Controlling Network Latency in Mixed Hadoop Clusters: Do We Need Active Queue Management?

Renan Fischer e Silva, Paul M. Carpenter
Barcelona Supercomputing Center—*Centro Nacional de Supercomputación* (BSC–CNS)
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
Email: {renan.fischeresilva,paul.carpenter}@bsc.es

***Abstract*—With the advent of big data, data center applications are processing vast amounts of unstructured and semi-structured data, in parallel on large clusters, across hundreds to thousands of nodes. The highest performance for these batch big data workloads is achieved using expensive network equipment with large buffers, which accommodate bursts in network traffic and allocate bandwidth fairly even when the network is congested. Throughput-sensitive big data applications are, however, often executed in the same data center as latency-sensitive workloads. For both workloads to be supported well, the network must provide both maximum throughput and low latency. Progress has been made in this direction, as modern network switches support Active Queue Management (AQM) and Explicit Congestion Notifications (ECN), both mechanisms to control the level of queue occupancy, reducing the total network latency.**

This paper is the first study of the effect of Active Queue Management on both throughput and latency, in the context of Hadoop and the MapReduce programming model. We give a quantitative comparison of four different approaches for controlling buffer occupancy and latency: RED and CoDel, both standalone and also combined with ECN and DCTCP network protocol, and identify the AQM configurations that maintain Hadoop execution time gains from larger buffers within 5%, while reducing network packet latency caused by bufferbloat by up to 85%. Finally, we provide recommendations to administrators of Hadoop clusters as to how to improve latency without degrading the throughput of batch big data workloads.

***Keywords*—Data Center, MapReduce, Hadoop, DCTCP, ECN**

I. Introduction

With the advent of Big Data, data center applications are processing multi-terabyte datasets, in parallel on large clusters, across hundreds to thousands of nodes. A typical framework for big data processing is the MapReduce programming model [1] and its open-source implementation Apache Hadoop [2]. Big data workloads based on Hadoop or similar frameworks generate significant communication among servers within the same data center. In particular, as explained below, the shuffle phase of MapReduce involves an all-to-all communication, which presents a stressful load on the network.

For these workloads, the highest performance is achieved using expensive network equipment with large buffers, which are better able to accommodate congestion and network traffic bursts. Large buffer switches, however, suffer from the bufferbloat phenomenon, in which TCP (greedily) makes full use of the available buffers, even when maximum performance can be achieved using much less buffering. Bufferbloat has been found to cause excessive packet delays within data centers [3]. Nevertheless, it is reasonable to expect that bufferbloat would have little direct effect on Hadoop, since its communication is dominated by long network flows.

Throughput-sensitive big data applications are, however, often executed in the same data center as other workloads that directly interact with external users, and these workloads *are* sensitive to network latency. In this case, the network is shared between both classes of application, so it should therefore provide not only the maximum possible throughput, but also the lowest possible latency.

This paper is, to the best of our knowledge, the first to analyze mechanisms to control packet delay in a Hadoop cluster. We study Active Queue Management (AQM) and Explicit Congestion Notifications (ECN), both of which are supported in modern network switches. ECN, in particular, has received major attention in recent years. For instance, in mid-2015, Apple enabled it by default on all its operational systems, in order to improve the customer experience. We perform a quantitative evaluation of the tradeoff between throughput and latency, for four different approaches for controlling buffer occupancy and latency: Random Early Detection (RED) and CoDel queues, both standalone and also combined with ECN and Data Center TCP (DCTCP). This is done in the context of MapReduce and Apache Hadoop [2], which present a specific traffic pattern in the shuffle phase.

Our work provides recommendations to administrators of Hadoop clusters. We show experimental results at the network level, in terms of network throughput and packet latency. More importantly, we also show the impact on Hadoop job execution time. Previous analysis suggested that CoDel and other techniques that reduce buffer occupancy would also translate to better performance during Hadoop's all-to-all communication phase [4]. We find that TCP already functions well, so Hadoop execution time is not improved by such techniques. Moreover, in some cases a poorly-chosen AQM configuration increases the execution time by an unacceptable 20%. We do, however, identify good AQM configurations that are able to maintain Hadoop execution time gains from larger buffer to within 5%, while reducing packet latency caused by bufferbloat by 85%.

In short, our contributions are threefold:

1) A study of mechanisms that can be employed on Hadoop clusters to control packet delay.
2) A quantitative evaluation of the tradeoff between

throughput and latency for RED and CoDel queues, both standalone and also combined with ECN and DCTCP.
3) Recommendations to cluster administrators to improve latency without degrading throughput.

The rest of the paper is organized as follows: Section II provides the background about MapReduce and Hadoop, the main TCP/IP problems encountered in modern data centers, Active Queue Management, Explicit Congestion Notifications and DCTCP. Section III compares our approach with related work. Section IV presents the methodology, quantitative results and analysis, from which Section V distills the most important recommendations. Finally, Section VI concludes the paper.

II. Background

This section describes the most important problems encountered in modern data center networks, and it describes the main solutions, both current practice and state-of-the-art. It also summarizes the Hadoop framework and MapReduce programming model.

A. TCP in Modern Data Centers

Recent studies show that 97% of the traffic in current data centers is carried by IP packets, either as TCP or UDP segments depending on the workload [5]. In 2010, Microsoft Research published a study of 150 TB of network traces that showed that, for their data center, TCP segments made up more than 99% of their internal traffic [6].

TCP was initially designed for Wide Area Networks (WANs) [7], and certain aspects of its design, such as the minimum Retransmission Timeout (RTO) of $200\,\mathrm{ms}$ are better suited to WANs than data center LANs. Problems that arise in such a low-latency environment include (a) *TCP Incast* [7], a dramatic loss in throughput for many-to-one communication patterns, where congestion leads to packet loss, (b) *TCP Outcast* [8], where (surprisingly) the throughput to a congested node may be much lower from nearby nodes than from more distance ones, and (c) *Bufferbloat* [3], where congestion causes excessive packet buffering, leading to high and variable latency.

There are two situations where network equipment most benefit from larger buffers. Firstly, in upper-layer devices, at the aggregation and core layers, when bursty traffic on multiple incoming links is redirected to the same outgoing port, the switch will have to queue the packets before transmitting them. Secondly, in the access layer; i.e. in the Top-of-Rack (ToR) switches, incoming traffic going to the server nodes may arrive on a link that has higher bandwidth than the link to the server; e.g. packets arrive at 10GbE but must be transmitted at 1GbE.

In an ideal case, data center networks should accommodate long flows, which require high throughput, and also allow short flows to have low latency in scenarios where buffers are heavily used. Doing so may not be possible on some workloads, and trade-offs have to be considered to adjust each case to the best possibility.

B. Controlling latency and buffer occupancy

*1) **Active Queue Management**:* AQM schemes have been proposed to manage buffer occupancy and keep the average latency of the buffers below a determined threshold. The goal is eliminate the problem found on DropTail queues that tend to penalize bursty flows and also introduce high latency into the network. Bursty flows are penalized when TCP global synchronization happens. Instead, an AQM scheme aims to keep the delay controlled by providing feedback to the end points through appropriately dropping packets. Another goal of these smarter queues is to support the use of Explicit Congestion Notifications, found on TCP network protocol, that allow end points to react before congestion happens. On this paper we selected two AQMs to compare, Random Early Detection (RED) [9] and Controlled Delay (CoDel) [10].

Random Early Detection (RED) was proposed in 1993 [9] and since then, it has been widely studied and adopted. Implementations of RED are found on Linux, Solaris, and FreeBSD [11]. It is also implemented by network equipment vendors including Cisco [12] and Juniper Networks [13]. RED uses configurable thresholds to decide when to mark packets when combined with ECN, and drops packets based on a probability that grows with the queue occupancy.

Controlled Delay (CoDel) was proposed in 2012 and since then, it has gained more attention. Its usage is recommended by the Bufferbloat initiative [3] [14]. It promises to be easier to configure than RED, which has several parameters and variants to be configured. CoDel claims it has no parameters to set at all, but still, the user needs to configure the target *delay*, which is the tolerable delay per-packet when queued until it is transmitted, and the *interval* how often the delay per packet of transmitted packets within the interval is evaluated. If any packet has a delay grater that the target, the interval is shortened, otherwise it is reset at the end of its cycle.

*2) **Explicit Congestion Notifications**:* ECN are helpful to indicate a pre-state of congestion on the network and allow senders to proactively react before it happens. Instead of waiting for the buffer to drop packets and trigger the fast retransmit state of TCP, which involves waiting for the RTO timeout of TCP (typically on a range from $200\,\mathrm{ms}$ to $1\,\mathrm{ms}$) and then reset to the slow start state of TCP, the sender reduces its congestion window by the number of marked packets, alleviating the pressure under the buffer that signalized the congestion which helps to reduce latency and specially jitter.

On data center networks, using such feature may reduce throughput of applications while keeping the latency and buffer occupancy low, which may not be desired on frameworks with high throughput requirements. As alternate solutions, DCTCP has been proposed which involves some modifications on the TCP network protocol to specifically fit data center network requirements: high throughput and small latency.

*3) **Data Center TCP**:* DCTCP is an extension to the TCP network protocol proposed by Microsoft Research Center as an alternate solution to specifically reduce the latency on data center networks without affecting throughput. On its

evaluation using commodity switches, DCTCP was able to deliver even better performance than TCP itself. Currently network equipment manufactures are iterating into their lineup and recommending the usage of deep buffer switches for Big Data frameworks, specially Hadoop, which demands more analyses of how DCTCP performs on such type of workloads and using these new network equipments.

C. MapReduce and Hadoop

In 2004 Google introduced the MapReduce programming model for reliable fault-tolerant processing of huge data sets on large commodity clusters [1]. The programmer is given a data abstraction in terms of *map* and *reduce* operations on key/value pairs, and the framework takes care of the implementation details including automatic parallelization, task scheduling with data locality, monitoring, redundant distributed data storage, and re-executing failed tasks. The input and output data for the MapReduce jobs are stored on a distributed filesystem known as Apache HDFS (Hadoop Distributed File System), which uses disks attached to the same nodes used for computation. The job scheduler tries to schedule tasks to run on nodes where data is already present, resulting in high data locality [2].

The MapReduce framework first splits the input data set into independent chunks, which are processed in parallel by the map tasks. It then sorts the combined outputs from the maps, in the so-called shuffle stage, passing the sorted data to the reduce tasks. This stage involves an all-to-all communication among nodes.

Several open-source MapReduce frameworks have been developed over the years, with by far the most popular one being Apache Hadoop [2]. The MapReduce programming model [1] targets commodity hardware and network equipment using the TCP/IP protocol. But with the advent and revolution of the Internet, more modern equipments are offered, which include bigger buffers to handle bursty network communication, and therefore, yielding the amount of information generated by the Big Data era.

III. Related Work

Heterogeneous clusters are becoming relatively more common on modern data centers as an attempt to reduce cost and avail the built infrastructure. As an example, Apache Myriad is a open source project that enables Apache Hadoop to run side-by-side with other type of applications, dynamically sharing cluster resources [15].

Several vendors are positioning themselves for the Big Data market and have introduced network equipment with the promise of increased performance for Big Data applications. Arista Networks is marketing their new 7048T, 7280E and 7500R switch series with large buffers as recommended solutions for optimum performance for Hadoop [16]. Cisco published a study that found that network latency has little impact on job completion time, among other factors such as availability and resiliency, burst handling queuing, oversubscription ratio and data node network speed [17]. In the same study, burst handling queuing capability was considered as the second most important factor that affects job completion time.

A lot of attention so far has focused on RED, which is widely used as a baseline for the evaluation of new AQMs. Also it has based versions implemented by network vendors as Cisco [12] and Juniper Networks [13].

The CoDel IETF draft [4] suggests that CoDel would be useful in environments other than the normal Internet, including in data center switches, particularly for MapReduce clusters. As described, a CoDel queue tuned for such an environment promises to minimize packet drops, while keeping throughput high and latency low.

DCTCP was presented before CoDel, and it used a variant of the RED queue, with recommended values of the minimum and maximum thresholds both equal to 70 packets. When DCTCP was compared to RED, it was suggested that although RED combined with ECN would dramatically reduce network throughput in data centers, DCTCP would maintain high throughput (while providing low queue occupancy and low delay). The explanation was that, instead of dramatically cutting the TCP congestion window by the number of marked acknowledges, DCTCP reduces the window gently to maintain high throughput.

Wu et al. presented a comprehensive study on the tuning of ECN for data center networks, which they described as ECN* [18]. Their new approach performs as well as DCTCP, but it requires modifications to the ECN marking scheme in the network switches (the transport protocol and end points are not modified). They proposed marking packets when they leave the network queue ("dequeue marking") instead of when they enter the network queue (classical "enqueue marking", as used in RED). Dequeue marking seems to deliver similar results to DCTCP's gentle congestion control. This proposal also came before the introduction of CoDel, which also marks packets when dequeue occurs, suggesting that CoDel would also deliver better performance than RED. Big Data workloads or deep buffer switches were not considered, missing a more profound analysis of the scenarios that typically present bufferbloat phenomena. When specifically compared using NS–2 simulations with synthetically generated traffic with a Pareto distribution, CoDel delivered lower latency than RED but the latter was still considered as a good candidate for an AQM [19].

Incast was shown to have little significant impact on the performance of Hadoop, assuming a well-tuned Hadoop cluster. Incast is characterized by many-to-one communication where the buffer on the receiver pipe is heavily pressured so packets are lost. But its effect is specially devastating on partition/aggregation workloads which perform small queries, because the default RTO (Retransmission Timeout) penalty of $200\,\mathrm{ms}$ represents a big overhead on the overall performance. Hadoop presents many-to-one network communication in its shuffle phase, but on a well-configured system there is not significant overhead caused by incast. Also, buffers of network equipments are highly used, especially during the shuffle and write phases [20]. As we demonstrate in our results, Hadoop

TABLE I
SIMULATED ENVIRONMENT

Category	Parameter	Value
Simulated hardware		
System	Number nodes	160
	Number racks	4
Node	CPU	Intel Xeon 2.5 GHz L5420
	Number cores	2
	Number processors	2
Network	Each node	*1GbE:* 1 —
	Each leaf switch	*1GbE: 40 10GbE:* 2
	Each spine switch	— *10GbE:* 4
Buffers	Commodity switches	200 packets - max 133 KB per port
	Expensive switches	2000 packets - max 1.33 MB per port

is highly throughput-sensitive. Therefore, in contrast to the recommendation to use a smaller minRTO in data centers, specially when using big buffer equipments which tolerate more bursty communication, reducing the RTO from $200\,\mathrm{ms}$ to $1\,\mathrm{ms}$ can impact on a fake RTO penalty. For example, an in-flight packet could still be queued in a buffer and since on bufferbloat scenarios the average latency per packet can be higher than such small RTO, TCP would trigger its timeout even if the packet is not dropped. For this reason our simulations use the default TCP minRTO of $200\,\mathrm{ms}$ and the overall results can be verified on the next section.

IV. RESULTS

This section first describes the experimental methodology, and then presents the quantitative results, giving the runtime, throughput and latency for Hadoop using control delay mechanisms.

We evaluate control delay mechanisms as a function of the network topology, hardware configuration, and transport protocol, using the NS-2 packet-level network simulator [21]. This simulator has been extended with CoDel [22] and DCTCP [23] implementations and is driven by the MRPerf MapReduce simulator [24]. This methodology gives full visibility of the fine-grain details of the TCP/IP protocol in the data center environment. Extensive evaluation has been done with RED, CoDel and DCTCP using NS-2 which brings us to a fair comparison with our results.

A. Simulation Environment and Workload Characterization

The simulated hardware is shown in Table I. We simulate a 4-rack cluster with 40 nodes per rack, each node having the throughput of a two-core Xeon at $2.5\,\mathrm{GHz}$ and a single 1GbE link to the top-of-rack (ToR) switch.

The data center architecture selected for this work was the leaf-spine architecture [25] as seen in Figure 1. Leaf-spine architecture is the recommended architecture for warehouse scale computer. The topology brings spine and leafs switches similar to the access and aggregation switches found on the classical k-ary fat tree [26]. The advantage of leaf-spine when compared to the k-ary fat free is its full mesh connectivity

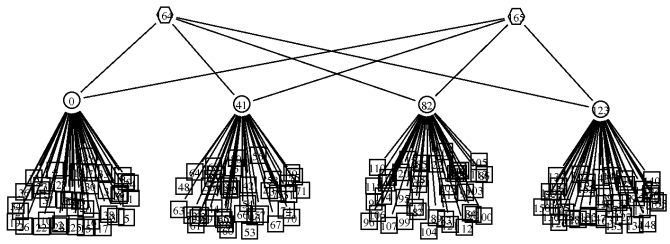

Fig. 1. Leaf-spine Cluster Topology

between leaf and spine switches. Each leaf switch, also known as top-of-rack (ToR) switch, is connected to the spine switch, regularly named aggregation switch on the fat tree model, using a single 10GbE link. The over-subscription ratio on the access layer is equal to 2:1. This matches cluster design recommendations that MapReduce clusters should be deployed with an over-subscription ratio not greater than 4:1 at the access layer [17] [27].

Is important to mention that the leaf-spine topology can also be categorized as a 2-level fat-tree topology, i.e. without the core layer, and it is not organized in pods. Since the focus of this work was to analyze MapReduce workloads in depth we decided to consider only the leaf-spine topology for this paper, which seems to be recommended for Hadoop, as seen in various references for cluster design [16] [28] [29]. Nevertheless, from the results obtained on this work we expect similar phenomena in many other distributed applications that are throughput sensitive and use either a traditional fat-tree topology or a leaf-spine topology.

On our cluster we used the multiPath option from NS-2 that simulates the equal cost multi-path routing through two equal cost routes. Equal Cost Multi Path (ECMP) feature is essential for a representative analysis of this cluster topology, which offers multiple routes and loaded over-subscription. Recent works as [30] show the benefit of using multipath TCP achieving improved network utilization and better reliability. Since multipath TCP is not yet commonly adopted in mainstream and we want to specifically investigate the impact of latency control mechanisms and DCTCP on Hadoop we decided to

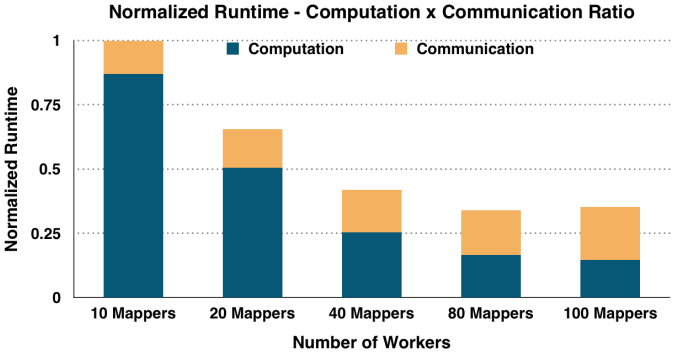

Fig. 2. Shuffle Characterization

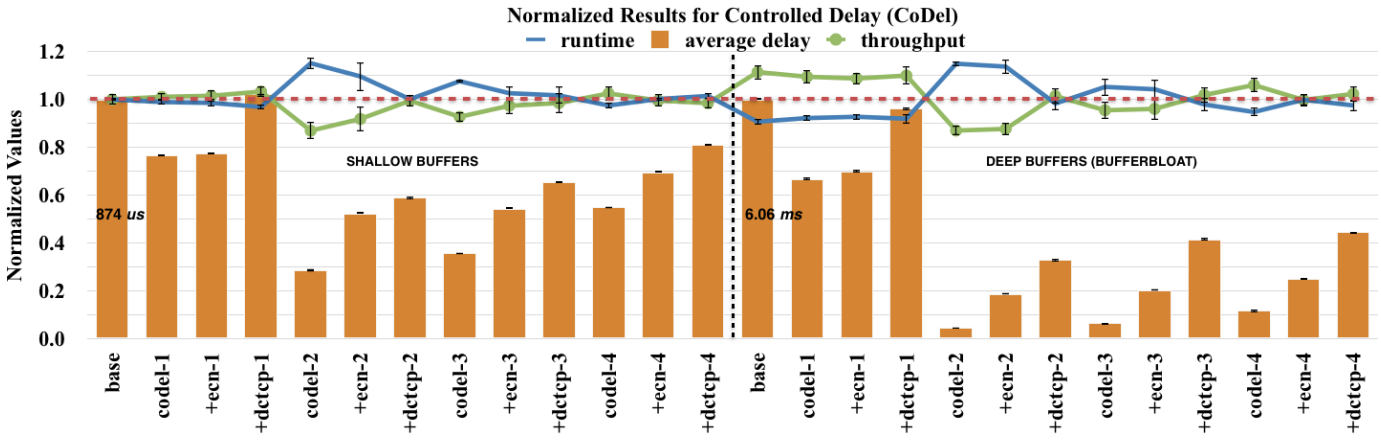

Fig. 3. Normalized Results for Controlled Delay Queue

TABLE II
CONTROLLED DELAY SETTINGS

	Target delay	Interval	Reference
Config 1	500 µs	20 ms	[14]
Config 2	300 µs	0.75 µs	[31]
Config 3	400 µs	1 ms	Proposal 1
Config 4	800 µs	1.5 ms	Proposal 2

bound the scope of this work using only ECMP feature.

We provide results for both shallow and deep buffer switches. Hadoop clusters often use inexpensive commodity switches, which have small (shallow) buffers. Small buffers can cause excessive packet loss over bursty communication and network equipment vendors are already promoting deeper buffered equipments for Hadoop clusters as seen in Section III. Related work comparing Active Queue Management tend to use packets instead of buffer size so we selected two different values for queue sizes as 200 packets or 2000 packets. For packets using maximum payload size of 1500 bytes translate our max capacity per port of 133 KBytes or 1.33 MBytes respectively.

Table I also shows the configuration of the simulated workloads. We reserve one node for Hadoop housekeeping, to serve as namenode and jobtracker, with the remaining nodes used as worker nodes for processing map and reduce tasks. On previous work [32] we simulated a smaller cluster with different type of workloads to understand the nuances found on a MapReduce cluster. We found small variability, most caused by the Hadoop scheduler. As for this work we expanded the size of our cluster, to limit the noise introduced by different scheduling decisions, we analyzed a single Terasort job configured to sort 6.4 GBytes (random elements) with 100 mappers. Terasort is a popular batch benchmark commonly used to measure MapReduce performance on a Hadoop cluster. In order to make it representative we characterized the shuffle phase as shown in Figure 2, to be consistent with a study of traces obtained at Facebook, which shows that most of the jobs were small and shuffle represented more than 50% of the execution time of a job [33]. Shuffle is considered the MapReduce phase that mostly stresses the network because its all-to-all communication between mappers and reducers.

We run the same Terasort job changing the number of map inputs and chunk size and fixing the number of reducers to the recommended, which is the number of workers in order to use the full system [34]. In a real cluster is recommended to use the factor 0.95 and leave a few nodes free in case of node failures. Since MRPerf does not simulate failing nodes we modeled the Terasort task divided by servers, each of them handling a proportion of the output from the map nodes. As our cluster has capacity of 2 mapslots and 2 reduce slots per node it give us relatively low utilization of not more than 40% which also matches with production tracers [33]. The communication, most of which is in the shuffle stage, is close to proportional to the workload size, but increases as the network becomes the bottleneck. Since the communication pattern is also repetitive, we can obtain representative figures using only one task as network and cluster utilization were proportionally designed based on available tracers.

We use three performance metrics: the *runtime* which is the total time needed to finish the Terasort workload, found to be inversely proportional to the effective throughput of the cluster; the *average throughput* per node and the *average end-to-end latency* per packet.

For throughput and runtime we used the same baseline for the whole set of results which is the DropTail queue for shallow buffers. This way we were able to compare improvements on runtime when using deep buffer equipments as promoted by new vendors.

For latency we considered two baselines. The naive assumption would be assuming congestion is something rare that happens only when there are peaks during the communication. On networks, congestion happens all the time, which means it is the steady state of the network. The goal of any transport protocol as TCP is to maximize the usage of the network.

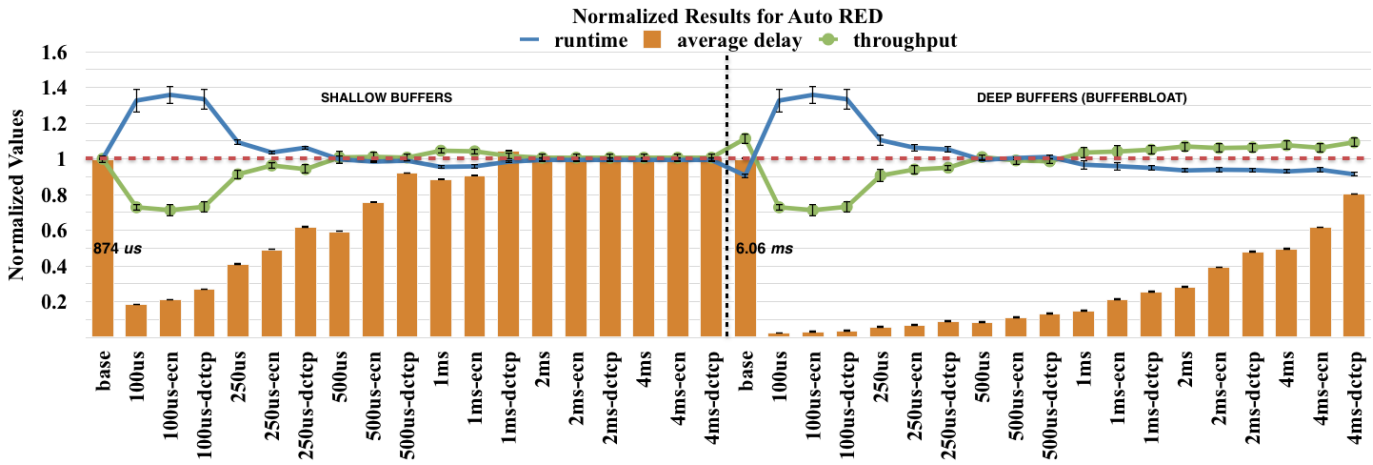

Fig. 4. Normalized Results for Auto RED Queue

TABLE III
AUTO RANDOM EARLY DETECTION SETTINGS

Target delay	1 GbE thresholds	10 GbE thresholds
100 µs	12.5 -37.5	125 - 375
250 µs	31.25 - 93.75	312.5 - 937
500 µs	62.5 - 187.5	625 - 1875
1 ms	125 - 375	NO AQM
2 ms	250 - 750	NO AQM
4 ms	500 - 1500	NO AQM

TCP, or any other congestion protocol, will be probing the network, trying to find how many packets the network can carry until it loses the packet and then back-off. On other words, TCP is always pushing the network into congestion and then backing-off. Using deeper buffer equipments will automatically increase the average delay per packet in order to obtain some gain on throughput and bursty tolerance. It is a trade-off between adding extra latency to the network while having more tolerance to bursty communication and also obtaining a higher throughput. For this reason we considered the DropTail queue of each set of simulations as the baseline for latency. When comparing the delay on shallow buffers, the smaller DropTail queue is the considered baseline. The same happens with the deep buffer set of results, which means the larger DropTail queue is the baseline for latency on bufferbloat scenarios.

B. Controlled Delay (CoDel)

For Controlled Delay queue we used the available implementation without any modification. CoDel is considered a parameterless queue, but the network administrator still has to provide two values, target and interval as described on Related Work (see Section III). For our evaluation we considered values found on previous publications and we also proposed new values as well to tolerate a bigger delay on bufferbloat scenarios. The used values on our simulations can be found in Table II. Still, we couldn't find many references specifically related to tune CoDel for data center networks. We can list the values found on Bufferbloat project [14] which recommends the use of 500 µs for target and 20 ms for interval. Another short paper [31] tested different range of values for CoDel recommending 300 µs for target and the much smaller 750 µs for interval. We also proposed 400 µs and 1 ms; and 800 µs and 1.5 ms for target and interval respectively as found in Table II. Results can be found in Figure 3.

Starting by the results with shallow buffers, CoDel was able to reduce the average delay per packet by half using our second proposal (config 4) with virtually no loss on throughput. When using the settings recommended by the Bufferbloat project (config 1) the delay dropped about 25% when combined with classical TCP. For all configurations, the smaller delay was achieved with standalone CoDel, followed by CoDel + ECN and at last, DCTCP showed the higher delay. On bufferbloat scenarios, we see that configurations that tolerate a higher delay also offer similar higher throughput. The settings recommended by Bufferbloat project were able to reduce the latency by 35% and still keep about 10% improvement on execution time. As the baseline for such scenarios has an average delay per packet of about 6 ms, for latency sensitive applications the configurations using CoDel stand alone were able to reduce latency to less than 700 µs with config 4 and even less than 400 µs using config 2. As a downside, specially configurations 2 and 3 were too aggressive reducing the congestion window and throughput was severely impacted.

C. Random Early Detection (RED)

The default implementation of Random Early Detection queue needed to be adapted for the high-speed networks found on data centers. RED is typically implemented using the average queue length for decisions of marking or drop of packets. We tried to use the average queue length on our experimentation but our results were too similar to the DropTail queue. Therefore, to allow simplification we do not

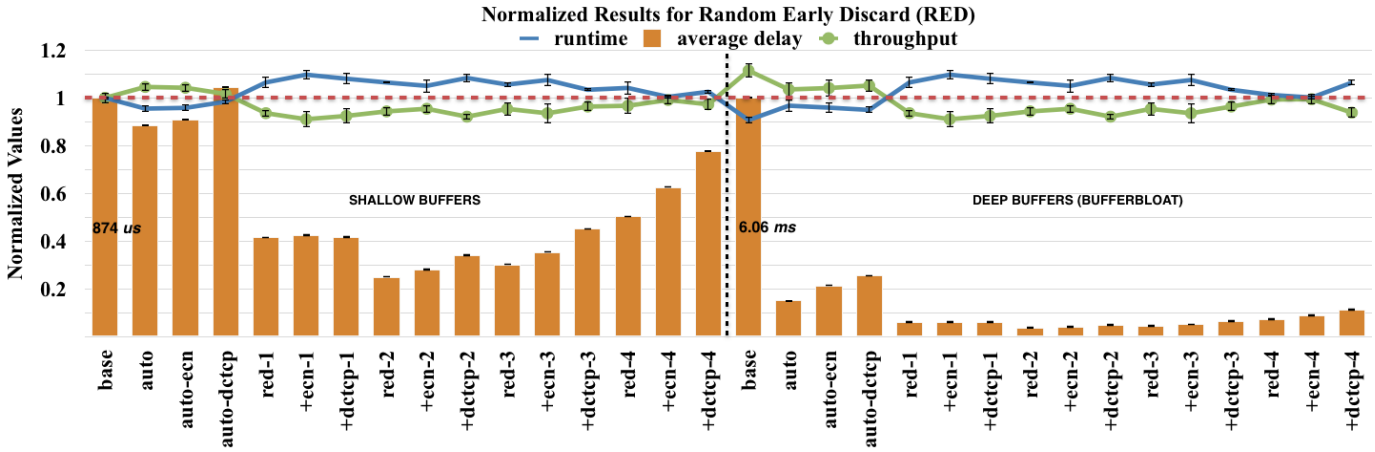

Fig. 5. Normalized Results for Random Early Detection Queue

TABLE IV
RANDOM EARLY DETECTION SETTINGS

	Min Th.	Max Th.	Reference
Config 1	70	70	[6]
Config 2	25	51	[31]
Config 3	25	75	Proposal 1
Config 4	50	150	Proposal 2

include them. Instead, we used the instant queue length on all our simulations with RED queue. We don't claim novel on this as previous works already demonstrated that instant queue length fits better for high speed networks as within the data center, we just confirmed and emphasize what was already demonstrated on previous evaluation [6].

RED offers an auto configuration setting that needs only one parameter: the target delay [11]. Such feature takes in consideration the speed of the network interface card and based on the "packet time constant", which is the maximum number of average sized packets that can be transmitted per second. If the network interface is fast enough and the target delay is also big enough, the thresholds will not be useful and the queue will behave as a DropTail queue. We couldn't find any references on literature so we performed a sweep using different values as found in Table III. Results are found in Figure 4.

We also considered different values found on previous publications and also proposed new values. With two new proposals, we also tried to match the performance of CoDel on the same base. Our considered values for RED are found in Table IV. Results for fixed settings are found in Figure 5.

It is easy to verify that RED auto configuration feature shows a clear tradeoff between latency and throughput. It seems to simplify the process of tuning the queue as we can clearly see that small values as $100\,\mu s$ will drop the performance by unacceptable 30%. As the values increase, we can see the average delay also increasing. To compare with the other values proposed on literature we decided to use $1\,ms$ as it was able to reduce the latency by 85% on bufferbloat scenario while still keeping a improvement of 5% on execution time.

When analyzing the fixed settings we can see that the first three configurations (config 1, config 2 and config 3) didn't perform well for RED, increasing the runtime in 10%. Configuration 4 was able to maintain the same execution time and still reduce latency by up to 90%. On the next subsection we compare the best values observed on CoDel and on RED.

D. CoDel x RED

Figure 6 compares the best settings of each queue side-by-side. When comparing CoDel and RED side-by-side we selected the two best values of each. The first pair are the settings that offer the best throughput but do not have a considerable cut on latency. The last pair are the settings which offer the best reduction on delay while still maintain some gain on throughput. We can see that even though CoDel and RED use values that are more or less at the same scale ($800\,\mu s$ vs. $1\,ms$), RED tend to achieve smaller latency than CoDel. On the other hand, CoDel was able to achieve the best runtime. One explanation for such difference is that RED is using the instant queue length, which turns out to be more responsive than the dynamic calculation performed by CoDel. Also, CoDel feature of marking packets on dequeuing, when combined with DCTCP, seems to be too conservative on reducing the congestion window and therefore does not reduce the latency as much as standalone CoDel or CoDel combined with ECN. By choosing other settings as $2\,ms$ or $4\,ms$ from RED auto configuration we would be able to match CoDel's performance but latency would be increased as well.

V. DISCUSSION AND RECOMMENDATIONS

As mentioned in the introduction, data center networks are starting to employ a new generation of switches with larger buffers, in order to accommodate bursty communications and deliver better application performance. Almost all applications

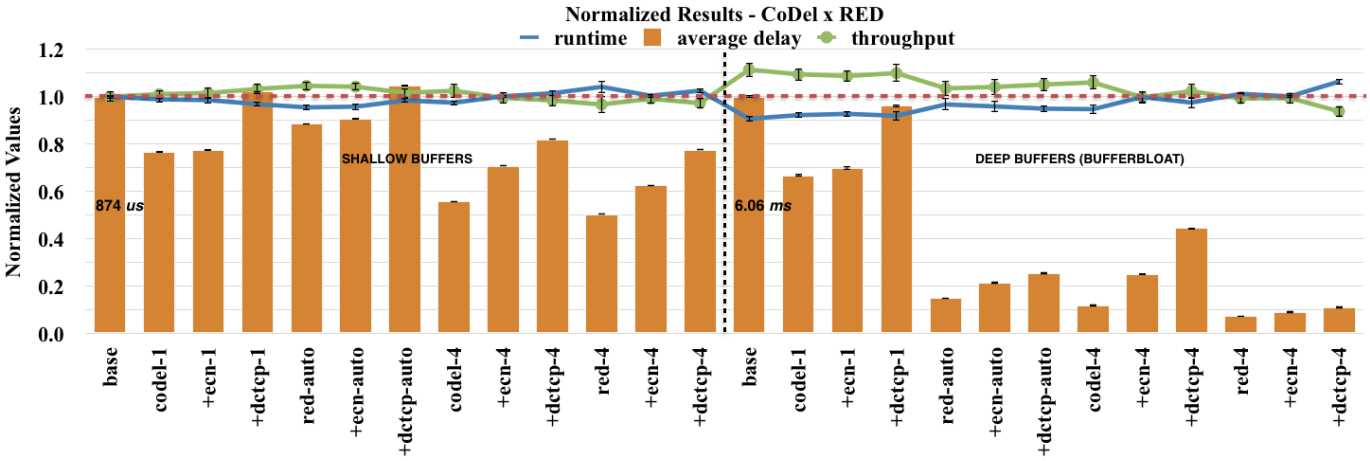

Fig. 6. Normalized Results for CoDel x RED Queues

currently use TCP as the transmission protocol, so they will suffer from large packet latency, due to TCP's tendency to fully utilize the available buffering. Recent switches that implement AQM already support ECN, but until a good understanding of the impact of congestion control on real application performance has been reached, these features are likely to remain switched off, unnecessarily forgoing a feature that could significantly reduce latency.

This paper contributes to the necessary understanding by analyzing the tradeoff in detail for MapReduce workloads, which are representative of modern big data applications in modern data centers.

When congestion happens at an AQM queue, if combined with ECN, it will tell the sender proactively. The naive assumption would be to think that packet loss is always destructive, but an equally naive assumption is to believe that something even more destructive would be the reduction in the size of the congestion window. As seen in the results section, CoDel combined with DCTCP can be too conservative on reducing the congestion window. Therefore the delay is not reduced as much as it was on standalone CoDel or CoDel combined with ECN while the throughput remained about the same.

Surprisingly, with standalone RED or RED combined with ECN, we were able to considerably reduce the latency on bufferbloat scenarios by 85%, and still maintain performance gains from larger buffers within 5%. Using instantaneous queue length instead of the average queue length was extremely necessary to obtaining these results. It also open discussions whether CoDel should have faster converging mode, once its interval, even smaller than 1 ms, seems to deliver much higher latency on similar bases as RED, specially when it is combined with DCTCP.

Finally we finish with recommendations for system administrators. We observed that DCTCP could not achieve the best cuts on latency. That can be explained because DCTCP had not been yet analyzed on scenarios with deep buffers. In deep buffer scenarios, the baseline for latency is considerably higher and the original recommendation of 70 packets, which was presented in DCTCP's evaluation, does not suit for such scenarios. Also, as we mention previously, it may not perform well with just any AQM, as it was the case with CoDel. Therefore we recommend careful consideration before deploying DCTCP on bufferbloat scenarios.

In summary, our observations and recommendations are threefold:

1) Contrary to what could be expected, both RED and CoDel queues may perform better standalone than combined with ECN or DCTCP.
2) For high-speed networks, using instant queue length on RED turned it more responsive than CoDel and its dynamic evaluation.
3) DCTCP delivered the highest latency per set of configuration on both RED and CoDel, but it was not translated on considerable gains on throughput to justify its usage.

VI. Conclusions

A new challenge for modern data centers is to reduce latency caused by large buffers in the network equipment. Specific efforts to reduce latency are related to the requirements of particular workloads. Such actions, however, should not be considered trivial, because the choice of congestion control has a significant effect on throughput and performance, and should not be adopted in practice until the effects on workload performance are well understood.

This paper presented our new work investigating the effects of latency control mechanisms on a Hadoop cluster. We demonstrated how throughput and burst tolerance play important roles for such big data workloads. We evaluated the performance impact on execution time, throughput and latency, when using RED or CoDel, both combined with and without ECN and at last DCTCP as the transport protocol, and found that the MapReduce programming model is not sensitive to the latency but it is sensitive to even small reductions in the network throughput. We demonstrated that in some cases with poorly-chosen AQM configuration the execution

time increases by an unacceptable 20%. We also identified good AQM configurations that were able to maintain Hadoop execution time gains from larger buffer to within 5%, while reducing packet latency caused by bufferbloat by 85%.

Therefore we suggest that cluster administrators carefully consider whether to adopt such techniques, depending on the cluster design and its utilization. For a heterogeneous cluster also running latency-sensitive workloads concurrently, it is important to reduce latency and buffer occupancy caused by larger buffers. In contrast, clusters used only for batch big data processing can neglect the latency, and benefit from the lowest execution time.

In future work, we plan to extend our study to use a protocol that manages better distribution of flows across the cluster as multipath TCP, which promises to improve network utilization and deliver better reliability for workloads.

VII. Acknowledgment

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007–2013) under grant agreement number 610456 (Euroserver). The research was also supported by the Ministry of Economy and Competitiveness of Spain under the contracts TIN2012-34557 and TIN2015-65316-P, Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272), HiPEAC-3 Network of Excellence (ICT- 287759), and the Severo Ochoa Program (SEV-2011-00067) of the Spanish Government.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264

[2] The Apache Software Foundation, "Apache Hadoop Project," http://hadoop.apache.org, accessed: 2016-04-20.

[3] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011. [Online]. Available: http://doi.acm.org/10.1145/2063166.2071893

[4] K. Nichols and V. Jacobson, "Controlled delay active queue management," 2016.

[5] P. Rygielski, S. Kounev, and S. Zschaler, "Model-based throughput prediction in data center networks," in *2013 International Workshop on Measurements and Networking Proceedings (M&N)*. IEEE, Oct 2013, pp. 167–172.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[7] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proceedings of the 1st Workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: ACM, 2009, pp. 73–82. [Online]. Available: http://doi.acm.org/10.1145/1592681.1592693

[8] P. Prakash, A. Dixit, Y. C. Hu, and R. Kompella, "The TCP outcast problem: Exposing unfairness in data center networks," in *Proceedings of the 9th Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 30–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228339

[9] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug 1993.

[10] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2208917.2209336

[11] "References on RED (Random Early Detection) Queue Management," http://www.icir.org/floyd/red.html, accessed: 2016-04-20.

[12] "Configuring weighted random early detection," https://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfwred.pdf, accessed: 2016-04-20.

[13] "RED Congestion Control," https://www.juniper.net/techpubs/en_US/junos12.2/topics/concept/random-early-detection-congestion-control-overview.html, accessed: 2016-04-20.

[14] "Technical introduction to bufferbloat," http://www.bufferbloat.net/projects/bloat/wiki/Bufferbloat, accessed: 2016-04-20.

[15] "Apache Myriad: Deploy Apache YARN Applications Using Apache Mesos," http://myriad.incubator.apache.org/, accessed: 2016-04-20.

[16] A. Bechtolsheim, L. Dale, H. Holbrook, and A. Li, "Why Big Data Needs Big Buffer Switches. Arista White Paper," Tech. Rep., 2011.

[17] Cisco Systems, Inc, "Big Data in the Enterprise - Network Design Considerations White Paper," Tech. Rep., 2011.

[18] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for Data Center Networks," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413181

[19] N. Kuhn, E. Lochin, and O. Mehani, "Revisiting Old Friends: Is CoDel Really Achieving What RED Cannot?" in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, ser. CSWS '14. New York, NY, USA: ACM, 2014, pp. 3–8. [Online]. Available: http://doi.acm.org/10.1145/2630088.2630094

[20] Y. Chen, R. Griffit, D. Zats, and R. H. Katz, "Understanding TCP Incast and Its Implications for Big Data Workloads," no. UCB/EECS-2012-40, Apr 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-40.html

[21] "Network Simulator ns-2," http://www.isi.edu/nsnam/ns, accessed: 2016-04-20.

[22] "Controlled Delay (CoDel) Active Queue Management NS-2 code," http://pollere.net/Codel.html, accessed: 2016-04-20.

[23] "Data Center TCP NS-2 code," http://simula.stanford.edu/~alizade/Site/DCTCP.html, accessed: 2016-04-20.

[24] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "Using realistic simulation for performance analysis of Mapreduce setups," in *Proceedings of the 1st Workshop on Large-Scale System and Application Performance*, ser. LSAP '09. New York, NY, USA: ACM, 2009, pp. 19–26. [Online]. Available: http://doi.acm.org/10.1145/1552272.1552278

[25] Cisco, "Cisco data center spine-and-leaf architecture: Design overview," Tech. Rep., 2016.

[26] A. Vahdat, M. Al-Fares, and A. Loukissas, "Scalable commodity data center network architecture," Jul. 9 2013, uS Patent 8,483,096.

[27] Hortonworks, "Cluster planning guide," http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.3.7/bk_cluster-planning-guide/content/typical-hadoop-cluster-hardware.html, accessed: 2016-04-20.

[28] E. Networks, "Extreme networks: Big data a solutions guide," Tech. Rep., 2014.

[29] Cisco, "Ciscos massively scalable data center: Network fabric for warehouse scale computer," Tech. Rep.

[30] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 266–277. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018467

[31] S. N. Ismail, H. A. Pirzada, and I. A. Qazi, "On the effectiveness of codel in data centers," Tech. Rep.

[32] R. Fischer e Silva and P. M. Carpenter, "Exploring interconnect energy savings under East-West traffic pattern of MapReduce clusters," in *40th Annual IEEE Conference on Local Computer Networks (LCN 2015)*, Clearwater Beach, USA, Oct. 2015, pp. 10–18.

[33] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *2011 19th International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, July 2011, pp. 390–399.

[34] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.