Performing accurate simulations for deadline-aware applications

Guthemberg Silvestre, Sébastien Monnet

Performing accurate simulations for deadline-aware applications

Guthemberg Silvestre and Sébastien Monnet
LIP6/UPMC/CNRS/INRIA
4 place Jussieu - 75005 Paris - France.
Email: {guthemberg.silvestre, sebastien.monnet}@lip6.fr

***Abstract*—One of the most appealing aspects of cloud computing is its potential capacity of offering customized services for clients through Service Level Agreement (SLA) contracts. Emerging cloud providers are intended to provide services whose performance targets are defined by precise data transfer deadlines. Yet, enforcing strict transfer rates for cloud-like applications is not simple. This has caught system designers and analysts' interest, and drawn their attention to study and evaluate the performance of deadline-aware applications. In order to address the increasing demand for performance evaluation tools for deadline-aware simulations, we present a bandwidth scheduling component for the PeerSim simulator. Our component provides a lightweight and accurate fair-sharing of bandwidth and rate control enforcement. We assume a simple approach that does not reproduce in-depth transport protocol behaviour. Instead, our design focuses in reducing the inherent impact on the overall simulation scalability as bandwidth scheduling accuracy is improved. We have studied the scalability and bandwidth scheduling accuracy of four scheduling approaches, namely the simplest, lock-based, packet or slot-based, and connection-oriented bandwidth scheduler. We have verified that a connection-oriented approach allows us to (i) improve considerably the accuracy of fair bandwidth sharing, and (ii) implement a rate control mechanism properly. Our source code is available online [3].**

***Keywords*—Datacenter, Quality of Service, Cloud Services, SLA, Rate Control, Peer-to-Peer.**

I. Introduction

Cloud computing has reshaped the way we use the Internet. Network and Content Delivery Network (CDN) providers have made huge efforts on development and infrastructure investments to offer cloud services with high scalability and outstanding performance guarantees to customers. For instance, Amazon Elastic Compute Cloud (Amazon EC2) [1] offers a virtual computing environment where users can easily configure and run online servers, paying only for resources that they are actually using. Similarly, cloud users expect that emerging Internet services could provide outstanding performance guarantees, e.g. enforcing deadlines for data transfers through SLA contracts. Unlike fair-share scheduling, where bandwidth is equally distributed among concurrent transfers, a deadline-aware approach, such as D3 [12], uses explicit rate control to apportion bandwidth according to the flow deadline to prevent SLA violations.

While cloud computing gives new opportunities for customers to use innovative services, it makes software architectures and system analysts facing new development challenges. In this context, performance analysis plays a very important role, and permit improving considerably the overall outcome quality. In this work, we are mostly interested in performance evaluations for cloud services through simulations.

According to Raj Jain [5], simulation is a useful technique for computer systems performance analysis which provides an easy way to predict the performance or compare several alternatives. In the recent years, network simulation tools have been continuously improved and used by system analysts for carrying out proper performance evaluation studies. Scalability, accuracy, and usability make part of the main issues when analysts are looking for a simulator tool.

In this work, we present a PeerSim component for bandwidth scheduler that permits simulating deadline-aware applications. PeerSim [6] is a highly scalable Peer-to-Peer (P2P) simulator. We have been particularly interested in PeerSim's modularity and user-friendly API. Our component allows system analysts to easily define and run simulations where accurate fair-sharing bandwidth and strict rate control enforcement are key issues. It is suitable for simulating cloud application that requires a deadline-aware control protocol on top of P2P networks. We have designed and implemented an accurate and lightweight mechanism for enforcing fair-sharing policy on data transfers and strict rate control, whenever it is required. It does not reproduce the complexity of in-depth transport protocol behaviour. Instead, we focus on proper dynamics of bandwidth sharing based on a connection-oriented approach. It allows us to apportion bandwidth resources on cloud environments. Like PeerSim, our component is simple to use and can be easily customised for many different set-ups.

In this work, our contributions are three-fold:

- We provide a user-friendly PeerSim component that allows system analysts to simulate accurately and lightly fair-sharing of bandwidth resources on Peer-to-Peer networks.
- Our component offers an implementation of the main functionalities of the newborn deadline-aware control protocols. It permits enforcing rate control to allocate bandwidth following strict application-level SLA contracts for cloud services simulations.
- We evaluate the performance of four simple bandwidth

scheduling mechanisms regarding their scalability and capacity of providing accurate bandwidth sharing on Peer-to-Peer networks.

For space constraints, in this work we focus on the design of our lightweight connection-oriented bandwidth scheduler component and on evaluating its transfer accuracy, flexibility, and overhead. More details about its functioning are available on the component repository [3]. Further evaluations and usage details of its functionalities in terms of deadline-aware services and transfer rate enforcement are available on previous works [8] and [9], where we study SLA-based resource allocation in edge networks. The remaining sections are organised as follows. Section II presents some related works. In Section III, we describe the design of our component in details by explaining its main functionalities and implemented protocol layers on PeerSim. Then we evaluate the performance of four different bandwidth scheduling approaches in Section IV. A comparative study of these approaches and some final discussions are presented in Section V. Finally, Section VI concludes.

II. Related Work

We have focused our efforts in studying works that help us to develop our component for Peer-to-Peer networks. We are mostly interested in evaluating scalability, accuracy, and usability of mechanisms and components for simulating deadline-aware cloud applications properly.

Weingartner *et al.* [11] evaluated the overall performance of five popular simulators. They concludes that ns-3 [2] outperforms its opponents, including OMNet++ [10]. One of the most advantages of using simulators such as ns-3 is that they provide a full set of realistic protocol models. However, their intrinsic set-up complexity undermines the simulator usability, particularly for inexperienced analysts, and its realistic models implementations might reduce significantly the scalability for cloud services, such as data transfer with rate control. According to a survey in Peer-to-Peer simulators by Dinh *et al.* [4], the use of simulators like ns-3 is actually very occasional, and most analysts opt to develop their own tool. However, these tools are not necessarily made publicly available, becoming single paper or study-simulators, preventing simulation results reproductions.

Following a different approach, PeerSim [6], a highly scalable P2P simulator written in Java, is much more straightforward, and offers a modular API and a number of built-in P2P protocols. It permits performing discrete-event simulations through two different engines, namely cycle-driven and event-driven simulation engines. Simulation engines basically differ in complexity and representativeness. Cycle-driven engine runs quicker, is easier to configure, but assumes simpler scenario definitions. Event-driven engine is much more flexible, permits implementing a larger number of protocol and algorithms, and gives fine-grained results. Since the main focus of PeerSim is scalability, its built-in modules lack of packet-level or flow-level data transfer simulation support by default, therefore limiting accuracy in bandwidth usage simulations. In order to overcome this issue, Russo *et al.* [7] have proposed a PeerSim slot-based Network protocol that simulates bandwidth usage based on priority sharing policy. While priority sharing policy enhances the level of details of bandwidth usage simulations, allowing fair chunk-level simulation without causing major damages to scalability, it does not provide any rate control mechanism, and it is not accurate enough for deadline-aware data transfer simulations.

Recently, researchers have extensively studied strict SLA enforcement in datacenters networks. D3 [12] is a remarkable example of this approach. It provides a deadline-aware control protocol that uses explicit rate control to apportion bandwidth according to flow deadline. That provides the ability to increase the aggregate throughput in datacenter environments compared to TCP. In this work, we are proposing an implementation of this kind of bandwidth scheduling as a lightweight PeerSim component.

III. Component Design

In this section we outline our approach for simulating deadline-aware applications. We present the design of our PeerSim component, detailing its protocol layers, main modules, and functionalities.

We have implemented a modular component for simulating deadline-aware cloud application on top of PeerSim simulator. Figure 1 shows the layers of our component and their interactions with PeerSim. Configurations are made through the main PeerSim API, ensuring usability.

Our component is composed by three layers: Network, Transport, and Application. Each layer provides an interface for the upper layer and allows developers to easily add additional functionalities. For simplicity, we have selected the PeerSim node, with identifier zero, for being a special node in our PeerSim component. It is called coordinator. It stores Global structures that are essential for consistency and state of the on-going simulation, such as addresses mapping and full connections' state.

A Monitoring module was implemented to provide periodical information about the state of nodes. It tracks data transfers information such as number of accomplished flows, instantaneous number of bits sent, bandwidth usage, active connections, and so on.

In this Section, a data communication between pair of nodes is named according to the layer level as connection, flow, and transfer for Network, Transport, and Application layer respectively. The following Subsections describe the main functionalities of each layer in details.

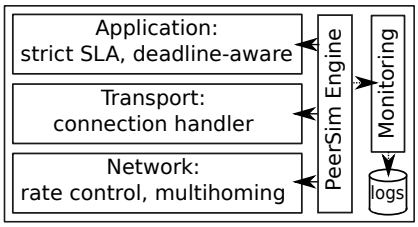

Figure 1. A modular PeerSim component for simulating deadline-aware applications.

A. Network Layer

Network layer implements a lightweight connection-oriented bandwidth scheduler and offers two essential bandwidth mechanisms for deadline-aware applications: fair-share and strict rate control.

Accurate fair-share scheduling of bandwidth permits apportioning network resources equally and dynamically throughout active connections. It simulates the common behaviour of communications between nodes without traffic priorities. Our evaluations focus on the accuracy of this functionality in this work.

Enforcing strict rate control is the main goal of our component. It aims to provide a precise rate control mechanism for enforcing strict SLA contracts. A deadline is attributed to each connection according to the upper layer requested rate. Deadline is the maximum amount of time, in milliseconds precision, for a connection ends. During the bandwidth allocation for connections, each node verifies the requested rate and reserve bandwidth properly. Towards accurate bandwidth allocation that captures the main aspects of network dynamics, we consider that correct fair-share of bandwidth is applied for non-reserved resources, as depicted in Figure 2.

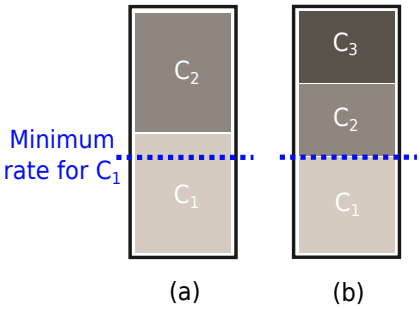

Figure 2. Rate control enforcement and fair-sharing scheduling dynamics.

Figure 2 (a) shows that a minimum rate of 40% of the bandwidth is enforced in connection C_1 that competes with connection C_2 for network resources. Since further resources are available, C_1 would be able to have more than required rate, finishing earlier than expected. In this particular case, fair-share scheduling ensures 50% of the bandwidth for C_2 . In Figure 2 (b), we consider, in the meanwhile, the arrival of a third connection C_3. C_2 and C_3 do not require a minimum rate. As result, Network layer enforces the minimum rate for C_1, what is equal to 40% in this example, and applies fair-share for the remaining resources, 60% of the bandwidth, throughout C_2 and C_3. It allows us to accurately reproduce network dynamics with rate control enforcement. Further insights into deadline-aware approach used in this work are available on Wilson *et al.* work [12].

We have also designed a multihoming scheme as part of the Network layer. Our goal is to allow users to easily define cloud-like scenarios, from machine-network virtualization to datacenter environments. Figure 3 shows the main blocks of our multihoming network model. We have defined Broadcast Domain (BD) as an essential block of our model. In a nutshell, it models isolated portions of PeerSim node's bandwidth. BD_0, or host domain, represents the whole bandwidth available on a node, which might simulate a datacenter uplink as well a simple host's network interface card. Then multihoming takes place by adding additional in-built BDs, with indexes greater or equal to one, so-called guest domains. Thanks to a maximum guest domain bandwidth limit enforcement, any bandwidth values might be freely and independently assigned to guest domains. Last but not least, this modules keeps and exports an address table, including BD to global pseudo-network address mapping, to easy usage and provide transparency of multihoming network functionality. Evaluations with deadline enforcement and multihoming functionalities have been omitted from this work due to space constraints, but are available on our previous works [8] and [9].

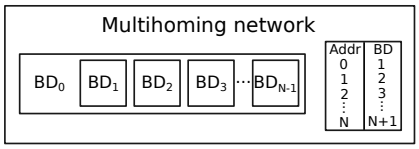

Figure 3. A multihoming network model for cloud-like simulations.

B. Transport Layer

We propose a Transport layer to interface with our deadline-aware Network layer. We assume that data transfers are handled between pair of nodes as connection-oriented data streams, called flows. It provides two main services for upper layers: connection management and accountability.

Flow management allows us to have full control of Network connections, including addresses' mapping for multihoming, creation, and deletion operations. Whenever a new node from upper layer is inserted in the simulation, Transport layer handles the mapping between Application and Network addresses. There is an single instance of this mapping structure per simulation in the coordinator node. It ensures mapping consistency and proper node selection in multihoming scenarios. Transport layer provides a easy way to create flows with a couple of parameters. It stores connections identifiers returned by the Network layers for accountability or deletions. A wide range of flow information and statistics are available, e.g. instantaneous

and precise number of bits already sent can be easily retrieved from Network.

C. Application Layer

This layer provides the main interface for running deadline-aware simulations. Many logical nodes may be hosted in a single Application layer in cloud-like scenario. So that, in a single PeerSim node, we are able to easily simulate multiple logical nodes, such as multiple virtual machines or even an entire datacenter. To each logical node is assigned an application-level address, which is mapped to network-level address through Transport layer mapping. This allows users to define and to operate multihoming properly.

Along with few definitions in the main PeerSim configuration file, Application layer requires a CSV input file with SLA contracts' definitions. Our current design allows us to define and use multiple SLA contracts. A SLA contract defines a strict transfer rate for a class of nodes. Then SLA contracts should be assigned to nodes accordingly.

Its functioning is straightforward. Application events or logical nodes' transfer requests are sent to Transport layer that interacts with Network for performing a transfer. Its behaviour depends on how Network layer is operating. Considering a Network layer applying just fair-share of bandwidth, an event is always treated, and when the transfer finishes, Transport layer notifies Application. Then Application checks if the transfer meets its respective SLA contract on behalf of the requester. On the other hand, if Network is configured to enforce rate control from SLA contracts, before starting the transfer, an admission control process takes place for verifying if there is enough network resources in both source and destination Networks for fulfilling the Application request. If there is no enough spare resources, Network raises a connections failure message, and requester's Application layer is notified.

IV. Simulations Results

In this Section, we aim to evaluate the performance and measure the accuracy of different bandwidth scheduling approaches in order to provide deadline-aware data transfer with strict rate control. We have studied four scheduling approaches for bandwidth allocation on top of PeerSim simulator: the simplest bandwidth scheduler, a lock-based bandwidth scheduler, a packet-based bandwidth scheduler, and a connection-oriented bandwidth scheduler.

We have defined an evaluation scenario with 1000 nodes. For simplicity, we assume a fully meshed and connected network topology, in which each node is equipped with a full-duplex 10Mbps link. We simulate data transfers in a content distribution network where each node plays a distinct role of either content provider or consumer. Content consumers randomly select a source per request for downloading. We consider that content size follows a bounded Pareto distribution that ranges from 1MB to 1GB. We simulate one hour of content transfer. Our primary factors for performance analysis are:

- Number of content sources: that is the number of nodes that play content provider role, whose uplink bandwidth is shared by consumers' downloads for content distribution.
- Content mean size: by changing the shape parameter of Pareto distribution, we have been able to evaluate bandwidth allocation performance with different content average sizes.
- Degree of parallelism: by degree of parallelism we mean the number of simultaneous active downloads performed by a content consumer node. During the simulations bootstrap, a number of parallel downloaded is launched from each consumer at the same time, according to the degree of parallelism . When a download is accomplished, a new download is immediately performed in order to keep the degree of parallelism.

The evaluation of our simulations towards deadline-aware transfers have been measured by three simple metrics: average uplink bandwidth usage of nodes playing content provider role, average memory usage, and average computation time per content transfer. We have performed our simulations using server with an Intel Xeon E5450 3.00 GHz, and a RAM of 4GB. We describe and evaluate each approach in the following Subsections.

A. The Simplest Bandwidth Scheduler

Here, we describe an easy way to implement a bandwidth scheduler for data transfer simulations. In order to transfer data between two nodes, the source node computes the bandwidth available to the destination by simply selecting the smallest bandwidth value between the two nodes, and then computing the duration of the data transfer based on the requested content size. This scheduler provides a static bandwidth allocation between the source and destination, that extremely speeds up simulation.

Assuming an evaluation scenario where each content consumer does not perform parallel downloads, that means degree of parallelism equals to one, and a content mean size of 8 MB, we have varied the number of content sources or providers from 10 to 50 % of the total of nodes. Figure 4 shows the average uplink bandwidth usage per content provider.

For this scenario, our implementation requires about 9.2 MB of memory on average, and the average computation time is only 0.4 microseconds per message sent. Although the simplest scheduler consumes very few computational of resources, it provides highly imprecise bandwidth usage results as the number of sources decreases, and concurrent content accesses occur. As expected, the maximum average is reached when 50% of nodes play the role of content provider. This evaluation metric soars to 90Mbps when 10% of nodes are content providers because there is no bandwidth sharing

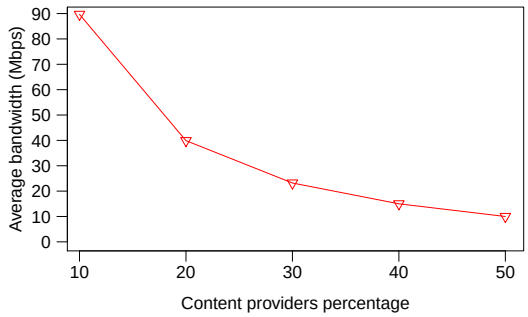

Figure 4. Average uplink bandwidth usage of content providers for different number of content sources.

policy for concurrent transfers on sources, generating highly inconsistent results.

B. A Lock-based Bandwidth Scheduler

We enhanced the previous bandwidth scheduler in order to improve bandwidth allocation precision without much increasing in computational resources consumption. We implemented a lock-based mechanism that prevents overlap of bandwidth consumptions when there are more consumers than sources. Its behaviour is quite similar to the previous approach. But, instead of starting a new transfer whenever it is requested, the content provider verifies the availability of the whole bandwidth in both source and destination. If at least one of them has already started a new data transfer, the request is queued on the content source, and FIFO policy is enforced. This simple improvement permits to avoid over-consuming in bandwidth usage of nodes. However, it undermines bandwidth allocation efficiency. Considering the worst case of the previous approach, where 10% of nodes are content providers, we show in Figure 5 what happens with average bandwidth usage on content providers when content mean size changes. Average bandwidth usage falls sharply from 8.5 to 5.1 Mbps when mean content size is multiplied by 4. That causes an increasing amount of idle bandwidth resources despite the higher load, simulating inaccurate bandwidth apportion.

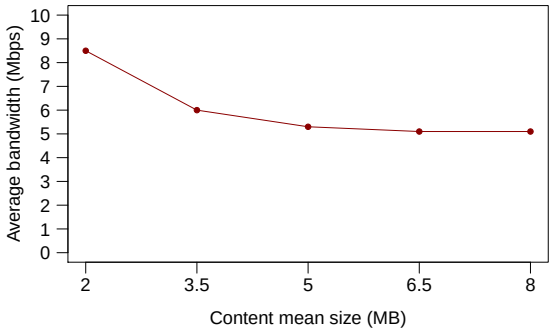

Figure 5. Impact of content mean size on average uplink bandwidth usage of content providers.

The performance of bandwidth allocation for this approach is optimal when data transfers have the same length, e.g. considering a uniform distribution for content size. Whatever the workload, parallel transfers can not be simulated properly.

C. A Packet-based Bandwidth Scheduler

This approach was based on the PeerSim bandwidth manager module proposed by Russo *et al.* [7]. In order to simulate parallel transfers and bandwidth sharing with low computational resources consumption, they introduced a bandwidth scheduler based on slots and priority sharing policy. They assume whenever a source node connect to a destination to transfer data, it allocates a bandwidth's slot for an amount of time, depending on the data size. They consider that uplink and downlink are asymmetric, with downlink greater than uplinks. In general, it permits that multiple uplink slots match into a single downlink, providing transfer parallelism with high performance. Although it is highly configurable, and simulates fairly bandwidth sharing for heterogeneous networks transmitting blocks of equal size, it is hard to be successfully reused in generic scenarios, particularly when accurate fair-share scheduling is required.

We have implemented a simpler version of this approach, called a packet-based bandwidth scheduler, that is easier to configure, and enhances significantly the bandwidth scheduling mechanism. In our implementation, we promote the chunk size as a key parameter. Slots duration does not depend of bandwidth match any more. Instead, we assume that a slot is a portion of bandwidth according to the chunks size definition. For example, if a uplink bandwidth is equal to 10Mbps, and the chunks size is 1MB, the number of slots is defined by dividing bandwidth by chunks size, in this case, it would be about 10. We have also added a queue for untreated or on-going requests that permits improving significantly the scheduling of content transfer with multiple chunks. When a request does not have enough available slots on both source and destination for transmitting its all chunks, the remaining chunks are put into the queue. Remaining chunks are served following FIFO queueing and according to the slots availability. Considering a degree of parallelism equal to four and content mean size of 3MB, we have evaluated the content delivery performance by computing the average uplink bandwidth usage on content providers, and scalability through measuring the computation time per transfer of different chunk sizes. Figure 6 shows that we are able to improve significantly bandwidth sharing accuracy by reducing the chunk size. In this case, when the chunk size is reduced from 8MB to 500KB, the average uplink usage increases from 2.5Mbps to 8.9Mbps. Yet improved accuracy pays its price, as depicted in Figure 7. The computation time per transfer for the same range of chunk sizes is increased from 3.1 to 5.3 microseconds.

Although this approach improves the dynamics of bandwidth sharing resources, it introduces a crucial trade-off between accuracy and scalability. The smaller is the chunk size, the better is the accuracy it provides, but with an increasing computational resources consumption. Whatever the chunks

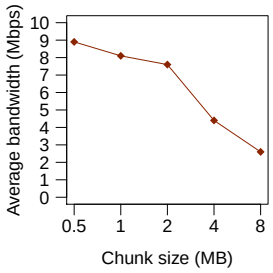

Figure 6. Average uplink bandwidth usage on content providers with different chunk sizes.

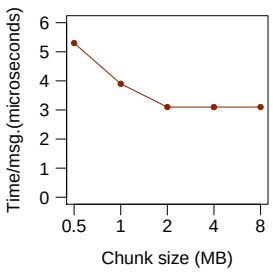

Figure 7. Average computational time per transfer with different chunk sizes.

size and the related computational cost, the use of packet-based, or slot-based, approach causes bandwidth allocation imprecision for incoming requests, that must wait the next free slot, or packet reading cycle, before starting. This is particularly damaging for small transfer lengths and strict rate control enforcement.

D. A Connection-oriented Bandwidth Scheduler

To overcome packet-based scheduler issues, improve accuracy in fair-share apportion, and simulate proper rate control for deadline-aware applications, we have designed and implemented a connection-oriented bandwidth scheduler in our Network layer for PeerSim, that was also initially based on the bandwidth manager module proposed by Russo *et al.* [7]. In fact, we are not interested in reproducing a wide range of realistic networking aspects, as data retransmission, packet-loss, or jitter. Instead, we focus on implementing a lightweight and accurate bandwidth scheduler that simulates the overall dynamics of bandwidth allocation on end-nodes. Therefore, we put great emphasis on enforcing fair-share scheduling for Peer-to-Peer networks. In our model, connection objects keep only the more precious information about the transfer, such as source-destination addresses, data to be transmitted, current allocated bandwidth, and remaining time. It permits computing bandwidth allocation and reproducing parallel and concurrent data transfer properly. We have run and compared the accuracy of our connection-oriented approach and a packet-based one, as depicted in Figure 8. For a chunk size of packet-based approach equals to 500KB, our connection-oriented bandwidth scheduler performs roughly 10% better.

Since fair sharing policy is enforced properly, we have been able to implement an accurate rate control mechanism based on D3 [12], discussed in Subsection III-A, that permits simulating data transfer for deadline-aware applications. Despite of improving consistently the bandwidth allocation accuracy, unsurprisingly it requires more computational resources than other approaches evaluated in this work.

V. Performance Analysis Summary

To provide a comparative study among the evaluated bandwidth schedulers, we have measured the performance in terms

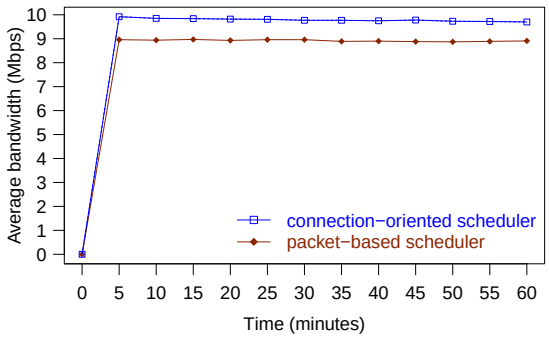

Figure 8. Average bandwidth usage on content providers.

TABLE I
Outline of scalability and accuracy results of four bandwidth scheduling approaches.

Approach	**Accuracy**	**Parallelism**	**Computation time per transfer (μs)**	**Average memory usage (MB)**
Simplest	Lowest	Not allowed	0.5	8
Lock-based	Limited	Not allowed	2.2	162
Packet-based	Fair	Allowed	3.8	177
Connection-oriented	Highest	Allowed	46.8	431

of scalability and accuracy over a common simulation set-up. In this common scenario, we assume the default configurations of Section IV, we set the degree of parallelism to one, number of content providers to 100, or 10% of the total of nodes, and the content mean size to 3MB. For packet-based approach, we chose a chunk size of 500KB, half of minimum content size. Table I provides an outline of evaluation results for all bandwidth schedulers. We highlight our main finds in the remaining part of this Section.

The simplest bandwidth scheduler: While the simplest approach is the easiest to implement, and highly scalable with a staggering computation time of only 0.5 μs per accomplished transfer and average memory usage of only 8MB, it performs the worst bandwidth sharing precision, and does not permit simulating parallel transfers. This approach might be useful for huge Peer-to-Peer networks that does not require fine-grained bandwidth tracking.

A lock-based bandwidth scheduler: Compared to the simplest approach, it performs a much better bandwidth allocation precision with excellent scalability in simulation duration, a still tiny computation cost per transfer of 2.2 μs. But the fact of implementing a FIFO queueing for scheduling incoming transfer requests causes a 20-fold increase in the average memory usage. Neither it offers parallelism, for us, an essential data transfer functionality. Lock-based approaches are rather suitable for scenarios with large number of nodes where data transmission in pipeline is acceptable.

A packet-based bandwidth scheduler: A packet-based bandwidth scheduler provides a quite fair scalability. Compared to a lock-based scheduler, it performs a minimal rise in average memory usage, an additional memory usage of 14MB on average, and a relatively high, but still impressive enough, increase of 70% in the computation time per transfer. Allowing parallelism in data transfers, it offers a good bandwidth scheduling mechanism for a wide range of applications. However, it exposes analysts to a trade-off between scalability and bandwidth scheduling precision through the choice of the chunks size. Assuming bandwidth allocation precision as a key feature for deadline-aware applications, this approach does not fit for purpose.

A connection-oriented bandwidth scheduler: Accuracy in simulating fair-share scheduling of bandwidth is at the core of our implementation. It performs precise bandwidth sharing thanks to connections-oriented approach. Unlike simulators that implement in-depth transport protocol behaviours, we minimize the computational cost improving scalability by maintaining only the most essential connections information. Although it causes a nearly 11-fold increase in computation time per transfer and a memory usage two and half times higher both compared to packet-based approach, connection-oriented bandwidth scheduler provides proper fair-share of bandwidth, what allowed us to successfully implement a strict rate control for simulating deadline-aware cloud applications.

VI. Conclusions

We have designed and implemented a PeerSim component that provides accurate bandwidth sharing and rate control properly. Our component is particularly useful for predicting the performance of deadline-aware cloud services. It allows analysts to easily set-up and customized simulations thanks to the PeerSim modular and user-friendly API. We have evaluated the performance our bandwidth scheduling approach and compared it to three simpler and more scalable bandwidth approaches. As we have shown, simulation accuracy always pays its price. Our approach is not the fastest, neither reproduces in-depth transport protocol behaviours. We have rather chosen to implement a straightforward, lightweight, and accurate enough approach based on a connection-oriented bandwidth scheduler that fits deadline-aware cloud application requirements properly.

Acknowledgments

We are grateful to Véronique Simon and Alessandro Russo for their valuable help at the early stages of this work.

References

[1] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/, 2012.

[2] ns-3: a discrete-event network simulator for internet systems. http://www.nsnam.org/, 2012.

[3] Arenbm: a simple bandwidth scheduler component for peersim. https://github.com/guthemberg/arenbm.git, 2013.

[4] Tien Tuan Anh Dinh, M. Lees, G. Theodoropoulos, and R. Minson. Large scale distributed simulation of p2p networks. In *Proc. 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing PDP 2008*, pages 499–507, 2008.

[5] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,*. May 1991.

[6] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, September 2009.

[7] A. Russo and R. Lo Cigno. Delay-aware push/pull protocols for live video streaming in p2p systems. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –5, May 2010.

[8] G. Silvestre, S. Monnet, R. Krishnaswamy, and P. Sens. Aren: a popularity aware replication scheme for cloud storage. In *ICPADS*, 2012.

[9] Guthemberg Silvestre, Sébastien Monnet, Ruby Krishnaswamy, and Pierre Sens. Caju: a content distribution system for edge networks. In *Proceedings of the 1st Workshop on Big Data Management in Clouds (BDMC '12)*, August 2012.

[10] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[11] Elias Weingartner, Hendrik vom Lehn, and Klaus Wehrle. A performance comparison of recent network simulators. In *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009)*, Dresden, Germany, 2009. IEEE.

[12] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, pages 50–61, August 2011.