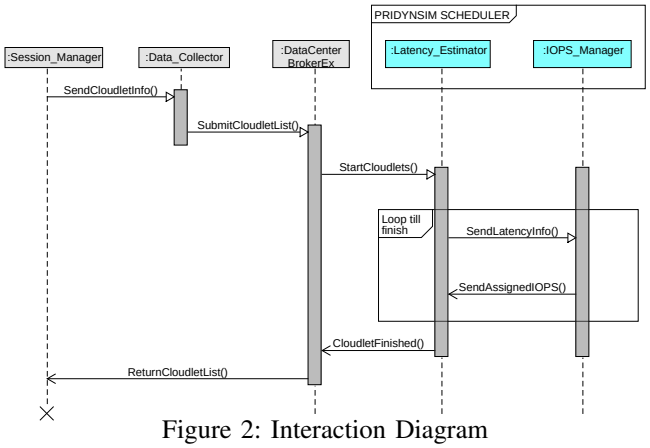

Figure 2: Interaction Diagram

This information is sent to the *IOPS_Manager* entity which performs the allocation of disk IOPS to all the cloudlets. It assigns the disk IOPS desired for completion of the *Priority* Cloudlet directly and the remaining cloudlets get an equal share of the remaining disk IOPS. The time of the next event is calculated as the time of completion of the *Priority* Cloudlet. Also, the information about the IOPS assigned to the cloudlets is sent back to the *Latency_Estimator* such that the calculations can be performed again at the next event. The number of I/O operations that are remaining for each cloudlet are recalculated based on the time elapsed and the previous allocation of IOPS. The entire process is repeated in loop at every I/O scheduling event until all the cloudlets have finished their execution.

In this manner, the *PriDynSim* scheduler dynamically enables differentiated disk IOPS for the cloudlets as per their deadlines. The Cloudlets having an approaching deadline are given higher priority and assigned greater share of disk IOPS compared to other Cloudlets having more lenient deadlines. In this way, such that the deadlines are not violated and timely completion is guaranteed for all the jobs. When all the cloudlets have completed execution, the *DatacenterBrokerEx* is informed about the completion of the cloudlet list and the information about the completion time of the cloudlets is sent back to the *Session_Manager* for displaying to the user. The same functionality is repeated in every simulation session.

IV. Performance Model and Algorithms

We now elaborate upon the implementation details of *PriDynSim* scheduling policy with the help of its algorithmic representation. The notations used in the algorithm have been summarized in Table I for easy reference. The number of cloudlets C running in parallel on a VM are assumed to be N whereas the total number of cloudlets submitted at the start of simulation are N_{Total}. The values of the total number I/O operations and the start times of the jobs running on these cloudlets, which are given by the user, are denoted by IO and ST respectively. Deadlines D dictate the time by which the cloudlet is required to complete execution. Latency L is the estimated time in which a cloudlet is expected to complete its job depending upon the assigned disk $IOPS$ and the number of IO operations remaining at a given point of time. The cloudlet chosen as the *Priority* Cloudlet is denoted as $C_{Priority}$ and it is assigned disk IOPS represented as $IOPS_{Priority}$. Using these notations, the working algorithm of the *PriDynSim* scheduler is explained in Algorithm 1.

The algorithm receives the set of N_{Total} cloudlets, their start times and the number of I/O operations as inputs and decides the IOPS to be assigned to the cloudlets to satisfy the deadlines of all the cloudlets. Initially, when the cloudlets are first submitted, the *Priority* cloudlet is not defined, and thus, the value of the IOPS assigned to the *Priority* cloudlet is set to zero. Firstly, the value IOPS assigned to each of the cloudlets is found out with the help of the *IOPS_Manager* module which has been separately explained in Algorithm 2. This module receives the total number of actively running cloudlets and the value of the $IOPS_{Priority}$. When the value of $IOPS_{Priority}$ is zero, for e.g., during the first iteration of the main algorithm, the total disk IOPS are simply assigned equally to all the cloudlets by diving by the N. However, when $IOPS_{Priority}$ has an non-zero value, this value is subtracted from the value of $IOPS_{max}$ and this is divided equally among all the cloudlets except for the *Priority* Cloudlet itself. The set of $IOPS$ values for the cloudlets is returned to the main algorithm. After having found out the initial values of $IOPS$ for all the cloudlets, the steps 3 to 24 are repeated in a loop until all the cloudlets have finished their execution. The latency for all the cloudlets is then calculated with the help of $IOPS$ previously estimated and the number of I/O operations for each cloudlet. In the first iteration $IO_{(i)}$ will be equal to the total number of IO operations to be executed for the cloudlets. These latency values are compared with the required deadlines for the cloudlets to find if any cloudlet is expected to violate its deadline. If there are multiple such cloudlets, the one with the earliest approaching deadline is chosen as the *Priority* Cloudlet. The value of $IOPS_{Priority}$ is assigned as the desired IOPS for the *Priority* Cloudlet to finish within the deadline shown in Step 14.

Before choosing the *Priority* Cloudlet, it is ensured that the value of IOPS that is desired by the cloudlet to finish its execution within the deadline is attainable as per the system constraints, that is, the desired IOPS is not greater then the value of $IOPS_{max}$. If such is the case, then this cloudlet is not assigned as the *Priority* Cloudlet and given proportional disk share since it is not possible to satisfy the performance requirements of the cloudlet on the given system setup. Such a cloudlet will necessarily miss its deadline and should be ideally migrated to a different system.

After finding the *Priority* Cloudlet, all the running cloudlets are assigned the disk IOPS with help of the *IOPS_Manager* module once again. For every cloudlet, if this cloudlet is the *Priority* Cloudlet then it is assigned the $IOPS_{Priority}$ and otherwise, its is assigned an IOPS value by equally sharing the disk IOPS among all the non-priority cloudlets as described in Algorithm 2. In the next iteration of the *while* loop, which will start at the next event, after the completion of the *Priority* Cloudlet, we will again consider the set of

Table I: Terminology for PriDynSim Algorithm

ATTRIBUTE	*NOTATION*	*DESCRIPTION*
Cloudlet	$C =< C_{(1)}, C_{(2)} \dots C_{(N)} >$	The cloudlets currently submitted to the simulator, N is the number of active cloudlets at a given time out of the total N_{Total} submitted in the list. .
Deadline	$D =< D_{(1)}, D_{(2)} \dots D_{(N)} >$	$D_{(i)}$ is the time by which the job assigned cloudlet $C_{(i)}$ is required to finish execution (measured in seconds).
Number of IO Operations	$IO =< IO_{(1)}, IO_{(2)} \dots IO_{(N)} >$	$IO_{(i)}$ is the number of IO operations remaining to be completed by job assigned to cloudlet $C_{(i)}$ at a given time. At the time of submission, $IO_{(i)}$ will represent the total number of IO operations for the cloudlet.
Start Time	$ST =< ST_{(1),}ST_{(2)} ... ST_{(N)} >$	$ST_{(i)}$ is the starting time for the job assigned to cloudlet $C_{(i)}$.
Disk IOPS	$IOPS =< IOPS_{(1)}, IOPS_{(2)} \dots IOPS_{(N)} >$	$IOPS_i$ is the current value of disk IOPS assigned to cloudlet $C_{(i)}$.
Latency	$L =< L_{(1)}, L_{(2)} \dots L_{(N)} >$	$L_{(i)}$ is the estimated time that cloudlet $C_{(i)}$ will require to finish its job based on IOPS allocated to it. (Measured in seconds).
Priority Cloudlet	$C_{Priority}$	$C_{Priority}$ denotes the cloudlet which has been assigned the highest priority such that it will be assigned the desired value of disk IOPS required for the completion of the job before its deadline.
Priority IOPS	$IOPS_{Priority}$	$IOPS_{Priority}$ is the value of IOPS that must be assigned to the Priority Cloudlet to satisfy the deadline.
Maximum IOPS	$IOPS_{max}$	$IOPS_{max}$ denotes the maximum value of IOPS that can be assigned to a cloudlet running on the VM. If a cloudlet requires IOPS greater than this value to meet its deadline, it should be migrated to different system.

active cloudlets (not including the *Priority* Cloudlet which has already finished execution). The latency values will be recalculated based on the previously assigned IOPS values and number of I/O operations remaining to be completed at that time after subtracting the number of I/O operations already completed in the last iteration. The algorithm will repeat again and choose another cloudlet to be given highest priority based on the value of deadlines. This process is repeated until all the cloudlets complete their execution and the value of active cloudlets N becomes 0.

The algorithm described in this section is the generic implementation and users can implement new policies by specifying different parameters or extending the default classes as per their specific requirements for prioritizing the applications. In the next section, we describe a case study to illustrate the performance of the current *PriDynSim* scheduling policy.

V. Case Study

In order to validate the functionality of *PriDynSim*, we performed a case study by simulation of real-life workloads. We used the I/O workload traces available at SNIA IOTTA repository [13]. These are block I/O traces from the servers at Microsoft Cambridge [14] which serve a variety of enterprise workloads such as web services, media applications, email etc and can be considered to be a good representative of typical Cloud workloads in data centers. The traces consist of files belonging to different applications having information about the I/O size and the start time of the requests of the application in sequence. For this case study, we have divided the traces into two main categories. In the first category, we choose the traces of applications that are latency sensitive such as media or web server, having strict deadlines for the completion times of their requests for satisfaction of performance requirements. For the second category, traces of delay tolerant applications such as long running research applications or logging activities were chosen. Such applications do not have strict latency requirements and can be assigned lenient deadlines. For our experiments, a suitable combination of requests from these two categories were chosen for simulation on *CloudSim* framework such that the setup closely resembled a typical Cloud setup in real world. Each cloudlet is assigned the requests being read from a trace file as jobs and multiple such cloudlets are executed in parallel. The values of the number of I/O operations and the start time of the job is read from the trace file and assigned to the cloudlets to be submitted to the broker. The traces do not contain the deadline values for the requests and therefore, we have used common techniques widely used in literature to design a deadline assignment scheme for the cloudlets. This scheme has been explained in our previous work [3] and is summarized here for the sake of clarity and completeness.

A. Deadline Assignment for Jobs

For assigning deadlines to the application requests of the trace files, we use techniques commonly used in previous literature [15]–[19]. In an ideal setup, every application running on a server is expected to get the entire disk IOPS and we denote the expected time taken by the *Makespan* of the request. This is the minimum time required to complete the request as per the system constraints and it is calculated by using the value of the maximum IOPS available for the cloudlet on the VM as follows:

$$Makespan = IO/(IOPS_{max}) \quad (1)$$

Algorithm 1 Priority Scheduler

Require: $N_{Total}, C, Total\ IO, D, ST$
Ensure: $IOPS$

```
Initialize IOPS_Priority as 0
for each C_(i) in < C_(1)...C_(N) > do
    Call IOPS_Manager(N, IOPS_Priority)
end for
while (N >= 0) do
    for each C_(i) in < C_(1)...C_(N) > do
        Calculate L_(i) = IOPS_(i)/IO_(i)
    end for
    if (exists C_(i) s. t. L_(i) > D_(i)) then
        Find C_(i) where D_(i) is minimum
        if ((IO_(i)/D_(i)) - ST_(i)) > IOPS_max then
            Continue to next C_(i)
        else
            C_Priority = C_(i)
            IOPS_Priority = IO_(i)/(D_(i) - ST_(i))
        end if
    end if
    for All C_(i) in < C_(1)...C_(N) > do
        if C_(i) = C_Priority then
            IOPS_(i) = IOPS_Priority
        else
            Call IOPS_Manager(N, IOPS_Priority)
        end if
    end for
end while
```

Algorithm 2 IOPS Manager

Require: $N,\ IOPS_{Priority}$
Ensure: $IOPS_{(i)}$

```
if (IOPS_Priority = 0) then
    IOPS_(i) = IOPS_max/N
else
    IOPS_(i) = (IOPS_max - IOPS_Priority)/(N - 1)
end if
return IOPS_(i)
```

But in a Cloud environment, disk IOPS get divided among multiple applications contending for the resources at the same time and therefore, the requests may take longer to finish execution than the *Makespan*. We introduce a parameter δ to model the delay tolerance property of an application based on its characteristics. Based on this parameter, the deadline is assigned to a request as:

$$Deadline = Makespan + (Makespan \times \delta) \quad (2)$$

This equation however gives us deadlines that are proportional to the I/O sizes of the requests whereas in a typical data-center such direct dependency may not be present and small I/O requests may sometimes take longer to complete due to system conditions. It is therefore appropriate to introduce a factor of randomness in the calculation of deadline values. This is done with the help of $rand$ function as follows:

$$Deadline = rand\,[Makespan,\ Makespan{+}(Makespan{\times}\delta)] \quad (3)$$

In this way, the deadlines for the requests are calculated and assigned based on the latency characteristics of the applications. For e.g. a latency sensitive application will be have smaller values of δ while applications which can tolerate delays in response time will be assigned higher values of δ.

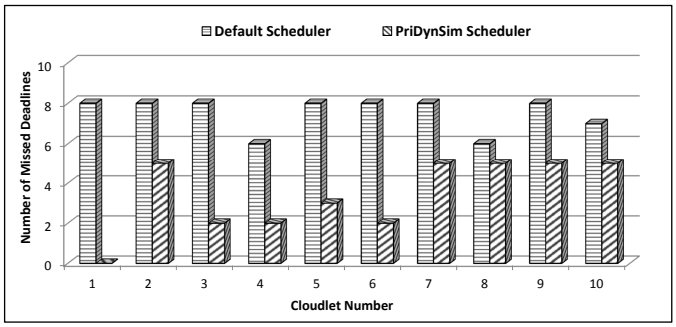

Figure 3: Comparison of Missed Deadlines for 10 Cloudlets

B. Results

In order to illustrate the functionality of *PriDynSim* scheduler, we compare its performance in terms of the number of cloudlets satisfying their deadlines as compared with the default fair-sharing scheduler currently included in the *CloudSim* environment. A series of experiments were performed with varying number and combinations of cloudlets assigned to applications having different latency characteristics. In the first set of experiments, a set of 20 cloudlets was simulated to run in parallel on *CloudSim* having equal number of latency sensitive and delay tolerant jobs modeled by the I/O requests belonging to Media server and Research server respectively. The deadlines were calculated for the cloudlets based on the scheme described before, strict deadline values were assigned to Media server requests since they are interactive and require short response times. However, the response times for the delay tolerant Research applications can be higher and therefore a high value of deadline was assigned to such jobs by using a large value of δ parameter. The simulation session was repeated 10 times separately for default and the *PriDynSim* scheduler and the total number of times where the response time was greater than the deadline was measured for the latency-sensitive cloudlets.

Figure 3 shows the comparison between the two cases, the cloudlets are numbered and represented by the *x-axis* while the *y-axis* denotes the number of times these cloudlets missed their deadlines out of the total of 10 sessions. It can be seen that *PriDynSim* achieved better results for all cloudlets as the number of times the deadline was missed is reduced considerably. This is because, the default scheduler assigns disk IOPS to all cloudlets on equal sharing basis without taking

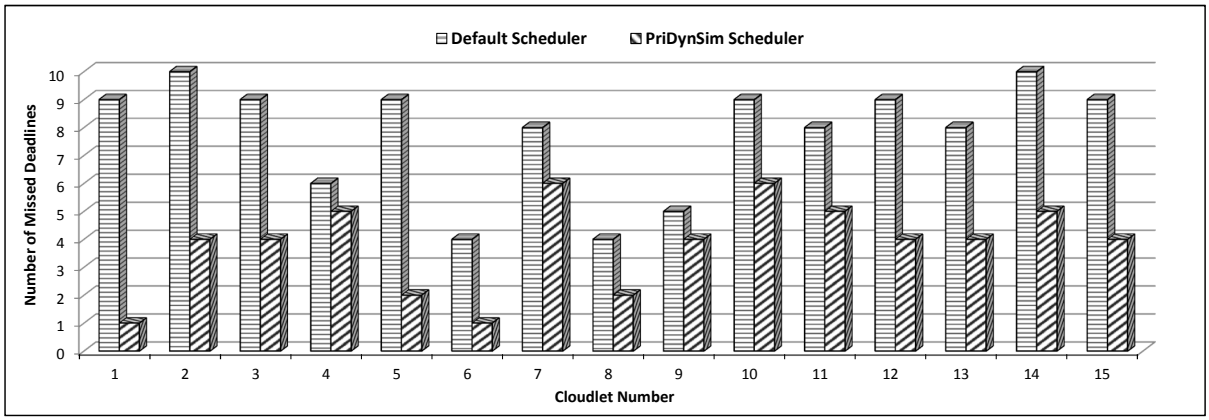

Figure 4: Comparison of Missed Deadlines for 15 Cloudlets

cognizance of their deadlines, and therefore, the cloudlets having strict deadlines are likely to miss their deadlines having received the same disk priority as that of other delay tolerant cloudlets. However, the *PriDynSim* scheduler is aware of the latency requirements of the cloudlets and provides a higher disk priority with desired value of IOPS to the cloudlets such that the deadlines are satisfied on the system. There are however, some cloudlets which will miss their deadlines even with *PriDynSim* scheduler, these cloudlets belong to the the worst case where it is not possible to satisfy the performance requirements due to system constraints, i.e. the required IOPS are greater than the maximum value of IOPS achievable for the system. Such cloudlets will necessarily miss their deadlines and therefore, they are given lower disk priority such that the performance of other cloudlets is not compromised at their expense. There were not more than one or two cloudlets belonging to this category for many of the sessions and such cloudlets had a very high value of response time as compared to their deadline. Therefore, apart from achieving the desired performance for the latency sensitive cloudlets, the *PriDynSim* scheduler also identifies the cloudlets that are a potential candidate for migration to other systems so as to ensure good overall system performance.

The simulations were also performed for higher number of cloudlets to explore the performance of the *PriDynSim* scheduler. In the next set of experiments, the number of cloudlets was increased to 30 with half of them being latency-sensitive and assigned strict deadlines. Figure 4 shows the comparison between the number of missed deadlines for the latency-sensitive cloudlets and that *PriDynSim* achieves better performance for all the cloudlets.

Further, in order to quantify the performance of *PriDynSim* scheduler, we measured the values of the deviations of the cloudlets from their deadlines, i.e. the difference between the values of response times (the time at which the cloudlet finished execution) and the value of deadline assigned to it. These $deviations$ were measured for both the default and the *PriDynSim* scheduler for only the cloudlets where the deadlines were missed, i.e., the $deviations$ are always either positive or assume zero value in the scenario where the cloudlet is able to finish earlier or at the deadline value. In Figure 5, we compare the values of the $deviation$ for the latency sensitive cloudlets for default and *PriDynSim* scheduler through a box plot over the set of ten sessions each. The plots are shown in sets of 2 corresponding to one cloudlet that are separated by dotted horizontal lines. Each set compares the $deviations$ for the cloudlet with the default fair-sharing scheduler and the *PriDynSim* scheduler (background shaded). The minimum value of $deviation$ is always zero for the case when the cloudlet satisfies the deadline whereas the maximum value denotes the highest $deviation$ from the deadline. The values of $deviation$ for the cloudlets identified as the worst case by the *PriDynSim* in any session is not taken into account since such cases are outliers for the experimental setup, the corresponding $deviations$ for the default scheduler were also omitted for ensuring correctness and fairness in the comparison. It can be observed from Figure 5 that *PriDynSim* scheduler achieves less $deviations$ for the cloudlets by prioritizing disk access as compared with the default scheduler which simply assigns equal IOPS to all. This demonstrates that for the cases where *PriDynSim* is unable to satisfy the deadline of the job, it ensures that the $deviation$ from the required deadline is the minimum such that the penalties for the delay in response times, if any, is minimized.

VI. Conclusions and Future Work

In this paper, we have proposed a novel *CloudSim* based simulator *PriDynSim* which can be used to explore policies for dynamic allocation of disk resources to I/O bound applications. The *PriDynSim* scheduler gives prioritized disk access to the jobs after identifying them as latency-sensitive. Experimental evaluation has shown the performance benefits of this policy through the achievement of guaranteed application performance for a wide variety of typical Cloud workloads modeled by real world I/O traces. *PriDynSim* is a generic simulator which can be used as a testbed by future researchers for the evaluation of their own specific scheduling policies. As future work, we plan to extend this policy to work at the data-center level where the allocation of cloudlets or job to a VM can be decided based on the application requirements such that the overall efficacy of the resource allocation can be optimized for Cloud data-center.

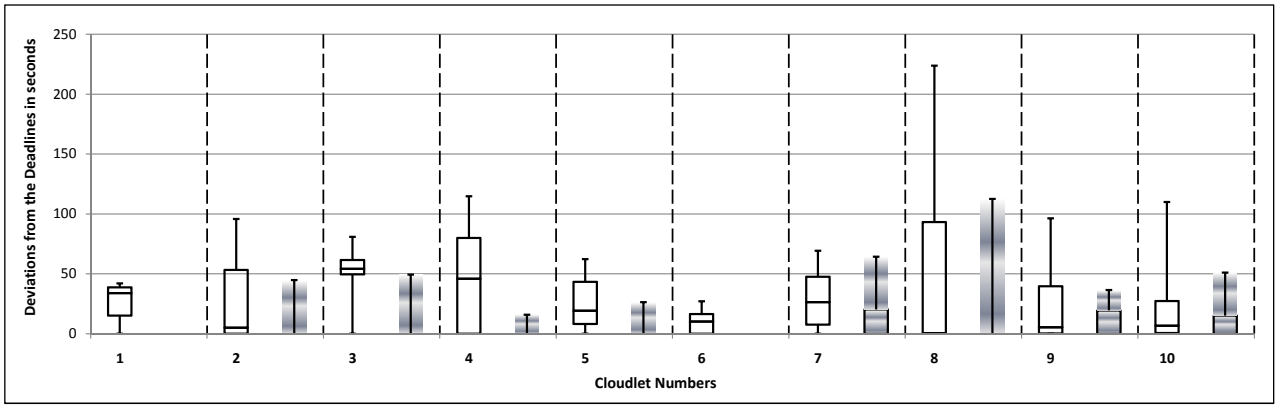

Figure 5: Comparison of Deviations from deadlines

ACKNOWLEDGEMENTS

We thank Rodrigo Calheiros, Amir Vahid Dastjerdi, Yaser Mansouri, and Chenhao Qu for their comments on improving this work. This work is partially supported by the Melbourne-Chindia Cloud Computing (MC3) Research Network.

REFERENCES

[1] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[2] N. Grozev and R. Buyya, "Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments," *The Computer Journal*, vol. 58, no. 1, pp. 1–22, 2015.

[3] N. Jain and J. Lakshmi, "PriDyn: Enabling Differentiated I/O Services in Cloud using Dynamic Priorities," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2014.

[4] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science 2012)*, pp. 1–8, IEEE, 2012.

[5] M. Bux and U. Leser, "DynamicCloudsim: Simulating heterogeneity in computational clouds," in *Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, p. 1, ACM, 2013.

[6] T. Teixeira SÃ¡, R. N. Calheiros, and D. G. Gomes, "CloudReports: An Extensible Simulation Tool for Energy-Aware Cloud Computing Environments," in *Cloud Computing* (Z. Mahmood, ed.), Computer Communications and Networks, pp. 127–142, Springer International Publishing, 2014.

[7] G. F. Anastasi, E. Carlini, and P. Dazzi, "Smart cloud federation simulations with cloudsim," in *Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds*, ORMaCloud '13, (New York, NY, USA), pp. 9–16, ACM, 2013.

[8] S. Frey and W. Hasselbring, "Model-based migration of legacy software systems to scalable and resource-efficient cloud-based applications: The cloudmig approach," in *Proceedings of the 1st International Conference on Cloud Computing, GRIDs, and Virtualization*, 2010.

[9] X. Li, X. Jiang, K. Ye, and P. Huang, "DartCSim+: Enhanced CloudSim with the Power and Network Models Integrated," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD 2013)*, pp. 644–651, June 2013.

[10] D. Kliazovich, P. Bouvry, and S. Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[11] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "MDCSim: A multi-tier data center simulation, platform," in *Proceedings of the IEEE International Conference on Cluster Computing and Workshops, 2009. (CLUSTER '09)*, pp. 1–9, Aug 2009.

[12] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator," *Journal of Grid Computing*, vol. 10, pp. 185–209, Mar. 2012.

[13] http://iotta.snia.org/. [Online], last accesed: 2014-12-15.

[14] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.

[15] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[16] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 22, IEEE Computer Society Press, 2012.

[17] R. N. Calheiros and R. Buyya, "Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds," in *Web Information Systems Engineering-WISE 2012*, pp. 171–184, Springer, 2012.

[18] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD 2010)*, pp. 228–235, IEEE, 2010.

[19] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1787–1796, 2014.