TCP Incast in Data Center Networks: Issue and Existing Solutions

Mahendra N. Suryavanshi, Ajay Kumar

Abstract: Data centers networks support heterogeneous kind of applications like social networking, e-commerce, web search, video data hosting, computation-intensive and data-storage. It has high-bandwidth links, low propagation delay and commodity switches with small-size buffers. Under cluster-based storage environment, data center supports barrier-synchronized many-to-one communication pattern where multiple worker nodes simultaneously transmit bulk of data to single aggregator node by running standard TCP protocol. This synchronized transmission may overload aggregator’s switch buffer, which leads to severe packet loss and overall throughput fall. This is called as TCP Incast problem. This paper analyses issue of TCP Incast and provides detailed survey about several solutions at link, transport and application layer to mitigate impact of TCP Incast in data center network. Solutions are described with their procedural approach for alleviating Incast. Comparative evaluation between these solutions provides understanding about their merits, demerits and applicability under various implementation circumstances.

Keywords: Application, Congestion, Data Center Networks, Goodput, TCP Incast.

I. INTRODUCTION

Today, computing paradigm has changed from mainframe, personal computing to client-server, web technology and cloud computing. Cloud computing offers direct cost benefits and transforms a data center from capital intensive setup to a variable priced environment [1]. This data center used by various institutions for huge data storage and to host web applications like search engines, scientific computing, video data hosting, social networking, large-scale computing, e-commerce, data warehousing and government [2][3][4].

Data center is a repository of computing and storage resources connected together using communication networks. Data centers are designed to achieve high throughput, low latency and better reliability on different load conditions [5]. Increase on load and size, increases complexity of data center networks [6]. There is big challenge for data center designers to design high performing data center network with minimum infrastructure cost and minimum internetwork complexity.

Applications running in data center exhibits barrier-synchronized and many-to-one communication pattern where multiple worker nodes simultaneously transmit bulk of data to a single aggregator node using standard TCP protocol.

This synchronized transmission may overload aggregator’s switch buffer, which leads to severe packet loss and overall throughput degradation. This overall throughput degradation is called as TCP Incast problem in data center network [7][8].

The objective of this survey research is threefold: to analyze TCP Incast issue in data center network, to provide detailed survey about existing link layer, transport layer and application layer solutions for mitigating TCP Incast problem and to perform comparative analysis between these existing solutions using various parameters. Section II is analysis about occurrence of TCP Incast problem in data center network. Section III contains detailed review of existing solutions for solving TCP Incast problem at link layer, transport layer and application layer which is followed by comparative analysis of those solutions in section IV. Section V concludes the paper.

II. TCP INCAST ISSUE IN DATA CENTER NETWORKS

Modern data center networks use TCP protocol to ensure reliable data delivery and efficient congestion control. TCP considered as backbone of the internet [4]. TCP works well in WAN but not suitable for data center networks. Data center networks use high-bandwidth (at least 1Gbps), low-latency (having RTT in hundreds of microseconds) links with Ethernet switches having small-sized buffers [9]. TCP’s default RTO_{min} (200ms) is much greater than RTT of data center network. This results into under-utilization of data center links during loss detection and retransmission mechanism. Under cluster-based storage environment, data center networks use distributed storage frameworks such as MapReduce[10], Panasas[11], HDF5[12], pNFS[13], GFS[14], Cassandra[15], Spark[16], Dryad[17] and BigTable[18] etc. to store immense amount of data on multiple servers. This improves overall performance, reliability and minimize data access latency of applications running in data center networks [19][20]. TCP incast problem initially reported by D. Nagle in PanFS distributed storage system [11].

Applications running in data center networks rely on distributed storage frameworks. The barrier-synchronized and many-to-one communication pattern, where multiple servers simultaneously send data to a single client is shown in Fig. 1. It is also called as Partition/Aggregate communication pattern [21], where a request is partitioned by aggregator (client-node) into small chunks and sent to number of worker nodes (server-node). All worker nodes synchronously transmit response packets back to a single aggregator node through a common Ethernet switch connected to aggregator [22].
TCP Incast in Data Center Networks: Issue and Existing Solutions

In one round of transmission each worker node returns at most 64KB of data which is very small compared to volume of data required by application [4][23][24][25]. This many-to-one communication pattern creates congestion, as multiple workers send response packets to same aggregator through switch having small buffer. This results into packet losses [26]. Packet loss causes client to wait minimum of 200ms. Furthermore, aggregator does not issue next data block request until all workers have finished transmitting their current portion of responses. Hence faster flows stay idle until slowest flow completes its transmission. During this idle period, client will not issue next request and servers cannot send any data even though they have successfully sent data to client. This results into overall throughput collapse.

Fig. 1. TCP Incast scenario in data center network.

Examples of few data center network applications using barrier-synchronized and many-to-one communication pattern are:

- Web-search: Each storage server simultaneously responds with its part of data for the search query [27].
- Social networking: Each server simultaneously responds with its part of user profile [28].
- MapReduce: Intermediate results from many mappers are synchronously transferred to appropriate reducers during the “shuffle” phase [10].
- GFS: After receiving a request each server will simultaneously reply with its part of the file [14]. There are many other e-commerce, data warehousing applications that use barrier-synchronized and many-to-one communication pattern.

Following are reasons for TCP Incast in data center networks:

- Use of limited buffer sized Ethernet switches.
- Mismatch between high-bandwidth, low latency data center networks (with very small RTT value) and TCP’s larger default RTO\textsubscript{max} value.
- Barrier-synchronized and many-to-one communication pattern.
- Servers respond with smaller SRUs e.g. SRUs of size up to 64KB. [27][28][29][30].

TCP Incast problem can be resolved or mitigated at link layer, transport layer and application layer. Various methods and techniques are analyzed and compared with their merits and demerits.

III. REVIEW OF EXISTING METHODS OF SOLVING TCP INCAST PROBLEM

The reduction of packet loss or quick recovery after packet loss can be implemented to resolve TCP Incast problem. Methods and techniques are implemented at link layer, transport layer and application layer.

A. TCP Incast Mitigation at Link Layer

At link layer, two techniques i.e. congestion control and flow control are used to mitigate TCP Incast issue; this is shown in Fig. 2. Link layer solutions depend on switch operations.

Fig. 2. Link layer solutions

1. Congestion Control Method: The QCN (quantized congestion notification) [31] is widely used as a link layer congestion control technique which is based on IEEE 802.1 QAU standards. After receiving congestion notification from switch, hosts reduce their sending rate to avoid frame loss [32]. QCN is implemented at two separate points. First, Congestion Point (CP): These are intermediate switches between source and destination. CP counts incoming packets and track growth rate of queue length. If there is congestion at CP, a negative feedback packet is sent to the source. Reaction Point (RP): RP is the source node which multiplicatively decreases link rate after receiving a negative feedback from CP. During congestion, CP sends same negative feedback value to all RPs which results into unfairness among multiple QCN flows. To avoid this drawback, a modified QCN called as AF-QCN [33] is proposed, which adds fairness among multiple flows. AF-QCN sends negative feedback to each RP based on its measured traffic rate. AF-QCN requires switch to monitor packet arrival rate of each flow, which incurs high overhead when number of flows are large.
2. Flow Control Method: According to Phanishayee et al. [25] TCP Incast problem can be mitigated by using Ethernet flow control (EFC) mechanism through switches that supports EFC. Upon overloading, switch may send a pause frame to sending interface. Upon receiving pause frame, devices connected to that interface stop sending data for a specific period of time. EFC does not work effectively in a topology where clients and servers are connected with multiple switches. EFC has a drawback of head-of-line blocking. Enhanced data center Ethernet performs congestion control through rate-limiting behaviour to improve pause functionality [34].

B. TCP Incast Mitigation at Transport Layer

TCP Incast problem resolved at transport layer can be categorized into three methods: recovery-based method, window-based method and rate-based method. Fig. 3 gives classification of few prominent transport layer solutions to TCP Incast problem.

![Fig. 3. Classification of transport layer solutions](image)

1. Recovery-based Solutions: In data center networks, TCP timeouts is primary reason of overall throughput collapse. With large number of synchronously transmitting servers, limited size of clients switch buffer and small SRUs it is almost impossible to completely avoid packet losses. Hence recovery-based solutions mainly focus on reducing impact after packet losses. There are three types of timeouts that cannot be avoided in data center networks [30][35].

   - Full window loss: Large number of synchronized transmitters generates huge traffic bursts which may cause full window loss of unlucky sender. Hence TCP sender can detect a segment loss only after long idle period of default RTO\(_\text{min}\) (200ms).
   - Last segment loss: TCP receiver cannot alert TCP sender about loss through duplicate ACKs if at least any one of last three segments of SRU dropped. Expiration of RTO\(_\text{min}\) timer is the only way to know packet loss.
   - Retransmitted segment loss: There is no way for sender to know loss if retransmitted segment is dropped again. Sender will be able to detect this loss only after RTO\(_\text{min}\).

   In recovery-based solutions, reducing RTO\(_\text{min}\) timer, T-RACKs, TCP-PLATO, GIP and PRIN does faster recovery from losses instead of allowing server to wait for longer period of time and also avoids above mentioned timeouts. Following are few recovery-based solutions:

   **Reducing RTO\(_\text{min}\) Timer:** Phanishayee A. et al. [25] stated that, in data center networks, default RTO\(_\text{min}\) value (i.e. 200ms) is considerably larger than the RTT value (hundreds of microseconds). After a packet loss at clients switch, server must wait at least for 200ms to detect a packet loss and to trigger retransmission. Client’s link becomes idle during this period of time. One of the solutions to mitigate TCP Incast is to reduce the value of RTO\(_\text{min}\) from default 200ms to a smaller value (e.g. 1ms or 200µs), so that after packet loss, server will detect packet loss and trigger retransmission immediately after 1ms or 200µs as shown in Fig. 4. This leads to efficient use of client’s link capacity and will improve overall throughput [36][37]. This method requires clock granularity in terms of hundreds of microseconds which is not supported by Linux TCP implementations. It is not suitable in a situation where server communicates with clients over a WAN. Reducing RTO\(_\text{min}\) timer results into premature timeouts, this will generate forged retransmissions and TCP will unnecessarily run slow-start though there is no packet loss. It requires modifications to server TCP.

   ![Fig. 4. Reducing RTO\(_\text{min}\) timer protocol procedure](image)
The premature detection of packet loss in T-RACKs may introduce spurious retransmissions and can create severe congestion in network. T-RACKs require interception of each TCP segment to update last received ACK number, SEQ number and other information. It requires additional data structures (flow cache pool, flow table) to be created hence for large number of flows it may create memory pressure.

**TCP-PLATO:** Shukla S. et al. [35] defined TCP-PLATO which detects packet losses using three duplicate ACKs and packet labelling scheme. This is advancement over traditional loss discovery mechanism of TCP. Procedure of TCP-PLATO is described in Fig. 8. Server transmits labelled segments upon receiving labelled ACKs from client and client transmits labelled ACKs upon receiving labelled segments from server. Intermediate switch favorably enqueue labelled segments and ACKs without dropping them. This is done repeatedly within duration of every RTT. This protocol uses labelled segments and ACKs to avoid ACK clock expiration. Furthermore, loss of unlabelled segments is detected by using three duplicate ACKs. After loss, server performs immediate retransmission before RTO\(_{\text{min}}\) expires. Overall goodput will degrade if labelled packets are lost. TCP-PLATO requires modification to server TCP, client TCP and at switch. Middleboxes (e.g. firewalls, IDS, NAT etc.) may drop or corrupt packets with unknown labelling information.
TCP Incast in Data Center Networks: Issue and Existing Solutions

**Guarantee Important Packets (GIP):** Zhang J. et al. [20] defined a simple protocol called GIP. GIP eliminates two kinds of timeouts. First mechanism is employed to avoid timeouts caused by full window losses. This is done by setting congestion window of each server to 2 at the start of each SRU. Second mechanism is used to redundantly (at most 3 times) retransmit last packet of each SRU to generate enough duplicate ACKs and to trigger FRR. This will avoid timeouts caused by lack of ACKs. GIP performs three operations: identifying boundaries of each SRU, decreasing congestion window of each flow at the start of each SRU and retransmitting last packet of each SRU for at most three times. Overall procedure of GIP is given in Fig. 9. The performance of GIP is worse than TCP when numbers of servers are less. GIP requires modification at server TCP.

**Priority-based solution to the TCP Incast problem (PRIN):** Zhang J. et al. [30] implemented PRIN algorithm which modify TCP at client and server side to avoid two kinds of timeouts. To avoid timeout caused by full window loss, congestion window of each flow is reduced by ½ at the start of each SRU. Timeout caused by last segment loss is avoided by setting higher priorities to last three packets and hence preventing them from dropping at switch. PRIN implementation performs three main tasks: to notify about SRU boundaries, to reduce congestion window of each flow by ½ at the start of each SRU and assigning priority to last three packets of SRU. PRIN protocol operation is shown in Fig. 10. For small number of servers, performance of PRIN is worse than TCP. It requires switch support for configuration of VLAN.
Table I is recovery-based protocols and types of unavoidable timeouts they avoid.

Table- I: Recovery-based Protocols Handling Unavoidable Timeouts

| Protocol / Timeout | Full Window Loss | Last Segment Loss | Retransmitted Segment Loss |
|--------------------|------------------|-------------------|---------------------------|
| Reducing RTO<sub>min</sub> Timer | No | No | No |
| T-RACKs | No | Yes | No |
| TCP-PLATO | Yes | Yes | Yes |
| GIP | Yes | Yes | No |
| PRIN | Yes | Yes | No |

As in Table I, TCP-PLATO avoids all three types of unavoidable timeouts. Reducing RTO<sub>min</sub> timer is not specifically to avoid any timeout but just reduce the value of RTO<sub>min</sub>. T-RACKs support large number of concurrent servers and avoid timeout caused by last segment loss only. GIP is easy to implement but supports less number of concurrent servers. TCP-PLATO and PRIN are difficult to deploy as they require operational support of switch.

2. Window-based Solutions: These solutions control incoming traffic by adjusting congestion or receiving window in order to avoid switch buffer overflow and packet losses. Window-based solutions generally necessitate enhancements to existing TCP protocol. Following are few window-based solutions:

Data Center TCP (DCTCP): Alizadeh M. et al. [38] developed DCTCP which uses modified ECN mechanism. It helps to achieve low latency and high throughput by keeping switch buffer vacancy less than threshold value. This protocol contains three components. Marking of CE codepoint at switch: if queue occupancy is more than marking threshold K, newly arrived packet at switch marked with CE codepoint. Setting of ECN-Echo flag at client: upon receiving packet marked with CE codepoint, client DCTCP sends ACK back to the server by setting ECN-Echo flag in it. Congestion controller at server: server receives ECN-Echo flag marked ACK packets whenever switch queue occupied higher than threshold K. Server estimates probability of queue size greater than K using this fraction of marked packets (i.e. α). After estimating α, server updates congestion window to lessen sending rate. This overall procedure is depicted in Fig. 11. DCTCP starts marking packets very early, which shows premature indication of congestion. It requires modifications at switch, server TCP and client TCP. For large number of concurrent servers, DCTCP cannot deal with incast problem. In DCTCP, significant amount of time is required for a new flow to obtain its share of bandwidth from an existing running flow having large window size.

Incast Congestion Control for TCP (ICTCP): Haitao et al. [39] proposed a protocol called ICTCP. ICTCP adjusts TCP receive window size of each TCP connection based on RTT and measured available bandwidth at client. It prevents switch buffer overflow by controlling sending rates of servers. Working of ICTCP is mainly divided into two parts. First, available bandwidth calculation: ICTCP calculates available bandwidth BW<sub>A</sub>. Second, receive window adjustment: The receive window adjustment is based on ratio of difference of incoming measured throughput (i.e. b<sup>n</sup>) and expected throughput (i.e. b<sup>e</sup>) over expected throughput. The receive window adjustment is performed by using two threshold values (γ<sub>1</sub>=0.1 and γ<sub>2</sub>=0.5) as shown in Fig. 12. ICTCP avoid Incast with none timeout events. It offers high goodput and allows fair sharing of available bandwidth among multiple flows. ICTCP is designed for mitigating incast only when congestion is between switch and client. ICTCP requires modification to client TCP. ICTCP requires calculation of throughput after every RTT on each TCP connection. It also requires implementation of additional shim-layer on top of TCP layer at client side.

Fig. 11. Data Center TCP (DCTCP) protocol procedure

Fig. 12. Incast Congestion Control for TCP (ICTCP) protocol procedure
M21TCP: M21TCP (Adesanmi A. et al [40]) protocol operation is shown in Fig. 13. M21TCP consists of three components. First, router/switch operation: switch keeps track about number of distinct flows currently traversing through the interface and determine count of synchronously transmitting servers. This server count, which is valid only for period of one RTT, is sent to the client by encoding it in a packet. Second, client operation: M21TCP client gets synchronously transmitting servers count from switch and forward it back to server through ACK packet. Third, server operation: Each server calculates Max_Wind using switch buffer size (B), minimum header size (MHS) and number of servers determined by switch (N). Then using Max_Wind, server calculates congestion window (cwnd). M21TCP achieves high throughput for up to 64 number of concurrent servers before incast occurs. This protocol requires modifications to switch, server TCP and client TCP. Additional 32-bit TCP or IP field must be supported by server. Switch must keep a list of all incoming flows. This adds an additional tracking overhead and memory pressure on switch with limited buffer size.

Fig. 13. M21TCP protocol procedure

Stochastic Adjustment TCP (SA-TCP): Ren Y. et al. [27] developed a simple mechanism called SA-TCP to mitigate impact of TCP Incast. SA-TCP uses additive increase and multiplicatively decrease algorithm for congestion control. Instead of constant values of α and β (i.e. α = 1 and β = 0.5), SA-TCP makes α and β as stochastic (random) variable values (i.e. α = random [α_min, α_max] and β = random [β_min, β_max]). Stochastic values of α and β helps to avoid synchronization among multiple concurrently transmitting servers. Fig. 14 is the flowchart of SA-TCP. SA-TCP requires modification at server TCP. SA-TCP allows multiple flows to fairly share aggregate bandwidth. SA-TCP supports small number of servers, i.e. overall goodput decreases when number of concurrent servers are 16 and above. SA-TCP requires modification at server TCP.

Fig. 14. Stochastic Adjustment TCP protocol procedure

Incast Decrease TCP (IDTCP): Wang et al. [41] implemented IDTCP to mitigate TCP Incast through three techniques. First, IDTCP continuously measure minimum RTT (RTT_base) and average RTT (RTT_avg) values. These values are used to estimate congestion level of link. Second, according to estimated congestion level of link, IDTCP slows down and dynamically regulates congestion window growth rate. With increase in congestion level, IDTCP decrease congestion window growth rate. Furthermore, number of concurrent servers increases by slowing down congestion window growth rate. Third, if average RTT is twice than minimum RTT and queuing delay equals to 1, link is considered as completely congested. IDTCP set congestion window to 1 if link is completely congested. Fig. 15 is the working of IDTCP. IDTCP requires modification to server TCP. It does not utilize network bandwidth efficiently when numbers of servers are small.

Fig. 15. Incast Decrease TCP (IDTCP) protocol
3. Rate-based Solutions: Rate-based solutions control incoming traffic and reduces load on clients switch buffer by adjusting server’s packet sending rate or client’s ACK sending rate. Rate-based solutions attempts to fully utilize bottleneck link for large numbers of servers before a switch buffer overflows. Rate-based solutions are given as follows:

Incast-Avoidance TCP (IA-TCP): Hwang et al. [24] implemented IA-TCP only at client side to control transmission rate of servers through randomized ACK regulation. The overall working of IA-TCP is shown in Fig. 16. Function of IA-TCP is divided into 2 parts. One is, controlling server’s transmission rate for Incast-avoidance: client initially controls each server’s congestion window size then adds random delay for rate control so that transmission rate of servers are controlled to be less than or equal to the bandwidth of link. Other is, regulating ACK at client: client regulates ACK intervals so that aggregate data packet rate will be less than or equal to link capacity. IA-TCP requires modification at client-side.

![Fig. 16. Incast-Avoidance TCP (IA-TCP) protocol procedure](image)

**LT-code based Transport Protocol (LTTP) procedure**

Jiang C. et al. [28] proposed LTTP. LTTP procedure is shown in Fig. 17. LTTP improves UDP-based LT (Luby Transform) code for reliable data delivery and use TFRC (TCP Friendly Rate Control) for congestion control to adjust data sending rates at servers. To perform LT code based transport, original data is divided into equal size blocks called as input symbols. Then encoding symbols are generated by performing XOR operations on separate d input symbols, d is the degree number of encoding symbol. After this, an encoding packet is generated by merging encoding symbol with degree number and indices of all input symbols of encoding symbol. Each encoding packet contains this extra information so that decoder can easily restore the original data. To perform TFRC based congestion control, after receiving packet, client send sequence number and timestamp value of received packet back to the server. Server uses these values to determine RTT. Server detects packet loss if there is a gap in received sequence number. Server terminates slow start phase and enters congestion avoidance phase, whenever client indicates losses through feedback packet. LTTP ensures equal bandwidth sharing between LTTP flows and other TCP flows. LTTP provides slow response to congestion. For small number of servers (less than 15) LTTP perform worse than TCP. LTTP introduces additional computation overhead for encoding and decoding data that is to be transmitted. LTTP wastes significant amount of bandwidth for achieving data redundancy.

![Fig. 17. LT-code based Transport Protocol (LTTP) procedure](image)

**Proactive ACK Control (PAC):** Bai W. et al. [19] designed Proactive ACK Control (PAC) protocol at client side. According to PAC algorithm described in Fig. 18, upon receiving new packet, PAC client generates ACK, enqueue it into ACKqueue, updates value of incoming traffic volume and update value of threshold. PAC client checks if releasing enqueued ACK will cause incoming traffic volume to be more than the threshold value and will not release ACK until there is enough space in switch buffer. The overall performance of PAC depends upon three key aspects. First, set proper threshold value: to prevent incast congestion and to ensure high bottleneck link utilization, threshold is set to be the size of switch buffer. Fraction of incoming ECN marked packets used to estimate congestion level and threshold is adjusted accordingly. Second, estimating incoming traffic: upon releasing an ACK packet and receiving data packet, increment and decrement incoming traffic value respectively. Third, schedule ACK packets: at client side, Multi-level Feedback Queue (MLFQ) is implemented to schedule ACK packets.
TCP Incast in Data Center Networks: Issue and Existing Solutions

MLFQ contains N distinct queues of different priorities. These queues are used to temporarily store ACK packets. Multiple PAC flows can fairly share available link capacity. As PAC schedules ACKs, RTOs are extended beyond RTO_min. Spurious retransmissions will be made after undesirable TCP timeouts. PAC requires additional memory space to temporarily buffer ACKs at client.

C. TCP Incast Mitigation at Application Layer

Solutions at application layer do not require any modification to TCP or switch. The main principle is to reduce packet loss by restricting number of synchronously communicating servers. Fig. 20 shows few application layer solutions to TCP Incast problem.

1. Scheduling Client’s Data Requests to Servers: Dou Ke et al. [3], proposed a solution to TCP throughput collapse by scheduling client’s data requests to concurrent servers. Data block requests made by a client can be split into K groups such that K = [n/N], where n is total number of concurrent servers and N is number of concurrent servers in each group. Furthermore, N = max {1, [B/C]}, B is switch buffer size in bytes and C is link capacity (in bits per second) [43][44]. After processing data received from one group of servers, client issue a data request to next server group. Moreover random intervals are used to send data request to each group randomly. This method does not support parallel data transfer hence underutilizes link capacity.

2. Increasing SRU Size: Overall link capacity can be effectively utilized by increasing SRU size. The continuous data transmission could not stop even if few servers are not able to perform transmission [7]. Also, increasing SRU size requires less number of servers involved in simultaneous transmission. Increased SRU size requires enough client kernel memory for storage hence increases memory pressure on client [25].

3. Limiting Number of Synchronously Communicating Servers: The simplest and effective application layer solution to mitigate incast collapse is to limit total number of concurrent servers involved in parallel data transmission. Krevat E. et al. [7] stated that, to effectively implement this solution, acceptable range of simultaneously transmitting servers should be identified through prior experiments. This solution can significantly avoid switch buffer overflow and achieves overall goodput improvement. This technique is not suitable for real data center networks.

4. Throttling Data Transfer: Krevat E. et al. [7] suggested throttling data transfer technique where client can advertise smaller TCP receive buffer and control data transmission rates of simultaneously transmitting servers. This solution helps to limit TCP window size, allows large number of servers to participate in synchronous transmission and hence improves overall performance. Limiting TCP window size to a smaller value by using static throttling rate can degrade utilization of client’s link capacity. Due to complexity, throttling technique is less advantageous.

5. Global Scheduling of Data Transfer: Global scheduling of data transfer is another application layer solution proposed by Krevat E. et al.
[7] where, SRU token is used to notify servers for data transmission. A particular server transfers data to a client only after obtaining client’s SRU token. Clustered-system determines maximum number of concurrent servers (say k) that can transmit synchronized data to a client before Incast occurs. System generates only k number of SRU tokens to be sent to k number of servers. Only these k numbers of servers can simultaneously transfer data to a client, even though client has sent request packets to all potential servers.

IV. COMPARATIVE ANALYSIS OF TCP INCAST SOLUTIONS

In this section, we have performed comparative analysis of 2 link layer, 14 transport layer and 5 application layer solutions to TCP Incast problem.

A. Comparative Analysis of Link Layer Solutions

Table II contains comparison between link layer solutions using server-side modification, client-side modification, switch operation modification and maximum number of concurrent servers supported parameters. It is showing that QCN and EFC both require server-side and switch operation modifications. QCN supports double number of concurrent servers than EFC.

B. Comparative Analysis of Transport Layer Solutions

Comparative analysis of transport layer protocols based on 6 parameters i.e. type of congestion control method, server TCP modification, client TCP modification, switch operation modification, additional shim-layer implementation and maximum number of concurrent servers supported is given in Table III. For better reading P-1, P-2, P-3, P-4, P-5 and P-6 naming labels are used respectively for above mentioned 6 parameters.

According to Table III, IA-TCP is easy to implement as it requires modification at client-side TCP only. TCP-PLATO, PRIN, DCTCP, M21TCP and LTTP modify TCP at both client-side and server-side. TCP-PLATO, PRIN, DCTCP and M21TCP demands modification at all types of network devices, hence they are complex to implement and deploy. SA-TCP is window-based solution to TCP Incast problem which supports least number of concurrent servers (i.e. up to 16). A rate-based solution, PAC supports very large (i.e. up to 1600) number of concurrent servers before TCP Incast occurs. Window-based solutions are complex to design and deploy. They support less number of concurrent servers as compared to recover-based and rate-based solutions. Rate-based solutions support large number of concurrent servers.

C. Comparative Analysis of Application Layer Solutions

Comparison of five application layer solutions to mitigate TCP Incast is given in Table IV. These solutions are compared based on violation of cluster-based system characteristic, underutilization of link capacity and increased memory pressure on client.

Table- II: Comparative Analysis of Link Layer Solutions

| Link Layer Protocol | Server-side Modification | Client-side Modification | Switch Operation Modification | Maximum Number of Concurrent Servers Supported |
|---------------------|--------------------------|--------------------------|-------------------------------|-----------------------------------------------|
| QCN                 | Yes                      | No                       | Yes                           | <= 64                                        |
| EFC                 | Yes                      | No                       | Yes                           | <= 32                                        |
It is concluded from Table IV that scheduling client’s data requests to servers, limiting number of synchronously communicating servers and throttling data transfer protocols underutilizes link capacity. Increasing SRU size and throttling data transfer protocols do not violate cluster-based storage systems characteristic. Increasing SRU size puts memory pressure on client but client with high memory capacity can be able to handle memory pressure easily.

Furthermore link, transport and application layer solutions are compared together based on switch modification, host modification, violation of cluster-based system characteristics and number of concurrent servers supported. Number of concurrent server supported is categorized into small (15-50 servers), medium (50-64 servers) and large (64 servers onwards). Table V describes that link layer protocols requires switch modification, host modification and support medium number of concurrent servers. Each transport layer protocol requires host modification either through additional shim-layer implementation or TCP protocol modification. Few of them (i.e. TCP-PLATO, PRIN, DCTCP, M21TCP and PAC) require additional burden of switch operation modification. As compared to link layer and application layer protocols, transport layer protocols supports large number of concurrent servers. Most of the application layer protocols losses parallelism in data transfer which is key characteristic of cluster-based storage systems and supports small number of concurrent servers.

V. CONCLUSION

In this paper, we have analyzed TCP Incast issue in data center networks. We also provided in-depth survey and comparative analysis of several existing solutions that are used to mitigate TCP Incast at link, transport and application layer. Protocols at application layer are easy to implement but support fewer number of concurrent servers hence less suitable for large-scale communication requirements. Rate-based transport protocol IA-TCP modifies client-side TCP only and supports large concurrent servers. Recovery-based (TCP-PLATO, PRIN) and window-based (DCTCP, M21TCP) transport protocols are difficult to implement and deploy as they need modifications at all network devices. Window-based transport layer solutions do not support large number of concurrent servers hence are less suitable for real data center network communications. Compared with recovery-based and window-based solutions, rate-based transport layer solutions improve overall goodput while supporting large number of concurrent servers. According to this survey, the most effective approach to mitigate TCP Incast is to control the congestion by varying intervals of packet transmission rates or ACK transmission rates. This can be achieved through modification of TCP stack either at server-side or client-side.

REFERENCES

1. R. Alshahrani and H. Peyravi, “Modeling and Simulation of Data Center Networks,” 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, May, 2014, pp. 75-82.
2. M. Al-Fares, A. Loukissas, and A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” ACM SIGCOMM 2008 conference on Data communication, vol. 38, no. 4, August 2008, pp. 63-74.
3. D. Ke, R. Yongmo, and L. Jun, “Avoiding TCP incast through scheduling data requests,” Fourth International Conference on Multimedia Information Networking and Security, January, 2013, pp. 453-457.
4. Y. Ren, Y. Zhao, P. Liu, K. Dou, and J. Li. (2012, August). A survey on TCP Incast in Data center networks, International Journal of Communication Systems, 27(8), pp. 1160-1172.
5. R. Couto, M. Campista, and J. Gehr. “A Reliability Analysis of Data Center Topologies,” IEEE Global Communications Conference (GLOBECOM), December. 2012. pp. 1890-1895.
6. F. Yao, J. Wu, G. Venkataramani, and S. Subramanian, “A Comparative Analysis of Data Center Network Architectures,” IEEE International Conference on Communications (ICC), June, 2014. pp. 3106-3111.
7. E. Krevat, V. Vasudevan, A. Phanishayee, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, “On Application-level Approaches to Avoiding TCP Throughtput Collapse in Cluster-based Storage Systems,” 2nd International workshop on Petascale data storage, November. 2007, pp. 1-14.
8. P. Sreekumari, and J. Jung. (2015, August). Transport protocols for data center networks: a survey of issues, solutions and challenges. Photonic Network Communications, 31(1), pp. 112-128.
9. Y. Yu, and C. Qian, “Space shuffle: A Scalable, Flexible, and High-Bandwidth Data Center Network,” IEEE 22nd International Conference on Network Protocols, October 2014. pp. 13-24.
10. J. Dean, and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, January. 2008, pp. 107-113.
11. D. Nagle, D. Serenyi, and A. Matthews, “The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage,” ACM/IEEE Conference on Supercomputing, November. 2004, pp. 53-62.
12. K. Dwivedi, and S. Dubey, “Analytical review on Hadoop Distributed file system,” 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), September 2014, pp. 174-181.
13. K. Deng, C. Lee, J. Chou, Y. Shih, S. Chuang, and P. Wu., “PNFS-Based Software-Defined Storage for Information Lifecycle Management,” International Conference on Cloud Computing and Big Data, November. 2015. pp. 89-92.
14. S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” In Proc. 19th ACM Symposium on Operating Systems Principles (SOSP), vol. 37, no. 5, October. 2003, pp. 29-43.
15. A. Lakshman, and P. Malik. (2010, April). Cassandra - A Decentralized Structured Storage System. ACM SIGOPS Operating Systems Review, 44(2), pp. 35-40.
16. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: a fault- tolerant abstraction for in-memory cluster computing,” 9th USENIX conference on Networked Systems Design and Implementation.
17. M. Isard, M. Budiu, Y. Yu, A. Birell, and D. Fetterly. (2017, March). Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Operating Systems Review. 41(3), pp. 59–72.
18. F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, F. Chandra, A. Fikes, and R. Gruber. (2008, June). Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems. 26(2), pp. 205-218.
19. W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, “PAC: Taming TCP Incast Congestion using Proactive ACK Control,” IEEE 22nd International Conference on Network Protocols, October. 2014. pp. 385-396.
TCP Incast in Data Center Networks: Issue and Existing Solutions

20. J. Zhang, F. Ren, L. Tang, and C. Lin, “Taming TCP Incast Throughput Collapse in Data Center Networks,” 21st IEEE International Conference on Network Protocols, October, 2013.

21. S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chakien, “The Nature of Datacenter Traffic: Measurements & Analysis,” 9th ACM SIGCOMM conference on Internet measurement, November, 2009, pp. 202-208.

22. T. Benson, A. Akella, and D. Maltz, “Network Traffic Characteristics of Data Centers in the Wild,” 10th ACM SIGCOMM conference on Internet measurement, November, 2010, pp. 267-280.

23. A. Abdelmonem, and B. Bensaou, “Curbing Timeouts for TCP-Incast in Data Centers via a Cross-Layer Faster Recovery Mechanism,” IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, April, 2018, pp. 675-683.

24. J. Hwang, J. Yoo, and N. Choi, “IA-TCP - A Rate Based Incast-Avoidance Algorithm for TCP in Data Center Networks,” IEEE International Conference on Communications (ICC), June, 2012, pp. 1292-1296.

25. A. Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, “Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems,” 6th USENIX Conference on File and Storage Technologies, February, 2008, pp. 1-14.

26. W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, and F. Baker, “Comprehensive Understanding of TCP Incast Problem,” IEEE Conference on Computer Communications (INFOCOM), August, 2015, pp. 1688-1696.

27. Y. Ren, J. Li, G. Wang, L. Li, and S. Shi, “SA-TCP: A Novel Approach to Mitigate TCP Incast in Data Center Networks,” International Conference on Computing and Network Communications, December, 2015, pp 420-426.

28. C. Jiang, D. Li, and M. Xu, (2014, January). LTTP: An LT-Code Based Transport Protocol for Many-to-One Communication in Data Centers. IEEE Journal on Selected Areas in Communications, 32(1), pp. 52-64.

29. J. Huang, T. He, Y. Huang, and J. Wang.” ARS - Cross-layer Adaptive Request Scheduling to Mitigate TCP Incast in Data Center Networks,” IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, April, 2016, pp. 1-9.

30. J. Zhang, F. Ren, L. Tang, and C. Lin, (2015, Feb). Modeling and Solving TCP Incast Problem in Data Center Networks. IEEE Transactions on Parallel and Distributed Systems, 26(2), pp. 478-491.

31. P. Devkota, and A. Reddy, “Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers,” 18th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), August, 2010, pp 235-243.

32. M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshminath, R. Pan, B. Prabhakar, and M. Seaman, “Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization,” 46th Annual Allerton Conference, September, 2008, pp. 1270-1277.

33. A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, “AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenant Data Centers,” 18th IEEE Annual Symposium on High Performance Interconnects, August, 2010, pp. 58–65.

34. M. Wadeker, “Enhanced Ethernet for Data Center: Reliable, Channelized and Robust,” 15th IEEE Workshop on Local and Metropolitan Area Networks, June, 2007, pp. 65-71.

35. S. Shukla, S. Chan, A. Tam, A. Gupta, Y. Xu, and H. Chao, (2014, January). TCP PLATO - Packet Labeling to Alleviate Time-out. IEEE Journal on Selected Areas in Communications, 32(1), pp. 65-76.

36. Y. Chen, R. Griffith, J. Liu, R. Katz, and A. Joseph, “Understanding TCP incast throughput collapse in datacenter networks,” 1st ACM workshop on Research on enterprise networking, August, 2009, pp. 73-82.

37. V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller, “Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication,” ACM SIGCOMM 2009 conference on Data communication, vol. 39, no. 4, October, 2009, pp. 303-314.

38. M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, (2010, October). Data Center TCP (DCTCP). ACM SIGCOMM Computer Communication Review, 40(4), pp. 63–74.

39. W. Haitao, Z. Feng, C. Guo, and Y. Zhang, (2013, April). ICTCP: Incast Congestion Control for TCP in Data Center Networks. IEEE/ACM Transactions on Networking. 21(2), pp. 345-358.

40. A. Adesami, and L. Mhamdi, “Controlling TCP Incast Congestion in Data Center Networks,” IEEE International Conference on Communication Workshop, June, 2015, pp. 1827-1832.

41. G. Wang, Y. Ren, K. Dou, and J. Li. (2013, November). IDTCP: An Effective approach to mitigating TCP Incast problem in data center networks. Information Systems Frontiers. 16(1), pp. 35-44.

42. S. Zou, J. Huang, Y. Zhou, J. Wang, and T. He, “Flow-Aware Adaptive Facing to Mitigate TCP Incast in Data Center Networks,” IEEE 37th International Conference on Distributed Computing Systems, June, 2017, pp. 2119-2124.

43. H. Zheng, and C. Qiao, “An Effective Approach to Preventing TCP Incast Throughput Collapse for Data Center Networks,” IEEE Global Telecommunications Conference, December, 2011.

44. M. Podlesny, and C. Williamson, “An Application-Level solution for the TCP-Incast problem in Data Center Network,” IEEE Nineteenth IEEE International Workshop on Quality of Service, June, 2011, pp. 1-3.

AUTHORS PROFILE

Mahendra N. Suryavanshi, Is Currently Working As Assistant Professor In Department Of Computer Science And Engineering, Pune, Maharashtra, India. He Is Pursuing Ph.D At Computer Science Department Of Savitribai Phule Pune University, Maharashtra, India. He is Currently Doing Research On Data Center Networks Communications Improvement. Mahendra N. Suryavanshi Has Completed M.Sc. (Computer Science) In 2010 From Savitribai Phule Pune University And Qualified UGC National Eligibility Test (NET) In 2013. Research Interests Of Mahendra N. Suryavanshi Include Network Programming, Cloud Computing, Data Center Networks And Computer Networks.

Dr. Ajay H. Kumar, has done B.Sc (Engineering), M.Sc Applied Science (Engineering and Technology) and Ph.D. His research areas are Cloud Computing, Internet of Things, Mobile Computing, Big Data Analysis and Biometric Identification. Dr. Ajay Kumar currently working as a Director at JSPM’s Jaywant Technical Campus Pune, Maharashtra, India. He is recognized Ph.D guide of computer science subject at Savitribai Phule Pune University. He has more than 35 years of experience in research, teaching and administration. He has published plenty of research papers in scopus, web of science and other international journals. He has been invited as speaker and session chair in various national and international conferences.

Published By: Blue Eyes Intelligence Engineering & Sciences Publication