Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
December 2012, 19(6): 52–62
www.sciencedirect.com/science/journal/10058885
The Journal of China Universities of Posts and Telecommunications
http://jcupt.xsw.bupt.cn

DATCP: deadline-aware TCP for the commoditized data centers

ZHANG Peng (✉), WANG Hong-bo, LI Yang-yang, DONG Jian-kang, CHENG Shi-duan

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract

Many flows in data centers have deadlines and missing deadlines would hurt application performance such as affecting response quality in Web applications or delaying computing jobs in MapReduce-like systems. However, transmission control protocol (TCP) which is widely used in data centers now cannot provide deadline-aware transmission service. Service differentiation only distinguishes flows with different priority but is unable to guarantee completion time. In this paper, we propose a new protocol named deadline-aware TCP (DATCP) to provide deadline-aware transmission service for the commoditized data centers, which can be used as a flexible method for flow scheduling. DATCP combines flow urgency and importance to calculate precedence. Flow urgency is dynamically adjusted according to the gap between desired rate and actual throughput. Setting importance can avoid starving the important but no-urgent flows. Furthermore, a flow quenching method is presented which allows as many high precedence flows as possible to meet deadlines under heavy network load. By extensive simulations, the performance of DATCP was evaluated. Simulation results show that DATCP can make flows meet deadlines effectively.

Keywords data center networks (DCN), deadline-aware transmission service, TCP, flow scheduling

1 Introduction

Currently data centers are hosting more and more applications, including online services such as Web search, social networks, and off-line application such as data mining based on MapReduce [1]. All of these applications generate many flows. Online services are comprised of multiple components like authentication, database, file system, management, and so on. These components are deployed across entire data center with intricate dependencies [2]. For example, social network service requires access to an authentication service for verifying users, and composes web page content by aggregating data spread across different data stores. Many flows are generated during multiple service components communicating with each other. Moreover, there are also a lot of flows in MapReduce workload at each computing stage, such as job initialing, data distributing, map phase, shuffle phase, reduce phase and result aggregation.

Flows in data centers usually have deadlines and missing deadlines would hurt application performance. For online service, flows generated by interactive components are usually latency-sensitive. Missing deadlines typically hurts response quality and wastes network bandwidth. Ultimately, operator revenue is affected [3]. In MapReduce, although flows are usually not latency-critical, the flows need to complete early to finish the computing job. For example, in shuffle phase, reduce tasks begin only if all outputs are received from multiple map tasks. Hence flows not complete in time will delay the computing jobs and then affect utilization of MapReduce cluster [4].

However, TCP which makes up more than 95% of the data center traffic [5–6] cannot provide deadline-aware transmission services. TCP tries to divide bottleneck bandwidth fairly and have no any application information. Moreover, service differentiation like DiffServ [7] and end-to-end mechanisms [8] can distinguishes flows with different priorities. Flows with high priority get more share than ones with low priority. But service differentiation only provides flows with proportional throughput and is unable to guarantee the flow completion time.

Received date: 28-05-2012
Corresponding author: ZHANG Peng, E-mail: zhp@bupt.edu.cn
DOI: 10.1016/S1005-8885(11)60318-X

In this paper, we propose a new transport protocol named DATCP to provide deadline-aware service for the commoditized data centers, which can be used as a flexible method to schedule flows. DATCP dynamically adjusts flow urgency based on the gap between desired rate and actual throughput. High urgency flows would get more bandwidth share. The urgency is increased every round-trip time (RTT) until actual throughput is equal or greater than desired rate so that the flow would meet deadline. To avoid starving the important but not-urgent flows, DATCP combines importance and urgency of flows to calculate the precedence according to the urgent/important matrix [9]. The importance is the same as conventional priority and is due to what the flow is used for. Therefore, the precedence of TCP flows can dynamically be changed according to network situation. When network load is too heavy, not all flows can meet deadlines, a flow quenching method is presented which makes low precedence flows quenched to allow other high precedence flows to meet deadlines.

DATCP needs the applications get flow size at flow initiation time. It is feasible in data centers. For example, many web applications have knowledge of flow size in advance. This holds for MapReduce. Even for application where this condition does not hold, the application designer can typically provide a good estimate for the expected flow sizes. Moreover, DATCP only needs minor modification to existing TCP and makes use of explicit congestion notification (ECN) [10], a feature already available in current commodity data center switches. Deploying DATCP in datacenters has no effect on the external traffic as connectivity to the external Internet is typically managed through load balancers and application proxies that effectively separate internal traffic from external [6].

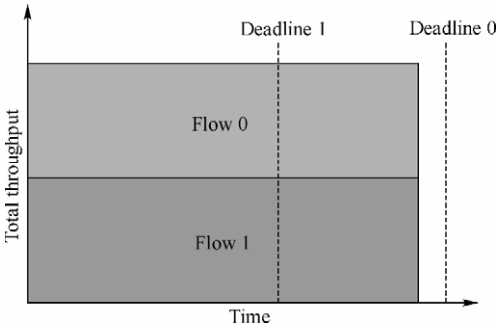
By extensive simulations, the performance of DATCP was evaluated. Simulation results show that DATCP can make flows meet deadlines effectively and without starving the important but not-urgency flows. A scenario of MapReduce was also considered to show how to schedule flows using DATCP.

The remainder of this paper is organized as follows. In Sect. 2 we introduce background and related works. The DATCP algorithm is presented in Sect. 3. Sect. 4 provides the evaluation results. Finally, Sect. 5 concludes the paper.

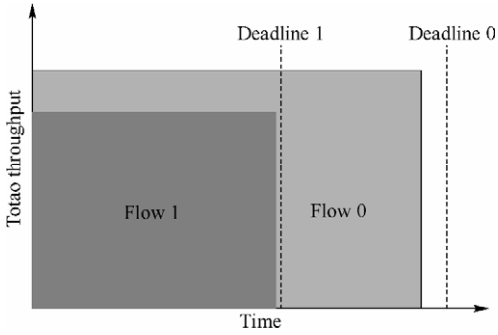
2 Background and related works

2.1 Benefits of deadline-aware

We take two examples in Ref. [3] to show benefits of deadline-aware. The first one, two flows share a bottleneck link and one of them has a tighter deadline than the other. Today's TCP tries to share link fairly and the flows would finish at similar times, so only one flow meets the deadline, as shown in Fig. 1(a). However, if flows are deadline-aware and the deadlines are set to flows, flow 1 would get enough share to meet the deadline, and then both of the two flows would meet the deadlines, as shown if Fig. 1(b).



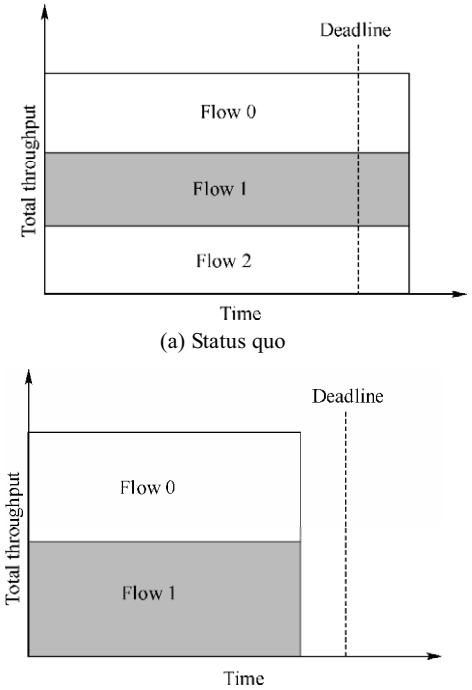
(a) Status quo



(b) Deadline aware

Fig. 1 Two flows with different deadlines

The second one, multiple flows share a bottleneck link and have same deadline. Assume that network load is heavy and network capacity is not enough to allow all the flows to meet the deadline. With today's TCP, as illustrated in Fig. 2(a), all flows get their fair share and finish at similar times and all of them would miss the deadline. Given flows deadline-aware, if some flows are quenched, there would be allow other flows to meet the deadline, as shown in Fig. 2(b).



(a) Status quo

(b) Deadline aware (flow 2 is quenched)

Fig. 2 Multiple flows with the same deadline

2.2 Flows in MapReduce-like systems

MapReduce is a computing framework proposed by Google for processing highly distributable problems across huge datasets using a large number of computers [1]. There are a lot of flows in Hadoop at each computing stage, such as job initialization, data distributing, map phase, shuffle phase, reduce phase and result aggregation. Fig. 3 presents the dataflow of Hadoop [11]. Input data is divided into splits, and a distinct map task is launched for each split. Inside each map task, map stage applies a map function to the input data, and spill stage stores map output on local disks. In addition, a distinct reduce task is launched for each partition of the map outputs. Inside each reduce task, the copier and merge stages also called shuffle stage run in a pipelined fashion, fetching the relevant partition over the network and merging the fetched data respectively; after that, the sort and reduce stages merge the reduce inputs and apply the reduce function respectively.

The size of most of these flows can be known before initiation. At job initialization, the JobTracker distributes the job JAR package to the TaskTracker. The size of the job JAR package is known in advance. For data distribution of Hadoop distributed file system (HDFS) [11] and Google file system (GFS) [12], block size is preset. At shuffle step, since the reducers copy map outputs after the map tasks have finished, the flow size is also known in advance.

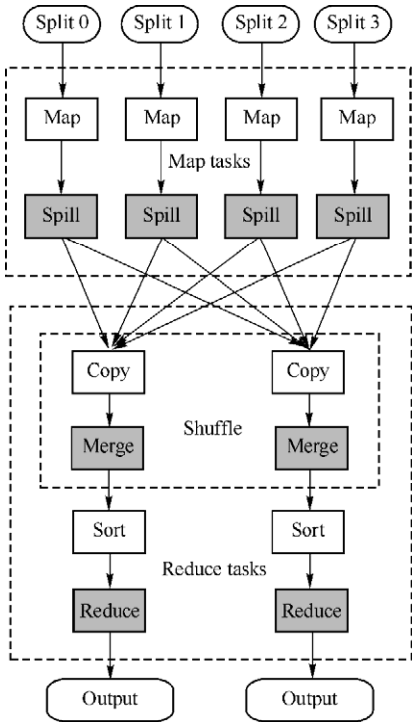


Fig. 3 Dataflow graph of the Hadoop

Map tasks within a job always have similar lengths because they run the same function [13]. So job scheduler can speculate finish time of maps. Flows of jobs which are map up of few flows should be set shorter deadlines. To reduce job completion time and increase utilization of Hadoop cluster, flows deadlines should be set carefully. Flows deadlines setting can be used for flows scheduling. Therefore, DATCP provide a flexible method for Hadoop to schedule flows.

2.3 Related works

There are some research works on data center transport protocol. A hot problem of TCP in data centers is the incast throughput collapse [14–15], where application throughput drastically collapses when multiple senders simultaneously send data to a receiver. But these works are deadline-agnostic. Other works propose new data center transport protocols [3,6]. DCTCP [6] aims to ensure low latency for short flows and good utilization for long flows by reducing switch buffer occupation meanwhile minimizing buffer oscillation. Reducing the switch buffer occupation can lower queuing time and benefit short flows while minimizing the oscillation can maintain high

throughput of links and do not affect the throughput of long flows. However, DCTCP does not provide the deadline-aware service and even has no service differentiation. The work most related to ours is D^3 [3]. D^3 uses explicit rate control to apportion bandwidth according to flow deadlines. Given a flow's size and deadline, source end hosts request desired rates to switches. The switches assign and reserve allocated rates for the flows. However, most data center switches do not support rate allocation and reservation due to the trend of constructing data centers with commodity servers and switches [16–18]. Unlike D^3, DATCP proposed in this paper only needs minor modification to existing TCP and makes use of a feature already available in current commodity data center switches.

3 The DATCP algorithm

The basic idea is that DATCP adjusts flow precedence dynamically to maintain flow rate and then make flows meet their deadlines. Flow rate is related to not only its precedence but also the number and precedence of other flows. This is because that flow precedence only brings proportional throughput differentiation, i.e., flows receive their weighted fair share of the network bandwidth. Therefore, flow precedence should be adjusted dynamically to let flow get the desired throughput.

3.1 Calculating precedence based on both urgency and importance

Flow precedence is inspired by the urgent/important matrix [9], which is a powerful way of thinking about priority, as shown in Fig. 4.

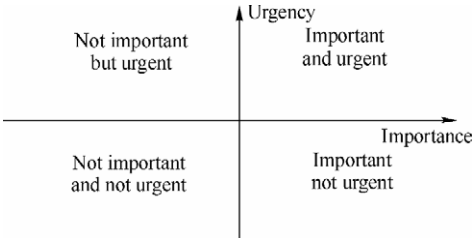


Fig. 4 Urgent/Important matrix

Using it helps people overcome the natural tendency to focus on urgent activities, so that you can keep clear enough time to focus on what's really important. Similarly, we define urgency and importance to flows. The urgency of a flow, which is adjusted dynamically, reflects the gap between its bandwidth demand and actual throughput. The importance of a flow is up to its usage and what service it belongs to. Then we calculate precedence based on both urgency and importance. The precedence is similar to conventional priority.

Algorithm 1 demonstrates the scheme of flow precedence calculating. The reason why $R_{avg} = R_{last}$ in slow start phase is that it can make R_{avg} converge quickly. Flow urgency is adjusted dynamically according to the gap between desired rate $R_{desired}$ and actual throughput R_{avg}. When the actual throughput is smaller than the flow demand, the urgency is increased to try to get more capacity share. When the actual throughput is larger than the flow demand, the urgency is decreased. To avoid too large or negative urgency, maximum and minimum urgency are set. Flow importance is preset according to its usage and what service it belongs to. Setting importance can avoid starving the important but not-urgent flows. We calculate precedence P based on both importance and urgency. The precedence P is also updated every window of data (roughly one round trip time (RTT)).

Algorithm 1 Flow precedence calculating

Description: flow precedence is calculated based on both urgency and importance. The importance is preset. The urgency is adjusted dynamically according to the gap between its desired rate and actual throughput.

Input:

flow size V_{total}, flow deadline $T_{deadline}$, flow importance I.

Variable Definition:

P: flow precedence.

U: flow urgency.

U_{max}: the maximum flow urgency.

U_{min}: the minimum flow urgency.

V_{rtt}: data volume sent last RTT.

V_{remain}: remaining data volume.

T_{rtt}: length of last RTT.

T_{remian}: remaining time before the deadline.

T_{now}: current time.

R_{last}: flow throughput of last RTT.

R_{avg}: present actual throughput.

$R_{desired}$: flow bandwidth demand.

α: parameter for updating R_{avg}.

φ : proportion of importance during precedence calculation.

G: the maximum granularity of urgency adjusting.

g: the granularity of urgency adjusting

Initialization:

$V_{remain} = V_{total}$;

$T_{remian} = T_{deadline} - T_{now}$;

Algorithm:

When receive every window of data (roughly one RTT)

$V_{remain} = V_{remain} - V_{rtt;}$;

$T_{remian} = T_{remian} - T_{rtt}$;

$R_{last} = V_{rtt} / T_{rtt}$;

if (in slow start phase) then

$R_{avg} = R_{last}$;

else if (in congestion avoidance phase) then

$R_{avg} = \alpha R_{avg} + (1-\alpha) R_{last}$;

end if

$R_{desired} = V_{remain} / T_{remian}$;

$g = \text{abs}(R_{desired} - R_{avg}) / R_{avg}$

if ($g > G$) then

$g = G$;

if ($R_{desired} > R_{avg}$) then

$U += g$;

if ($U > U_{max}$) then

$U = U_{max}$;

end if

else then

$U -= g$;

if ($U < U_{min}$) then

$U = U_{min}$;

end if

end if

$P = \varphi I + (1-\varphi) U$;

3.2 Adjusting congestion window according to given precedence

The idea is that flows with varying precedence take different response to network congestions. High precedence flows should cut window responding to congestion more mildly and low precedence flows more severely. This can achieve proportional throughput differentiation, e.g., throughput of a flow with precedence 2 is twice as a flow with precedence 1. In other words, high precedence flows have higher proportion of the capacity of the shared link than low precedence flows. DATCP dynamically allocates bandwidth by per flow according to the precedence not by per precedence.

As shown in Fig. 5, DATCP senders send packets to receiver. If network congestion happens, the switches mark the incoming packets. Then receivers echo the ACKs of marked packet to sender, and then congestion is detected by sender and sender take reaction according to its precedence.

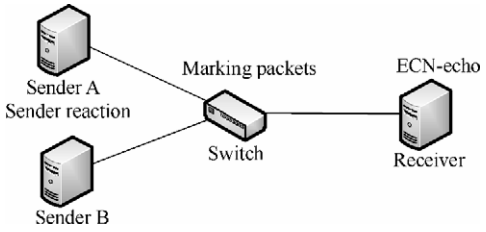


Fig. 5 The combination of DATCP and ECN

1) Detecting the network congestion

DATCP uses random early detection queue (RED) in conjunction with ECN which is already available in current commodity switches. RED/ECN monitors the average queue size and marks incoming packets based on statistical probabilities instead of dropping. DATCP use a variant of RED which marks the incoming packets based on instantaneous queue length instead of average queue length, as shown in Fig. 6. As the senders could receive marked ACKs during the first one or two RTTs to tame the size of follow up bursts, this could prevent buffer overflows. This is the same with DCTCP [6]. But the difference between DATCP and DCTCP is that we set low and high marking threshold different value. The benefit of this is that we can speculate congestion degree more accurately. In DCTCP, the low and high marking threshold are equal. If the instantaneous queue length exceeds K, all packets are marked and the congestion degree is hard to calculate accurately.

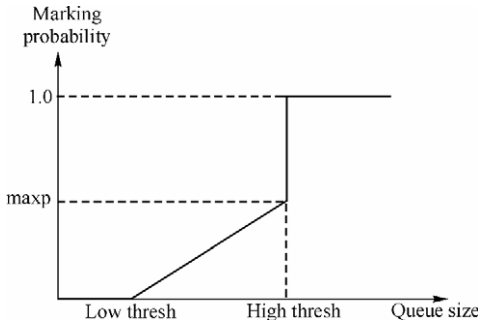


Fig. 6 DATCP use a variant of RED: marking is based on the instantaneous rather than average queue length

We do not change the ECN-support TCP receiver. Upon receiving a marked packet, the TCP receiver echoes back this congestion indication using the ECN-Echo flag in the TCP header in a series of acknowledgements (ACKs) packets until it receives confirmation from the sender that the congestion notification has been received [10].

2) DATCP sender behavior

Standard TCP use additive increase multiplicative decrease (AIMD) algorithm to adjust congestion window

size. The AIMD (a, b) policy increases the congestion window by a fixed amount a every round trip time. When congestion is detected, the sender decreases the congestion window by a multiplicative factor b. For standard TCP, a = 1, b = 0.5.

Flows with varying AIMD parameters take different responses to network congestions to get different throughput. DATCP does not change AIMD algorithm of TCP except the multiplicative decrease factor b. High precedence flows are set smaller b and low precedence flows are set bigger b. The reason we choose the multiplicative decrease factor b rather than the additive increase amount a is that this can minimize the traffic burst. Let's take an example, sending a 1 500 B packet in 0.3 ms RTT needs 40 Mbit/s bandwidth. Although smaller increase mount a also means smaller traffic burst, a lot of tinny packets would be generated and link utilization would be affected.

3) Determining multiplicative factor b

DATCP adjusts the window decrease factor b to provide throughput differentiation. Suppose there are two flows competing for a bottleneck link. The maximum congestion window sizes of the two flows are W_1 and W_2 respectively. When congestion is detected, two flows decrease their window by b_1 and b_2 respectively, as illustrated in Fig. 7.

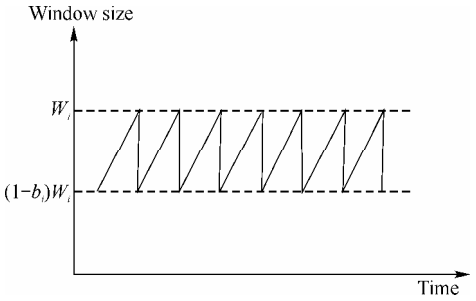


Fig. 7 The window size of flows

The ratio of two flows throughput is the ratio of average congestion window size.

$$\frac{R_{\text{flow1}}}{R_{\text{flow2}}}=\frac{\left[1+\left(1-b_1\right)\right]W_1}{\left[1+\left(1-b_2\right)\right]W_2} \tag{1}$$

We assume that two flows get proportional throughput differentiation. Let P_1, P_2 be the two flows precedence. So,

$$\frac{P_1}{P_2}=\frac{R_{\text{flow1}}}{R_{\text{flow2}}} \tag{2}$$

Assume that two flows have identical RTTs T_{rtt},

$$1=\frac{\left[1-\left(1-b_1\right)\right]W_1}{T_{\text{rtt}}} \tag{3}$$

$$1=\frac{\left[1-\left(1-b_2\right)\right]W_2}{T_{\text{rtt}}} \tag{4}$$

We can get,

$$\frac{P_1}{P_2}=\frac{\left(2-b_1\right)b_2}{\left(2-b_2\right)b_1} \tag{5}$$

For given P_1 we set b_1, and then for given P_2 we can get b_2 according to the Eq. (5). Usually we set the precedence of standard TCP to 1 and b to 0.5.

3.3 Flow quenching

When network load is too heavy, capacity available to flows is not enough to allow all the flows complete before deadlines, so some low precedence flows should be quenched to make the remaining high precedence flows meet the deadlines.

Algorithm 2 illustrates the scheme of flow quenching. If actual throughput is smaller than bandwidth share, DATCP tries to increase urgency to get higher R_{avg}. $\Gamma_{\text{profit}}\leqslant 0$ means that flow cannot get more bandwidth share. If flows cannot meet deadlines with the current R_{avg} and $\Gamma_{\text{profit}}\leqslant 0$, flows should be quenched. In order to mitigate the throughput instability, DATCP quenches flows when $\Gamma_{\text{profit}}\leqslant 0$ for several times continuously. Moreover, flows should be quenched with a certain probability. This could avoid flow quenching synchronization. Low precedence flows are quenched with a bigger probability. This makes sure as many high precedence flows as possible meet the deadlines. We set the quenching probability as p_{quench}. The quenching probability is related to the flow precedence. DATCP should allow more flow with high precedence complete first.

It is should be noted that flows could just be suspended for a period of time rather than be killed. When network load is light, the flows can continue to transmit data. However, whether flows are killed or just suspended in flow quenching depends on the application. For interactive web applications, overtime flows should be quenched because flows missing the critical deadlines are useless to applications. But for MapReduce applications, the deadlines are not critical and the computing tasks need all the flows to complete, so overtime flows should be suspended.

Algorithm 2 Flow quenching

Description: low precedence flows are quenched with a bigger probability. This can make higher precedence flows

meet their deadlines.

Input:

flow size V_{total}, flow deadline $T_{deadline}$, flow importance I.

ε: the flag identifies flows are killed or just suspended

Variable Definition:

P: flow precedence.

P_{max}: the maximum flow precedence.

R_{avg}: present actual throughput.

R'_{avg}: last actual throughput

Γ_{profit}: the increase proportion of R_{avg} after the added urgency.

$R_{desired}$: flow bandwidth demand

θ: the continuous times $\Gamma_{profit} \leqslant 0$ after increase urgency.

p_{quench}: the quenching probability.

Θ: the maximum θ.

Initialization:

$\theta = 0$;

Algorithm:

When receive every window of data (roughly one RTT)

if (urgency is increased last RTT) then

$\Gamma_{profit} = R_{avg} - R'_{avg}$;

if ($\Gamma_{profit} \leqslant 0$) then

$\theta = \theta + 1$;

if ($\theta \geqslant \Theta$) then

$p_{quench} = 1 - P / P_{max}$;

flow is quenched with p_{quench} and whether is killed according to ε;

$\theta = 0$;

end if

end if

else then

$\theta = 0$;

end if.

3.4 Discussion

Although above we let DATCP use ECN to detect network congestion, DATCP can also be deployed as pure end-based protocol and without any middle network support. However, since the bandwidth-delay product is small in data center networks, congestion window size of each flow is usually small. If not using ECN, network congestion is usually detected by packet losses rather than duplicated ACKs [19]. This would take longer time to detect congestion and affect the performance of DATCP. Moreover, the idea of DATCP is also suitable to rate-based TCP. The result of DATCP for rate-based congestion control is more accurate than window-based congestion control because it is difficult to maintain a rate in window-based congestion control especially in data center networks.

4 Evaluation

In this section, we perform extensive simulations with the network simulator NS2 (http://www.isi.edu/nsnam/ns/) to evaluate the performance of DATCP. A MapReduce scenario is also considered to introduce how to schedule flows using DATCP. The network model used in the simulations is that multiple servers are connected to a top-of-rack switch. The switch supports RED/ECN, an active queue manage scheme already available in current commodity switches.

4.1 Simulation parameters setting

We set simulation parameters as shown in Table 1.

Table 1 The simulation parameters

Parameters	Value
Capacity of links	1 Gbit/s
Propagation delay of links	75 μs
Buffer size of each switch port	200 KB
Max_p of RED	1.0
Min_threshold of RED	80 KB
Max_threshold of RED	120 KB
Queue weight of RED	1.0
MinRTO	10 ms
Packet size	1 500 B
Maximum window	50 packets

The capacities of links are set to 1 Gbit/s and the buffer size of each switch port is set to 200 KB. Both values are typical in current data centers. Since the RTT of data center network is very small, usually only hundreds of μs, we set the minimum retransmission timeout (RTO) to 10 ms instead of the default 200 ms. 10 ms is a typical value used in data center networks [6]. We set the queue weight of RED to 1.0 and then packets marking are based on the instantaneous queue length. Therefore the sender can detect the congestion quickly.

In addition, the precedence of standard TCP is set to 1, and $a = 1$, $b = 0.5$. According to Eq. (5), we can get the window decrease factor b for other precedence flows. For all flows, we set the window increase factor a to 1. This could limit the traffic burst. Moreover, we set the average throughput calculate factor a to 0.1.

4.2 Basic of DATCP algorithm

We will analyze the basic of DATCP algorithm in this subsection.

Deadline-aware: the deadline-aware feature of DATCP

under different deadlines is evaluated. We set up following simulation scenario: there are six flows competing for a 1 Gbit/s link, and three of them have deadlines and the others have no deadline. The size of the flows with a deadline is 100 MB and others have infinite data to send. The throughput of flows is computed at 0.1 s interval. We set the parameter φ to 0, i.e., only urgency is considered in this subsection. The deadlines are set to 4 s, 3 s respectively and the scenario of no deadline is also presented. The throughput of the flow under different deadline is shown in Fig. 8. As we can see from the figure, DATCP can meet the deadlines effectively. The throughput of background flows are also shown in the figure. The throughput of flows within the same urgency is almost identical so the fairness of DATCP is good. This holds for flows with no deadline.

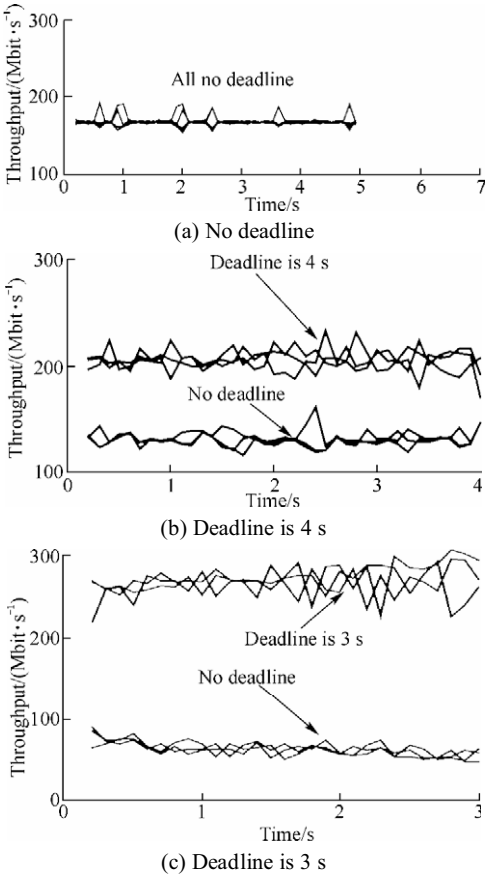


Fig. 8 The throughput of flows with different deadlines

Convergence: to show that DATCP flows quickly converge to their weighted fair rate, we perform a simulation similar to above. G is set to 0.05. We first start three deadline-agnostic flows, and then we sequentially start three deadline-aware flows, spaced by 1 s. The deadlines are all set to 4s. The time series depicting the overall flow throughput is shown in Fig. 9. As DATCP flows come and go, they quickly converge to their weighted fair rate.

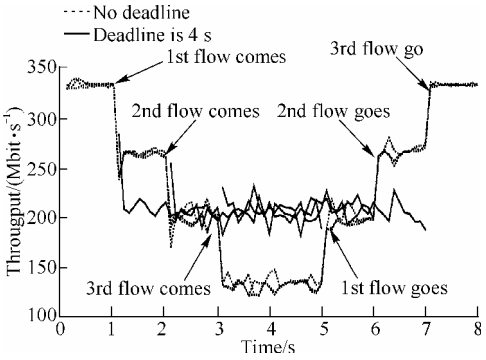


Fig. 9 The convergence test of DATCP

Combining importance and urgency: next the effects of importance are evaluated. We still use the simulation scenario above except importance of one background flow is set to 1, 2, 3 respectively. The precedence is calculated by both the importance and urgency of the flows and the parameter φ is set to 0.5. As we can see from the Fig. 10, the background flow with higher importance got more link share. This can avoid starving important but not-urgent flows.

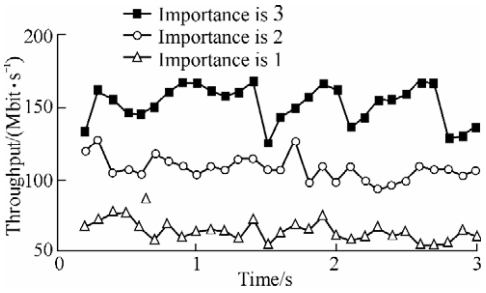
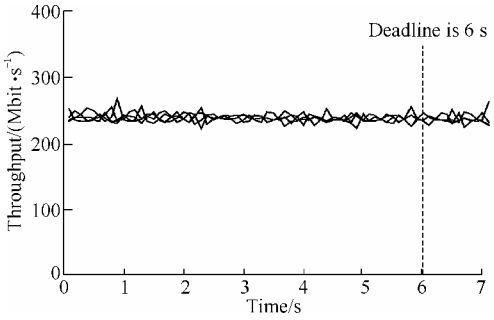


Fig. 10 The throughput of flows with different importance

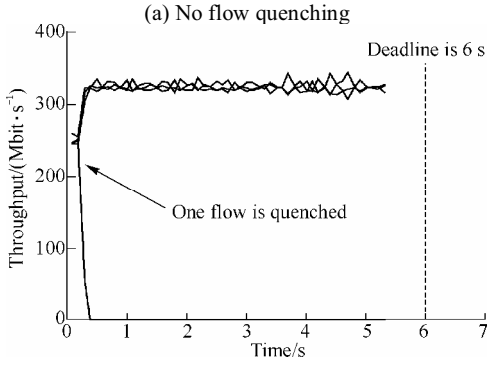
Flow quenching: we used another simulation scenario. There are 4 flows sized 200 MB with same deadline and same importance. The deadline is set to 6 s. If no flow is quenched, and then all flow cannot finish before deadline, as illustrated in the Fig. 11(a). Fig. 11(b) plots the result that DATCP quenched one flow so that other 3 flows meet the deadline.

Conventional TCP with priority: TCP with priority can give some flow more bandwidth share. But the share is proportional. We simulated 3 TCP flows with priority is 3 and some background flows share a 1 Gbit/s bottleneck link. The number of background flow is set to 3 and 6 respectively; the throughput of TCP flows is shown in Fig. 12. As we can see, same priority leads to different

completion time under different background flow number. That is, TCP with priority could not make sure flows meet deadlines.

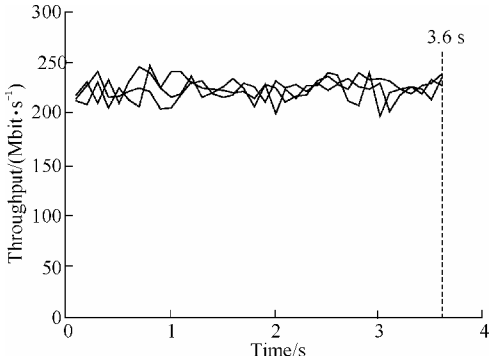


(a) No flow quenching

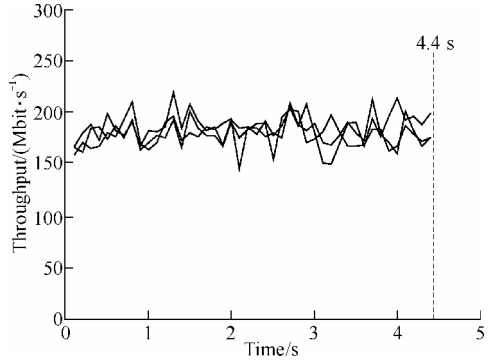


(b) With flow quenching

Fig. 11 The effect of flow quenching



(a) 3 background flows



(b) 6 background flows

Fig. 12 Flow throughput of conventional TCP with priority (priority is 3)

4.3 Generated traffic scenario

In this scenario, we evaluate the performance of our design under the generated traffic. Three deadline-agnostic background flows which have infinite data to send are first generated. We then generate three deadline-ware flows every ten seconds. The flow size is uniformly distributed on 10 MB~100 MB. The deadlines are also on uniform distribution and three kinds of deadlines are evaluated: lax, moderate and tight. We compare DATCP with TCP and D^3 in the simulation. The ratio of flows meeting deadlines is shown in Fig. 13. As we can see, DATCP outperforms TCP under all three kinds of deadlines. Though D^3 has similar results with DATCP, it needs new switches and new protocols. Unlike D^3, DATCP only needs minor modification to existing TCP and makes use of a feature already available in current commodity data center switches.

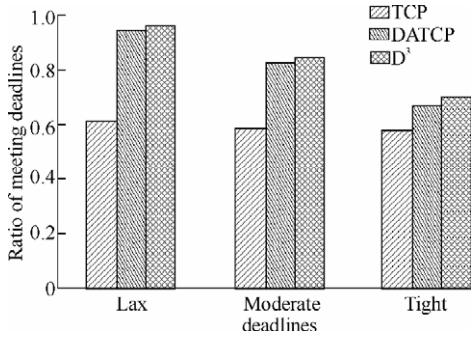


Fig. 13 The generated traffic scenario

4.4 MapReduce scenario

We consider a simple MapReduce scenario to show that DATCP could be used as a flow scheduling method. Suppose that there are two jobs ongoing. Job A is made up of four maps and a reduce while job B consists of two maps and a reduce. Jobs are scheduled by fair share Scheduler. Both job A and job B have two maps completed. Job B only has a reduce left but job A has two maps and a reduce left. Reduce tasks are going to copy the output of the maps. We suppose that the two reduce tasks are located in the same server. So there are four flows sharing the downlink of the server. Obviously, the two flows of job B are more urgent than the other two flows of job A because job B will finish once the two flows complete. After job B completes, the resource will be released and can be used for other jobs. Then the utilization of MapReduce cluster will be increased. Therefore, flows of job B should be

scheduled to complete as quickly as possible.

Next we will show how to schedule flows using DATCP. There are four parameters needed: flow size, flow deadline, flow importance and whether flows are killed or just suspended in flow quenching. We set the size of every map output to 100 MB and the length of all tasks to 5 s. In order to let flows of job B complete quickly, we set the deadline of flows of job B to 2 s and flows of job A to 5 s which is the same as the task length. All flows have no importance and flows are suspended in flow quenching. The completion times of two jobs with DATCP and TCP are evaluated. As shown in the Fig. 14, the completion time of job B with TCP is 3.2 s and with DATCP is 2 s while the completion of job A is not affected.

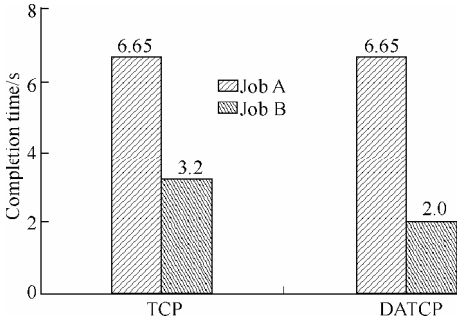


Fig. 14 The MapReduce scenario

From above we can see that DATCP can be used for flow scheduling and a good flow scheduling algorithm can improve the application performance greatly. But the strategy flow scheduling is out of the scope of this paper and we will study it in our future work.

5 Conclusions

In this paper, we present DATCP, a new transport protocol for providing deadline-aware transmission service for the commoditized data centers. DATCP combines flow urgency and importance to calculate precedence. And flow urgency is dynamically adjusted according to the gap between desired rate and actual throughput. DATCP only needs minor modification to existing TCP and makes use of ECN, a feature already available in current commodity switches. We evaluate DATCP via extensive simulations. Our results suggest that DATCP can make flows meet deadlines effectively without starving the important but not-urgency flows.

DATCP provides a flexible method to schedule flows. The strategy of flow scheduling is related to specific application and a efficient flow scheduling algorithm can improve the application performance greatly. We will study the flow scheduling using DATCP in future works.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (61002011), the Open Fund of the State Key Laboratory of Software Development Environment (SKLSDE-2009KF-2-08), the National Basic Research Program of China (2009CB320505), the Hi-Tech Research and Development Program of China (2011AA01A102).

References

1. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Proceedings of the 6th USENIX Symposium on Operating System Design and Implementation (OSDI'04), Dec 6–8, 2004, San Francisco, CA, USA. Berkeley, CA, USA: USENIX Association, 2004: 13p
2. Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild. Proceedings of the 10th Internet Measurement Conference (IMC'10), Nov 1–3, 2010, Melbourne, Australia. New York, NY, USA: ACM, 2010: 267–280
3. Wilson C, Ballani H, Karagiannis T, et al. Better never than late: meeting deadlines in datacenter networks. Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM'11), Aug 15–19, 2011, Toronto, Canada. New York, NY, USA: ACM, 2011: 50–61
4. Shieh A, Kandula S, Greenberg A, et al. Sharing the data center network. Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11), Mar 30–Apr 1, 2011, Boston, MA, USA. Berkeley, CA, USA: USENIX Association, 2011: 23
5. Benson T, Anand A, Akella A, et al. Understanding datacenter traffic characteristics. Proceedings of the 1st ACM Workshop on Research on enterprise networking (WREN'09), Aug 21, 2009, Barcelona, Spain. New York, NY, USA, 2009: 65–72
6. Alizadeh M, Greenberg A, Maltz D, et al. Data center TCP (DCTCP). Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM'10), Aug 30–Sep 3, 2010, New Delhi, India. New York, NY, USA, 2010: 63–74
7. Blake S, Black D, Carlson M, et al. An architecture for differentiated services. IETF RFC 2475.1998
8. Nandagopal T, Lee K W, Li J R, et al. Scalable service differentiation using purely end-to-end mechanisms: features and limitations. Proceedings of the 8th IEEE International Workshop on Quality of Service (IWQOS'00), Jun 5–7, 2000, Pittsburgh, PA, USA. Los Alamitos, CA, USA: IEEE Computer Society, 2000: 813–833
9. Covey R S. The 7 habits of highly effective people. Revised edition. Mankato, MN, USA: Free Press, 2004
10. Ramakrishnan K, Floyd S, Black D. The addition of explicit congestion notification (ECN) to IP. IETF RFC 3168. 2001.
11. White T. Hadoop: the definitive guide. 2nd edition. Sebastopol , CA, USA: O'Reilly Media / Yahoo Press, 2010
12. Ghemawat S, Gobioff H, Leung S T. The Google file system. Proceedings of the 19th ACM SIGOPS Symposium on Operating Systems Principles (SOSP'03), Oct 19–22, 2003, Bolton Landing, NY, USA. New York, NY, USA: ACM, 2003: 29–43
13. Zaharia M, Borthakur D, Sarma J S, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. Proceedings of the 5th European conference on Computer systems (EuroSys'10), Apr 13–16, 2010, Paris, France. New York, NY, USA: ACM, 2010: 265–278

14. Vasudevan V, Phanishayee A, Shah H, et al. Safe and effective fine-grained TCP retransmissions for datacenter communication. Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM'09), Aug 17–21, 2009, Barcelona, Spain. New York, NY, USA: ACM, 2009: 303–314
15. Chen Y P, Griffith R, Liu J D, et al. Understanding TCP incast throughput collapse in datacenter networks. Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN'09), Aug 21, 2009, Barcelona, Spain. New York, NY, USA: ACM, 2009: 10p
16. Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM'08), Aug 17–22, 2008, Seattle, WA, USA. New York, NY, USA: ACM, 2008: 63–74
17. Greenberg A, Lahiri P, Maltz D A, et al. Towards a next generation data center architecture: scalability and commoditization. Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO'08), Aug 17–22, 2008, Seattle, WA, USA. New York, NY, USA: ACM, 2008: 57–62
18. Guo C X, Wu H T, Tan K, et al. Dcell: a scalable and fault-tolerant network structure for data centers. Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM'08), Aug 17–22, 2008, Seattle, WA, USA. New York, NY, USA: ACM, 2008: 75–86
19. Zhang P, Wang H B, Cheng S D. Shrinking MTU to improve the TCP fairness in data center networks. Proceedings of the 3rd IEEE International Conference on Communication Software and Networks (ICCSN'11), May 27–29, 2011, Xi'an, China. Piscataway, NJ, USA: IEEE, 2011: 106–111

(Editor: WANG Xu-ying)

From p. 28

5. Saad W, Zhu H, Debbah M. Coalitional game theory for communication networks. IEEE Signal Processing Magazine , 2009, 26(5): 77–97
6. Zhu H, Zhu J. Fair multiuser channel allocation for OFDMA networks using Nash bargaining solutions and coalitions. IEEE Transactions on Wireless Communications, 2005, 53(8): 1366–1376
7. Hering P J J, van der Laan G, Talman D. Cooperative game in graph structure. No 2000-90. Tilburg, Netherlands: Center for Economic Research, Tilburg University, 2000
8. Nguyen T D, Han Y. A proportional fairness algorithm with QoS provision in downlink OFDMA systems. IEEE Communications, 2006, 10(11): 760–762

(Editor: WANG Xu-ying)