Solving the TCP-incast Problem with Application-Level Scheduling

Maxim Podlesny
D.R. Cheriton School of Computer Science
University of Waterloo, 200 University Avenue West
Waterloo, ON N2L 3G1, Canada
mpodlesn@cs.uwaterloo.ca

Carey Williamson
Department of Computer Science
University of Calgary, 2500 University Dr. NW
Calgary, AB T2N 1N4, Canada
carey@cpsc.ucalgary.ca

***Abstract*—Data center networks are characterized by high link speeds, low propagation delays, small switch buffers, and temporally clustered arrivals of many concurrent TCP flows fulfilling data transfer requests. However, the combination of these features can lead to transient buffer overflow and bursty packet losses, which in turn lead to TCP retransmission timeouts that degrade the performance of short-lived flows. This so-called *TCP-incast problem* can cause TCP throughput collapse. In this paper, we explore an application-level approach for solving this problem. The key idea of our solution is to coordinate the scheduling of short-lived TCP flows so that no data loss happens. We develop a mathematical model of lossless data transmission, and estimate the maximum goodput achievable in data center networks. The results indicate non-monotonic goodput that is highly sensitive to specific parameter configurations in the data center network. We validate our model using *ns-2* [1] network simulations, which show good correspondence with the theoretical results.**

I. Introduction

Data centers have become very popular for storing large volumes of data. In particular, companies like Amazon, Google, and Yahoo! routinely use data centers for storage, Web search, and large-scale computations. The main characteristics of a data center network are high-speed links, low propagation delays, and limited-size switch buffers. In addition, the data for a given client application are usually striped (spread) over many servers, for increased performance (i.e., parallelism). Recent research efforts have resulted in several architectures of data centers [2]–[4].

The Transmission Control Protocol (TCP) is used as the transport-layer protocol for reliable data transfer in data center networks, just like it is on the Internet. However, the network configurations in data centers are very different from the general Internet conditions, for which TCP was originally designed. In particular, the typical propagation roundtrip delay in a data center network is 0.1 ms, while the default retransmission timeout (RTO) on the Internet is 200 ms. On each user request for data, many servers transmit data over a data center network concurrently, which, in combination with the small switch buffers, leads to packet losses. For short-lived flows, packet losses cause TCP retransmission timeouts, which in turn degrade the goodput of data center applications. Such a decrease of goodput is called TCP-incast throughput collapse.

One approach for solving the problem involves fine-grain timer resolution for the TCP RTO_{min} [5], or the use of different TCP variants [6]. These approaches require changes at the transport level, or within the operating system (OS) kernel. The other potential solution is to rely on switch-based mechanisms within the network [7].

In this paper, we explore the use of application-level flow scheduling to solve the TCP-incast problem in data center networks. The research question we study is how to schedule responses to client requests to avoid data losses at a bottleneck link. In particular, we model application-level scheduling, which does not require any changes in the TCP stack or the network switches. The main result we derive is the achievable goodput of an application in a data center under lossless scheduling. To verify our theoretical results, we perform extensive simulations. The simulation results confirm the main theoretical results of our model. To the best of our knowledge, our work is the first[1] to propose a detailed application-level mechanism and evaluate its performance for avoiding the TCP-incast problem in data center networks.

The rest of the paper is structured as follows. In Section II, we describe the model and the proposed approach. Section III reports the theoretical analysis of the proposed approach. Section IV discusses the practical use of our solution for data center applications. In Section V, we present the simulation results for validating our model. Section VI surveys related work. Finally, Section VII concludes the paper.

II. Model and Proposed Solution

Figure 1 shows a simple model of a data center network. A client connects to the data center via a switch, which in turn is connected to many servers. The client requests data from one or more servers, and the data are transferred (left to right) from the servers to the client, via the switch. The bottleneck link is the link from the switch to the client.

The client requests data using a large logical block size. In particular, 1 MB is a common read size in distributed file systems, such as Google File System (GFS) [9] and Panasas

[1]An earlier version of this work appeared in poster form in IEEE IWQoS 2011 [8]. Our current paper provides further details on the analytical model, practical deployment issues, and additional simulation results.

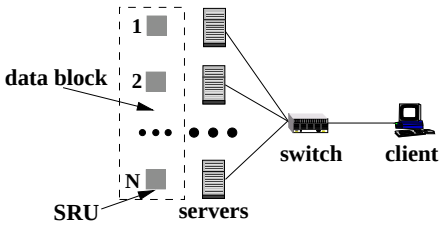

Fig. 1. Cluster-based storage system.

Parallel File System (PanFS) [10]. The actual data blocks are striped over many servers using a much smaller block size (e.g., 32 KB) called the Server Request Unit (SRU). A client issuing a request for a data block sends a request packet in parallel to each server that stores data for the requested data block. The request, which is served through TCP, is completed only after all SRUs from the requested data block have been successfully received by the client.

Because of the many-to-one fan-in from servers to the client, the data transfer workload can overflow the buffer at the bottleneck link, leading to packet losses, and subsequent TCP retransmissions. The typical SRU size is several kilobytes [11], though an SRU can be up to 256 KB in size [6]. Since the flow size for an SRU is small, the TCP congestion window is also small, which means that packet losses tend to cause coarse-grain TCP timeouts. Since the default value of RTO_{min} is 200 ms [12], and a propagation RTT in a typical data center is 0.1 ms [6], significant degradation of application goodput [6], called TCP-incast throughput collapse, can occur.

In our model, a client requests a data block of size S (in bytes), which is striped over N servers. The bottleneck link has capacity C (in bytes per second, Bps) and buffer size B (in bytes). The propagation delay between each server and the client is R. The size of data in each data packet, the size of an ACK packet, and the size of a data frame are S_{DATA}, S_{ACK}, and S_f, respectively. The TCP timer granularity of the client OS is Δt (in seconds). The initial TCP congestion window is one packet. We assume per-packet ACKs are used for TCP flows.

We consider an application-level solution that does not require any changes to the TCP stack or network switches. Since the main reason for TCP throughput collapse is data losses inducing a retransmission timeout, we explore how to schedule server responses to the same data block request so that there is no data loss at links (i.e., how to schedule requests for SRUs from the same data block without causing buffer overflow at the bottleneck link).

We find that the maximum goodput g of an application in a data center with lossless scheduling is:

$$g = \frac{S}{\widetilde{T}(\lceil \frac{N}{n} \rceil - 1) + T + d_{max}} \tag{1}$$

where

$$\widetilde{T} = \left\lceil \frac{T + d_{max}}{\Delta t} \right\rceil \Delta t \tag{2}$$

$$T = \frac{(S_f + S_{ACK})S_{SRU}}{C} + \lceil log_2(S_{SRU} + 1) \rceil \left(R + \frac{B}{C} \right) \tag{3}$$

$$n = \left\lfloor \frac{B}{S_f wnd_{max}} \right\rfloor \tag{4}$$

S_{SRU} is the size of an SRU in packets, wnd_{max} is the maximum number of packets of one flow in the network, and d is a random timer scheduling delay that we add to our model to account for real system scheduling variance. Similar to [5], we assume that d can be up to d_{max} = 20 μs. In the next section, we provide a step-by-step derivation of these expressions using our theoretical model.

III. Analysis

In our approach, each flow containing an SRU (i.e., a server response to a data block request), should avoid any packet loss. Due to the small size of an SRU, we assume that each SRU flow remains in TCP slow start throughout. Since the slow start behavior is the same in most versions of TCP, our model applies broadly to Reno, NewReno, etc. The size of an SRU, expressed in data packets, is:

$$S_{SRU} = \left\lceil \frac{S}{nS_{DATA}} \right\rceil \tag{5}$$

Let us derive the maximum window size wnd_{max} of an SRU under the assumption that an SRU is in slow start. In other words, we calculate the maximum possible number of outstanding packets in the network for a single flow. In slow start, the initial congestion window is one, and it is increased by one for each acknowledgement packet (ACK) received. Thus, the congestion window is doubled each RTT.

A. Motivating Example

In traditional TCP models, the data transfer occurs in rounds, based on the RTT. Furthermore, flows are treated in isolation, in that the data packets of one flow are not intermingled with data packets from other flows. This means that the size of a burst of TCP packets arriving at the bottleneck link follows the pattern:

$$\{1, 2, 4, 8, ..., 2^i, ...\}. \tag{6}$$

We illustrate the traditional model by the following example. Let us consider an SRU of size S_{SRU} = 9 packets. Initially, before any packets are sent, the congestion window $cwnd$ = 1, the number of packets in the network is wnd = 0, and the number of packets remaining to be sent is r = 9. At some time t_1, one packet is sent by the server; $cwnd$ remains 1, wnd becomes 1, and $r = 8$. Then, after the first transmitted packet is acknowledged, $cwnd$ becomes 2, which allows the server to transmit two more packets at some time t_2 (i.e., wnd = 2, and r = 6). At some later time t_3, the congestion window grows to $cwnd$ = 4, with wnd = 4, and r = 2. Then, at some time t_4, ACKs for all four packets successfully arrive at the server; wnd becomes 0, and $cwnd$ is increased by 4 and becomes 8. Since $cwnd > wnd$, the server can transmit the two remaining packets at some time t_5, and all of the remaining packets of the SRU are sent by the server. The step-by-step evolution of $cwnd$, wnd, and r is summarized in Table I.

In our model, we consider a general case of TCP behavior when the size of a burst of TCP packets arriving at the

TABLE I
TRADITIONAL EXAMPLE

Time	0	t_1	t_2	t_3	t_4	t_5
$cwnd$	1	1	2	4	8	8
wnd	0	1	2	4	0	2
r	9	8	6	2	2	0

bottleneck link is not constrained to the pattern described by (6), and data packets of one flow may be intermingled with data packets from other flows, although the congestion window of a flow is still doubled each RTT according to the TCP slow start algorithm. The behavior of TCP flows is reconsidered, because the propagation RTT in a data center network is 0.1 ms compared to the typical propagation RTT on the Internet of 10-100 ms. Assuming a packet size of 1 KB and a link capacity of 1 Gbps, the duration of sending one packet is around 0.01 ms. Thus, whereas packet transmission time is negligible in comparison with propagation RTT on the Internet, it is not the case in a data center network.

To demonstrate the difference between our model and the traditional one, we show how the example from Table I is changed if the size of a burst of TCP packets arriving at the bottleneck link is not constrained to the pattern described by (6). In particular, the evolution of $cwnd$, wnd, and r is the same in both models, until time t_4, when an ACK for one packet successfully arrives at the server; wnd becomes 3, and $cwnd$ is increased by 1 and becomes 5. Since $cwnd > wnd$, the server can transmit the two remaining packets at some time t_5. Although $cwnd$ remains 5, wnd becomes 5, and all of the remaining packets of the SRU are sent by the server. We present the step-by-step evolution of $cwnd$, wnd, and r in Table II.

TABLE II
ILLUSTRATING EXAMPLE

Time	0	t_1	t_2	t_3	t_4	t_5
$cwnd$	1	1	2	4	5	5
wnd	0	1	2	4	3	5
r	9	8	6	2	2	0

The key difference between these two examples is that whereas in the traditional model at time t_4 the whole window of four packets was acknowledged, in our model at time t_4 only one data packet was acknowledged, and the other three data packets are still in the network. Thus, in our model the instantaneous number of packets of one flow buffered at a switch, which is determined by wnd, may be larger in comparison with the traditional round-based modeling approach.

B. Theoretical Foundation

Now we introduce two new variables wnd_A and wnd_B for our analysis, defined as follows:

$$wnd_A = 2^m, \quad wnd_B = S_{SRU} - \sum_{i=0}^{m} 2^i \tag{7}$$

where

$$m : \sum_{i=0}^{m} 2^i \leq S_{SRU}, \sum_{i=0}^{m+1} 2^i > S_{SRU} \tag{8}$$

In other words, wnd_A (always a power of 2) is the maximum congestion window size reached during the window doubling phase of slow start for an SRU, while wnd_B is the number of leftover packets (if any) in the final window of data. For example, if S_{SRU} = 21 packets, then the evolution of congestion window $cwnd_i$ after the i^{th} RTT round looks as follows: $cwnd_0$ = 1, $cwnd_1$ = 2, $cwnd_2$ = 4, $cwnd_3$ = 8, and, finally, 6 remaining packets are sent. In this example, $wnd_A = cwnd_3 = 8$, and $wnd_B = S_{SRU} - \sum_{i=0}^{3} 2^i = 6$. In general, wnd_A and wnd_B can be calculated as follows:

$$wnd_A = 2^{p_1}, \quad wnd_B = S_{SRU} - 2^{p_2} + 1 \tag{9}$$

where

$$p_1 = \lfloor log_2(S_{SRU} + 1) \rfloor - 1, \quad p_2 = \lfloor log_2(S_{SRU} + 1) \rfloor \tag{10}$$

Theorem 1: The maximum number wnd_{max} of packets simultaneously in flight for a TCP flow in slow start is:

$$wnd_{max} \leq \begin{cases} wnd_A & if\ wnd_B = 0 \\ wnd_A + wnd_B - 1 & if\ wnd_A \geq wnd_B \\ 2wnd_A - 1 & if\ wnd_A < wnd_B \end{cases} \tag{11}$$

where wnd_A and wnd_B are defined by Equation (7).

Proof. We need to consider three possible cases.

Case 1: wnd_B = 0. This case corresponds to the scenario when an SRU is equal to the sum of a geometric progression with the common ratio of 2, i.e., the first inequality in (8) becomes equality. Therefore, the maximum number of packets in the network is the final term of the progression (wnd_A).

Case 2: $wnd_A \geq wnd_B$. Let us assume for the moment that wnd_{max} can be no smaller than wnd_A + wnd_B (i.e., ignore the condition stated in Equation (11)). After some i^{th} RTT round, the congestion window $cwnd_i = wnd_A$ and there are wnd_A packets of an SRU in a network. According to our assumption, at some later moment the congestion window is at least wnd_A + wnd_B. For growing the congestion window from wnd_A to wnd_A + wnd_B, at least one packet should be successfully delivered and acknowledged. Therefore, the maximum number of packets in the network should be no more than wnd_A + wnd_B - 1, which contradicts our assumption.

Case 3: $wnd_B > wnd_A$. Let us assume that wnd_{max} can be no smaller than $2wnd_A$, i.e., the third line of Equation (11) is not true. It means that the congestion window of an SRU is at least $2wnd_A$ at some moment. However, according to (7), $wnd_B < 2wnd_A$, which means that the size of an SRU is insufficient to achieve the congestion window of $2wnd_A$. Thus, we have a contradiction to our assumption.

Having considered all the possible cases, we have proved the theorem. ∎

Thus, the maximum number of packets l_{max} in a network sent by n flows from servers responding concurrently to a data block request is:

$$l_{max} = wnd_{max} n \tag{12}$$

In our further calculations, wnd_{max} is set to the upper bound of its possible values, i.e., is determined by equality in (11). To avoid data losses at the bottleneck link, the buffer size of the bottleneck link should be no less than the size of l_{max} packets:

$$B \geq S_f l_{max}, \quad or \quad B \geq n S_f wnd_{max} \tag{13}$$

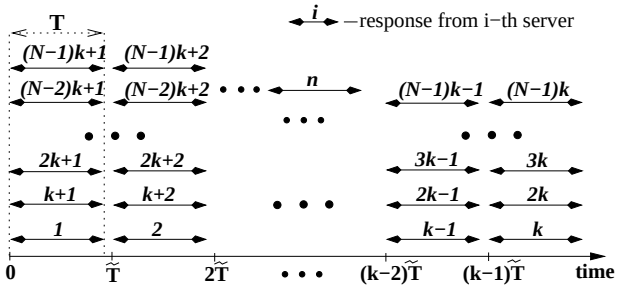

Fig. 2. Schedule of server responses.

From Equation (13), we conclude that to avoid data losses the maximum number of concurrent SRU flows in the network should be no more than:

$$n \leq \Big\lfloor \frac{B}{S_f wnd_{max}} \Big\rfloor \tag{14}$$

The buffer size of a switch B should be large enough for serving at least one SRU without data losses, i.e., n should be no smaller than one. Thus, we derive the applicability limit of our model:

$$\Big\lfloor \frac{B}{S_f wnd_{max}} \Big\rfloor \geq 1 \tag{15}$$

Let us estimate the flow completion time T of an SRU flow; T contains two parts. The first one T_{DATA} describes the data packet and ACK sending processes, and is calculated as:

$$T_{DATA} = \frac{(S_f + S_{ACK})S_{SRU}}{C} \tag{16}$$

The second item T_{ACK} describes the roundtrip delay between a server and a client, and queueing delay at the bottleneck link. The upper bound of the second item is calculated as:

$$T_{ACK} = \lceil log_2(S_{SRU} + 1) \rceil \Big(R + \frac{B}{C} \Big) \tag{17}$$

Finally, assuming that the data sending time and propagation delays R are disjoint, the flow completion time of an SRU flow is the sum of T_{DATA} and T_{ACK}:

$$T = \frac{(S_f + S_{ACK})S_{SRU}}{C} + \lceil log_2(S_{SRU} + 1) \rceil \Big(R + \frac{B}{C} \Big) \tag{18}$$

Since an OS has a finite timer granularity, we need to take that into account in our analysis. The other important thing is non-deterministic behavior of real system scheduling, which we account as a maximum random timer scheduling delay d_{max}. Thus, if the exact value of a completion time is T, then in a real system with timer granularity Δt the flow completion time $\widetilde{T}$, used by the system for scheduling calculations, is:

$$\widetilde{T} = \Big\lceil \frac{T + d_{max}}{\Delta t} \Big\rceil \Delta t \tag{19}$$

Thus, $\widetilde{T}$ is no smaller than T.

To avoid data losses, the responses of servers should be scheduled by batches, with n flows in a batch, at time moments 0, $\widetilde{T}$, $2\widetilde{T}$, ..., $\widetilde{T}\Big(\lceil \frac{N}{n} \rceil - 2\Big)$, $\widetilde{T}\Big(\lceil \frac{N}{n} \rceil - 1\Big)$. If generalizing, the i^{th} server, where $1 \leq i \leq N$, starts responding at time t_i and completes transmission at time $t_i + \widetilde{T}$:

$$t_i = \widetilde{T} \cdot mod(i-1, \Big\lceil \frac{N}{n} \Big\rceil) \tag{20}$$

where $mod(i-1, \Big\lceil \frac{N}{n} \Big\rceil)$ is the remainder when dividing $i-1$ by $\Big\lceil \frac{N}{n} \Big\rceil$. We define the number of batches of concurrent flows k as:

$$k = \Big\lceil \frac{N}{n} \Big\rceil, \tag{21}$$

In Figure 2, we show the proposed schedule of server responses, which has a maximum of n concurrent TCP flows in the vertical dimension and k batches of concurrent flows serialized in the horizontal (time) dimension. We need to mention two important things about the scheduling process. First, the number of SRUs may not always pack perfectly into a rectangle, which is shown by the time slot of the response of the N^{th} server. Second, the number of concurrent SRU flows is not always equal to n. For example, if $N = 6$ and $n = 5$, then according to Equation (20), there are two batches of concurrent SRUn flows with three flows in each of them. In practice, scheduling server responses is done as follows:

1) Requests to data servers are scheduled according to Equation (20).
2) Upon receiving a request, each server responds to it immediately.

Thus, the described scheduling implementation realizes the scheduling of server responses according to Equation (20).

We have shown that the response time t_r of a data block request under scheduling defined by Equation (20) is:

$$t_r = \widetilde{T}\Big(\Big\lceil \frac{N}{n} \Big\rceil - 1 \Big) + T + d_{max} \tag{22}$$

Thus, we have proved the following theorem about the goodput of a data center application, which is defined as the ratio between the data block size and the response time for a data block request:

Theorem 2: The maximum goodput of an application at a data center using the lossless scheduling specified by Equation (20) is given by Equations (1)-(4).

Next, we will discuss the practical use of our solution in data center applications.

IV. Practical Issues

A common structure for applications using data centers is a partition/aggregate design [11] as shown in Figure 3. The partition/aggregate design is widely used in large scale Web applications (e.g., search, social network content composition, advertisement selection). An application user makes a request, e.g., a search query, which is initially processed by a server called high-level aggregator (HLA). The HLA splits the request into several parts, e.g., a text part and a picture part, and sends it to servers called medium-level aggregators (MLA), one in each server rack, that are responsible for different parts of the request. Each MLA sends a request to workers that are data servers in the same rack as an MLA storing the information about a particular part of a request. The workers finally process a request and reply to an MLA. After completing a request, an MLA forms a response and sends it back to the HLA. On receiving the response from all MLAs, the HLA responds to the client by sending the data received from MLAs.

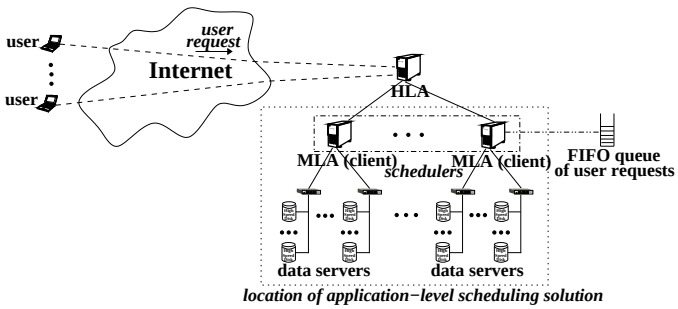

Fig. 3. Partition/aggregate large-scale application. The deployment of application-level scheduling in a data center.

Adopting our scheduling mechanism on MLAs will help to mitigate the TCP-incast problem. In Figure 3, we show where our scheduling solution is used to control the responses from data servers. The parameters of our solution, which are the data center network characteristics, are known to MLAs, and the whole scheduling process is performed by MLAs. If multiple users make requests to a data center application, then each MLA processes them through a First-In First-Out (FIFO) queue that stores the requests. On receiving the request from an HLA, an MLA schedules requests to data servers according to Equation (20), and each data server responds to the received request immediately. Thus, at each moment only one request is scheduled for responses from servers to an MLA. In Figure 4, we demonstrate how two user requests are processed, and show the processing stages of request #1 as an example. As we can observe, the MLA does not process request #2 until it receives all the responses on request #1 from servers (the end of stage **(e)**).

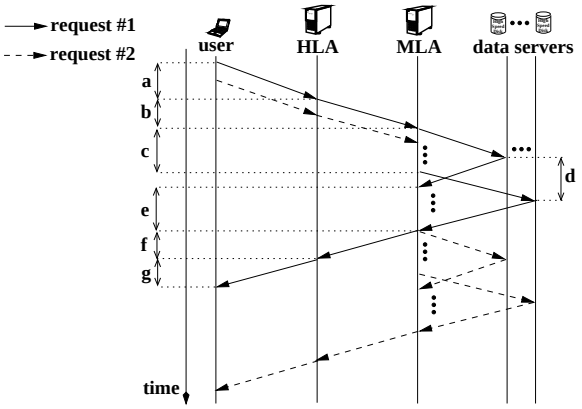

Fig. 4. Request processing and its stages: **(a)** delivering the request from an end-user to the data center (HLA); **(b)** delivering the split part of the original request from the HLA to the MLA; **(c)** sending requests to data servers by the MLA according to the proposed scheduling algorithm; **(d)** sending responses from data servers; **(e)** gathering the responses from data servers by the MLA; **(f)** delivering the response on the split part of the original request from the MLA to the HLA; **(g)** composing the final response to the end-user by the HLA, and delivering it from the HLA to the end-user.

Figure 5 demonstrates a typical fat-tree topology of a data center network. Since our scheduling mechanism is deployed over MLAs, and each MLA interacts only with servers within the same rack [11], scheduled traffic makes only two hops: 1) server - edge switch; 2) edge switch - MLA. Thus, there is no interference among traffic generated by MLA requests that are triggered by the same HLA request. Moreover, the scheduling approach is not limited to TCP only, and can potentially be

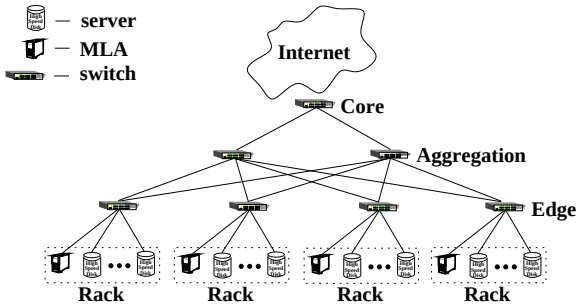

Fig. 5. Fat-tree topology of a data center network.

adapted to other transport protocols, which makes application-level scheduling an attractive solution for tackling more general oversubscription problem in data centers [13]. However, although we base our approach on the lossless scheduling, a reliable transport protocol is still required because losses not related to buffer overflow may happen. Finally, our model assumes that MLAs do not use persistent TCP connections; we leave the extension of our scheme to the case of persistent TCP connections as future work.

The potential concern that may arise is interaction of traffic of different applications within the same rack when an MLA server runs several applications with partition/aggregate design. The solution can be to use a special *MLA controller* (software module) deployed over the MLA server, the main goal of which is to ensure that MLA requests of different applications to the MLA server are processed in FIFO manner. Thus, a bottleneck link between an edge (Top-of-Rack) switch and the MLA server is shared by traffic generated by servers of only one application at a time.

V. Simulation Results

To validate our model, we perform simulations using the *ns-2* [1] network simulator. We use the network topology shown in Figure 1. The bottleneck link and each access link has 1 Gbps capacity and 0.025 ms propagation delay, and use the DropTail scheme. Thus, the propagation RTT for each server is 0.1 ms. The buffer size of the bottleneck link is 32 KB. These parameters characterize a typical data center [6]. We set the OS timer granularity Δt to 1 ms as the minimum value used in OS [5]. We use NewReno TCP in our evaluation. To account for real system scheduling variance, the response of the i^{th} server starts at time t_i+d, where t_i is determined by Equation (20), and d is a delay uniformly distributed between 0 and d_{max} = 20 μs. We run five simulations for each of the considered parameter settings.

A. Fixed SRU size

In this section, we explore the performance of a data center in a simple case when the SRU size is fixed to 10 KB. We vary the number of servers from 1 to 1000.

Figure 6 shows that the goodput converges to approximately 200 Mbps, and the simulation results show reasonable qualitative and quantitative agreement with the model. Specifically, goodput is non-monotonic with the number of servers, and there are sudden decreases of the theoretical and simulation

values of the goodput, the amplitude of which diminishes as the number of servers is increased. For example, the theoretical value goes from 203 Mbps for 15 servers to 162 Mbps for 16 servers. Such a jagged trajectory for the goodput is due to the packetized traffic and the integer number of batches of simultaneous responses from servers, which depends on $\lceil \frac{N}{n} \rceil$. The "length" of each step in the zig-zag path is determined by the number of servers n responding concurrently, which is $n = 5$ in this scenario.

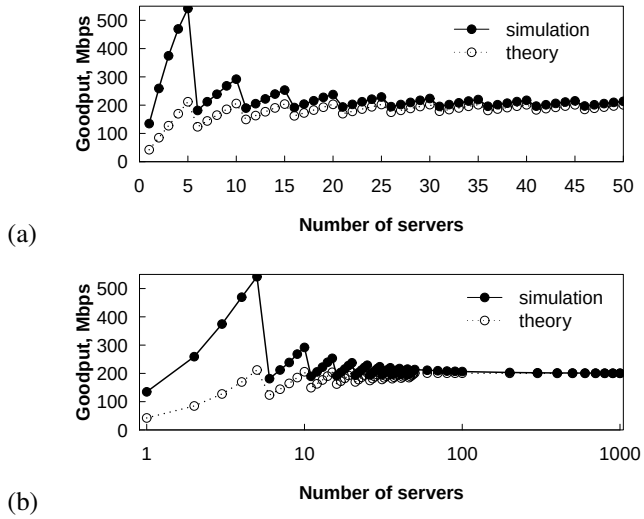

Fig. 6. Goodput for different numbers of servers with a fixed 10 KB SRU size: (a) small data center; (b) large data center.

There is good agreement between the simulation and analytical results, both qualitatively and quantitatively, as the number of servers increases. However, there is a large discrepancy between the theoretical results and the simulation results for small configurations, say with $N \leq 5$ servers. This gap is explained by the reduced buffer contention in scenarios with limited concurrency. That is, to estimate an SRU flow completion time, we assume that the buffer at a bottleneck link is full, i.e., the queueing delay at that link is $\frac{B}{C}$, where B and C are respectively the buffer size and bottleneck link capacity. However, since wnd_{max} = 6 and B = 32 KB, our model overestimates the maximum queueing delay at a bottleneck link for no larger than five servers. Moreover, since n = 5, all servers can respond simultaneously, and the OS timer granularity does not affect the goodput of an application. If $N \geq 6$ then there are at least two batches of concurrent SRU flows, i.e., $k \geq 2$, which means that the OS timer granularity should be taken into account for scheduling. According to Equation (22), the growth of the number of flows leads to more influence of $\tilde{T}$ on the request completion time in comparison with T. Therefore, the discrepancy between the model and the simulation results diminishes with the increase of the number of servers.

Without using our mechanism (not shown in Figure 6), the goodput drastically drops from 600 Mbps to 8 Mbps as the number of servers rises from 5 to 20 servers. The goodput continues dropping to 1.3 Mbps when the number of servers is 200, and does not exceed 3 Mbps with the further increase of the number of servers.

B. Varying SRU size

We next consider a more realistic scenario in which the SRU size is scaled automatically based on the number of servers. Specifically, the data block size is fixed to 1 MB, and the SRU size is $\frac{1}{N}$ MB, where N is the number of servers. To study how the number of servers affects the performance, we vary their numbers between 22 and 1000. The minimum number of servers is 22 according to the limitation of our model expressed by Equation (15).

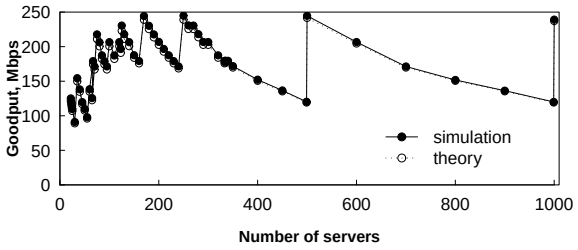

Fig. 7. Goodput for different numbers of servers with varying SRU size.

In Figure 7, we observe that the simulated goodput closely follows the theoretical value. In general, the simulated goodput slightly exceeds the theoretical one, which is expected since we estimate the upper bound of flow completion time T. In addition, there is a distinct sawtooth structure to the plots. We explain this behavior as follows. The goodput is determined by the response time of a data block request t_r. According to Equations (18), (19), and (22), t_r is a function of $\lceil log_2(S_{SRU} + 1) \rceil$, where $\lceil\ \rceil$ function reflects the packet granularity of traffic, and $\lceil \frac{N}{n} \rceil$, where $\lceil\ \rceil$ function represents the integer number of batches k of concurrent SRU flows (see Figure 2). The combination of these two $ceiling$ functions leads to the sudden jumps of the goodput function (e.g., from 119 Mbps for 499 servers up to 241 Mbps for 500 servers).

If not using our scheme (not shown in Figure 7), then the goodput drastically drops from more than 900 Mbps to less than 40 Mbps as the number of servers rises from 10 to 15 servers. With the further increase of the number of servers, the goodput continues dropping to 10 Mbps and 1.3 Mbps for 60 and 130 servers, respectively.

C. Influence of the buffer size

To investigate the performance of a data center for different switch buffer sizes, we vary it between 32 KB and 128 KB for 25 servers, and between 8 KB and 128 KB for 100 servers. The chosen intervals of the varied parameters fit Inequality (15) that describes the limitations of our model. Figure 8 shows the correspondence between the simulation results and the model. In particular, wnd_{max} is 24 for 25 servers, and 6 for 100 servers. The value of wnd_{max} determines the minimum buffer size of a switch for making our model applicable.

In addition, we observe that the discrepancy between the theoretical value of the goodput and the simulation value increases with the increase of the buffer size, which we explain as follows. In our analysis, we assume that the roundtrip delay

between a client and a server is the sum of the propagation delay and a queueing delay at a full switch buffer, which is expressed in Equation (17). However, as this is an upper bound estimation of the roundtrip delay between a client and a server, the increase of the buffer size leads to the increase of the roundtrip delay, and, therefore, the smaller theoretical goodput compared to the simulated goodput.

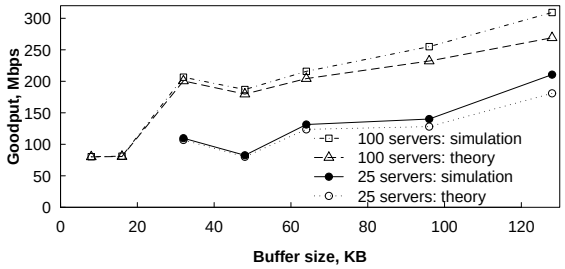

Fig. 8. Goodput for different switch buffer sizes.

Without applying our scheme (not shown in Figure 8), the goodput monotonically increases from 0.3 Mbps to 29 Mbps for 100 servers. If there are 25 servers, and the buffer size is no larger than 96 KB then the goodput does not exceed 39 Mbps; larger buffer size does not induce throughput collapse and supports the goodput more than 900 Mbps.

D. Influence of the timer granularity

We study how the timer granularity affects the performance by varying it between 0.001 ms and 20 ms. In addition, we run the simulations for 25, 50, 100, 200, and 500 servers. In Figure 9, we see that goodput is higher with fine-grain timers, and degrades as the timer granularity increases. Furthermore, the timer granularity has a greater influence when there are many servers. In particular, the goodput is approximately the same for 25 servers with timers of up to 1 ms, and for 200 servers with timers of up to 0.1 ms. Such a behavior is because the goodput is a function of $\left\lceil \frac{T+d_{max}}{\Delta t} \right\rceil \Delta t$, where T is an SRU completion time, the same for different timer granularities Δt. In general, the simulation results and the model fit well.

The goodput is generally a monotonically decreasing function as timer granularity increases. However, there is an anomaly with 2 ms timers for 25 and 50 servers, which we explain as follows. If the number of servers is 25, then the flow completion time of an SRU flow T is 2.858 ms. This completion time is tightly estimated (as 3 ms) if the timer granularity Δt is 1 ms or 3 ms, but poorly estimated (as 4 ms) if Δt is 2 ms. A similar phenomenon occurs for 50 servers, where T = 2.329 ms, for the same reason. We want to emphasize that our solution does not require fine-grained timers with granularity smaller than 1 ms, which is the minimum currently used in OS. However, our simulations show that the application performance can be better in some data center configurations if using timers with granularity smaller than 1 ms.

VI. Related Work

Chen *et al.* [14] studied the throughput collapse of TCP incast via conducting experiments and modeling the behavior.

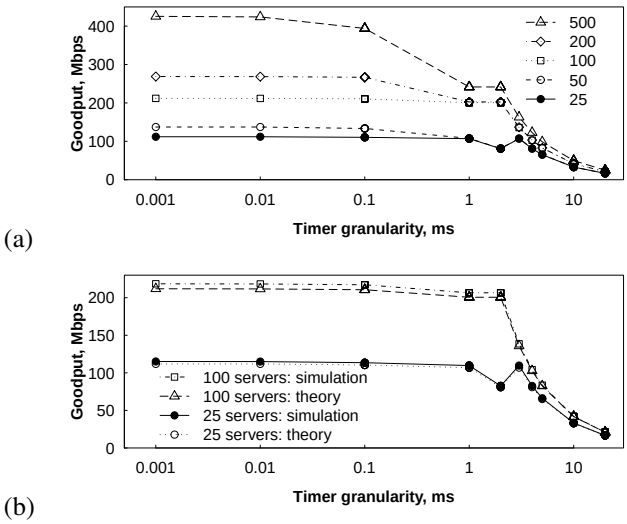

Fig. 9. The goodput at a Typical Data Center for different timer granularities: (a) theoretical results for different numbers of servers; (b) comparison with simulation results.

In [15], the authors explored how Quantized Congestion Notification (QCN) [16], an Ethernet layer congestion control mechanism, operated under TCP-incast scenario in data centers, and proposed some modifications to improve its efficiency. Shpiner and Keslassy [7] proposed a new architecture called Hashed Credits Fair (HCF) to avoid TCP throughput collapse in data centers. The key idea of the scheme is to serve the incoming traffic through two queues (high priority and low priority queues) instead of one traditional DropTail one.

Phanishayee *et al.* [6] explored the TCP-incast problem and some strategies for mitigating it. In particular, the authors examined the use of different TCP variants and the decrease of TCP retransmission timeout. Allman *et al.* [17] proposed Limited Transmit for improving TCP recovery from packet losses for small window sizes, which is the same problem as in the case of TCP-incast. However the proposed solution is efficient only when the congestion is not severe. In [5], the authors proposed to use microsecond-granularity retransmission timeouts in data center networks, which effectively solves the problem of TCP throughput collapse. Ghobadi *et al.* [18] showed that TCP pacing was not effective in data center networks due to their small propagation RTTs. Zhang *et al.* [19] proposed an analytical model for understanding the causes of the TCP throughput collapse. In particular, they found out that there were two types of TCP timeouts leading to TCP throughput degradation.

In [11], the authors proposed a transport protocol named DCTCP (Data Center TCP), which addresses the TCP-incast problem. The protocol relies on two main features. The first one is to employ ECN (Explicit Congestion Notification) [20], and the second one is to adjust the multiplicative decrease coefficient used for controlling congestion window according to the fraction of ECN-marked packets in the last window of data. Wu *et al.* [21] designed ICTCP (Incast congestion Control for TCP) as a receiver-side congestion control scheme.

The other potential solution is to increase the queue buffer size, making it proportional to the number of servers to decrease the chance of a packet loss [22], [23]. However, a large buffer size induces high queueing delays, which is inappropriate for data centers. Moreover, it requires an additional fast memory like Static Random Access Memory (SRAM), which increases the cost and complexity of a switch [24].

Krevat *et al.* [28] discussed a variety of possible application-level solutions to TCP throughput collapse problem. In particular, they suggested several ideas that may be explored for mitigating the TCP-incast problem: increasing request sizes; limiting the number of synchronously communicating servers; throttling, staggering, and global scheduling of data transfers. However, the authors did not provide any details on particular designs.

The main difference of our work from prior work is that, while relying on the ideas mentioned in [28], to the best of our knowledge, our paper is the first to propose and evaluate a particular mechanism at the application level for the TCP-incast problem in data center networks. Moreover, our approach does not require any changes in TCP stack or network switches. Since our scheme operates at the application level, its deployment is simpler than solutions requiring changes in the TCP stack; modifying the TCP stack in a data center having thousands of servers can be a challenge. Thus, deployability is the key advantage of our proposal over solutions relying on modifying a transport protocol.

VII. Conclusion

In this paper, we explored the TCP-incast throughput collapse problem in data center networks from an application-level perspective. In particular, we presented the model and analyzed the performance of the application-level approach under TCP-incast scenario. The main idea of the approach is to schedule the server responses to data requests so that no packet losses occur at the bottleneck link. The main result we derive is the achievable goodput of a data center application under lossless scheduling. The results indicate non-monotonic goodput that is highly sensitive to specific parameter configurations. The simulations confirmed the validity of our model and its theoretical results. Our future work will involve implementing our approach in a real data center network, and evaluating its performance for different application traffic scenarios.

Acknowledgement

Financial support for the work was provided in part by iCORE (Informatics Circle of Research Excellence) in the Province of Alberta, and by NSERC (Natural Science and Engineering Research Council) Discovery Accelerator RGPAS-380438 45185.

References

[1] S. McCanne and S. Floyd, *ns Network Simulator. http://www.isi.edu/nsnam/ns/.*

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings ACM SIGCOMM*, August 2008.

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proceedings ACM SIGCOMM*, August 2009.

[4] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, and P. Pat, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings ACM SIGCOMM*, August 2009.

[5] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Anderson, G. Ganger, G. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proceedings ACM SIGCOMM*, August 2009.

[6] A. Phanishayee, E. Krevat, V. Vasudevan, D. Anderson, G. Ganger, G. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," in *Proceedings FAST*, February 2008.

[7] A. Shpiner and I. Keslassy, "A Switch-Based Approach to Throughput Collapse and Starvation in Data Centers," in *Proceedings IEEE IWQoS*, June 2010.

[8] M. Podlesny and C. Williamson, "An Application-Level Solution for the TCP-incast Problem in Data Center Networks," in *Proceedings ACM/IEEE IWQoS*, June 2011.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proceedings ACM SOSP*, October 2003.

[10] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Zelenka, and B. Zhou, "Scalable Performace of the Panasas Parallel File System," in *Proceedings USENIX Conference on File and Storage Technologies*, February 2008.

[11] M. Alizadeh, A. Greenberg, D. Maltz, j. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings ACM SIGCOMM*, September 2010.

[12] S. Rewaskar, J. Kaur, and F. Smith, "A Performance Study of Loss Detection/Recovery in Real-world TCP Implementations," in *Proceedings IEEE ICNP*, October 2007.

[13] S. Kandula, J. Padhye, and P. Bahl, "Flyways To De-Congest Data Center Networks," in *Proceedings ACM HotNets*, October 2009.

[14] Y. Chen, R. Griffith, J. Liu, and A. Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *Proceedings WREN*, August 2009.

[15] P. Devakota and A. L. N. Reddy, "Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers," in *Proceedings IEEE MASCOTS*, August 2010.

[16] R. Pan, B. Prabhakar, and A. Laxmikantha, *QCN: Quantified Congestion Notification An Overviews*, http://www.ieee802.org/1/files/public/docs2007/au-prabhakar-qcn-description.pdf.

[17] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," IETF RFC 3042, January 2001.

[18] M. Ghobadi, G. Anderson, Y. Ganjali, J. Chu, L. Brakmo, and N. Dukkipati, "TCP Pacing in Data Center Networks," April 2010.

[19] J. Zhang, F. Len, and C. Lin, "Modeling and Understanding TCP Incast in Data Center Networks," in *Proceedings IEEE INFOCOM*, April 2011.

[20] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP," January 1999, rFC 2481.

[21] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP," in *Proceedings ACM CoNEXT*, November 2010.

[22] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proceedings ACM SIGCOMM 2004*, August 2004.

[23] R. Morris, "TCP Behaviour with Many Flows," in *Proceedings IEEE ICNP*, October 1997.

[24] S. Iyer, R. Kompella, and N. McKeown, "Designing Packet Buffers for Router Line Cards," Technical Report, TR-02-HPNG-031001, October 2002.

[25] Y. Gu, D. Towsley, C. Hollot, and H. Zhang, "Congestion Control for Small Buffer High Speed Networks," in *IEEE INFOCOM 2007*, April 2007.

[26] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One More Bit Is Enough," in *Proceedings ACM SIGCOMM 2005*, August 2005.

[27] N. Dukkipati, M. Kobayashi, and N.McKeown, "Processor Sharing Flows in the Internet," *Thirteenth International Workshop on Quality of Service (IWQoS)*, June 2005.

[28] E. Krevat, V. Vasudevan, A. Phanishayee, D. Anderson, G. Ganger, G. Gibson, and S. Seshan, "On Application-based Approaches to Avoiding TCP Throughput Collapse in Cluster-based Storage Systems," in *Proceedings Supercomputing*, November 2007.