Towards Efficient Load Distribution in Big Data Cloud

Zhi Liu[*†], Xiang Wang[*†], Weishen Pan[*], Baohua Yang[§], Xiaohe Hu[*] and Jun Li[†‡]
[*]Department of Automation, Tsinghua University, Beijing, China
[†]Research Institute of Information Technology, Tsinghua University, Beijing, China
[§]IBM China Research Lab, Beijing, China
[‡]Tsinghua National Lab for Information Science and Technology, Beijing, China
{zhi-liu12, xiang-wang11, pws11, huxh10}@mails.tsinghua.edu.cn, baohyang@cn.ibm.com, junl@tsinghua.edu.cn

***Abstract*—Traffic spikes in big data cloud have led to great challenges for middlebox management in the behind datacenter networks, especially for load distribution among varieties of middleboxes. Contemporary flat load distributing strategies are inadequate to handle traffic spikes in terms of time and resource efficiency. Based on the spike patterns, a hybrid load distributing solution is proposed with hierarchical architecture, which conducts static distributing and dynamic collaborative distributing in respect of flow types. The evaluation results from the prototype system and simulation show that our solution is superior to existing load distributing strategies in terms of both time and resource efficiency.**

Keywords—load distributing; middlebox; big data; datacenter network

I. INTRODUCTION

Big data cloud, providing the analysis of huge amount of data, has been growing rapidly recently. Applications in big data cloud require high throughput and low latency for the frequent data transferring [1], imposing huge traffic pressure to the behind datacenter networks. On the other hand, recent study [2-3] showed that a great amount of traffic in datacenter traverses through middleboxes (MBs), which conduct a variety of functionalities such as security, caching, compression and encoding. In this case, traffic needs to be redirected to MBs before coming to its final destination. In order to guarantee the capacity of MBs, load distributing is required among MB instances of the same functionality.

In datacenters supporting big data applications, the network is highly dynamic and forms complex traffic pattern. One key pattern is "bursty", which indicates that the traffic load varies greatly in seconds or even milliseconds and therefore results in traffic spikes. One important cause for spikes is the Partition/Aggregate workflow pattern [1] introduced by big data platforms like Hadoop [7], Spark [8], etc. These spikes bring about great challenges for middlebox load distributing, making it hard to achieve low packet latency as well as high resource utilization.

Currently, there are two common load distributing strategies, i.e. ***static load distributing*** and ***global load balancing***.

Static load distributing strategy collects network configuration (e.g. logical service chain, security policy, and QoS) from administrators, and network status from underlying infrastructure (e.g. topologies, link utilization, MB locations and capacity). Load distributing policies are then generated after a series of optimizations, including minimization of traffic traversing distance and elimination of link bottleneck. Load distributing policies are then compiled into flow entries in switches, which steers traffic accordingly. Most of the time, load distributing policies remain static, and are only updated when topology or logical connectivity changes. Recent studies [3, 4] proposed several techniques for better load assignments. With optimal flow assignment, static load distributing strategy minimizes the average flow latencies. However, it does not consider the spikiness of traffic load. As a consequence, it is not flexible enough to deal with bursty traffic efficiently. Current implementations have to over-provision MB resources to guarantee capacity and avoid packet loss.

Global load balancing aims at scattering the workloads among all the MB instances. Mostly, there are several distributed load balancer in the network, exchanging load information with a centralized controller. The controller implements the load balancing strategy into each load-balancer, which actually steers traffic when each flow arrives. This strategy is capable of preventing MB instance from being over-whelmed by the bursty traffic. However, when massive flows are redirected to remote MB instances, latency increase and link congestion become inevitable. Moreover, the global load updating of MBs also introduce high system overhead.

In summary, current solutions are not adequate to handle the load distributing task efficiently since they treat all the flows equally. This paper proposes a hybrid load distributing solution, which is able to handle bursty traffic with both time and resource efficiency, and hence improves overall traffic load distributing performance. The main contributions of this paper are:

TABLE I. Comparison of Load Distributing Strategies

	Static Load Distributing	**Global Load Balancing**
Time Efficiency	Short latency	Long latency
Resource Efficiency	Poorly utilized	Highly utilized

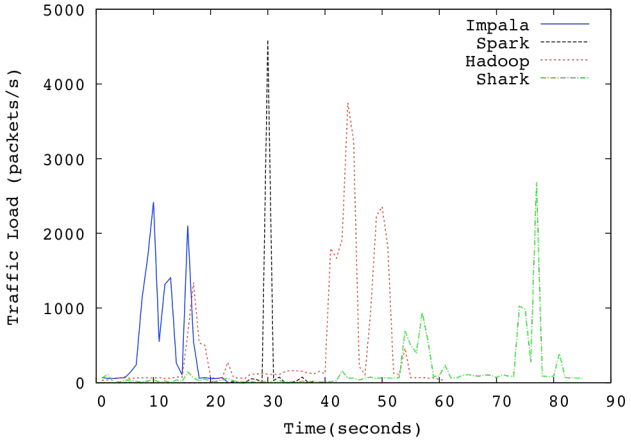

Figure 1. Bursty Traffic of Big Data Applications

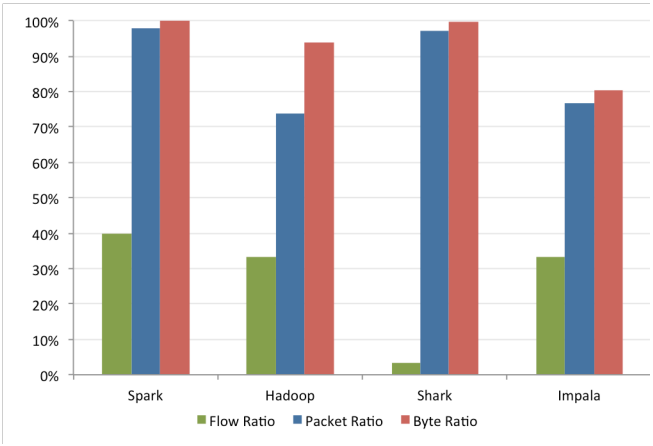

Figure 2. Temporary Flow Proportion in Bursty Traffic

- Studied the burstiness of big data applications and revealed that temporary flows with short durations contribute to large proportion of traffic spikes, and that bursty flows can be classified with high accuracy.
- A hybrid load distributing solution was proposed, which conducts ***static distributing*** and ***dynamic collaborative distributing*** for normal flows and bursty flows respectively.
- Experiment results on a prototype system and simulations demonstrated that the proposed solution is able to efficiently deal with bursty traffic in terms of processing time and resource utilization.

The rest of this paper is organized as follows: Section II studies traffic burstiness of big data applications and reveals the feasibility of distributing traffic spikes. Section III and IV elaborate our hybrid load distributing strategy and system design. Evaluations based on prototype system and simulations are given in section V. And we draw the conclusion in section VI.

II. BURSTY LOAD PATTERNS OF BIG DATA APPLICATIONS

To obtain more detailed insight of bursty traffic, four popular big data applications, i.e. Hadoop[7], Spark[8], Shark[14] and Impala[15] have been analyzed. All the applications run in experimental clusters, where all traffic was captured in real time. Figure 1 shows the inter-node traffic of each application. It is obvious that all of these applications have bursty patterns, leading to huge traffic rate fluctuations.

A. Bursty Traffic Patterns

When traffic spike occurs, the traffic rate rises rapidly within 1~2 seconds, with peak traffic rate around 5000 packet per second (pbs) and none bursty rate below 100pbs. In addition, all of the spikes have short durations, with a maximum duration of 10 seconds. According to previous studies [5, 6], traffic in datacenters also possess similar spike patterns. In [5], it was revealed that more than 80% of the flows last less than 10 seconds and forms transient spikes, which corresponds to our study.

Such bursty traffic brings about significant challenges for MB load distributing. A sharp leap in traffic rate always leads to overfilled MB buffer and packet loss, making spikes the primary cause of packet losses [6]. Without careful scheduling, such traffic is extremely prone to severe performance decline.

The compositions of traffic spikes are also studied by analyzing the spike traces. We zoomed into the bursty periods of "Spark-kmeans" (15 flows), "Hadoop-sort" (40 flows), "Shark-count" (608 flows) and "Impala-count"(75 flows). ***Temporary flows***, which are established at the very beginning of spikes and persist until the spike vanishes, occupy a dominant proportion of the traffic bursts. As is shown in Figure 2, temporary flows occupy more than 70% of the total packets and more than 90% of the total bytes of the spike loads. Therefore, it is found that the majority of the bursty traffic load resides in ***temporary flows***. This finding also reveals the possibility to distribute the bursty flows and amortize the load among multiple MBs.

B. Bursty Flow Signatures

One key problem of distributing spike loads is whether bursty flows can be identified. Taking a further step, we studied the unique pattern of the bursty flows and found that they always contain explicit signatures, which enables fast and accurate classification. By analyzing the spike traces, we categorize the bursty flow signatures into two classes:

Flow tuple signatures. It is found that some of the applications use pre-defined port to transfer the bursty flows. For example, Hadoop nodes listens to port 50010 for bursty load, therefore, flows with 50010 as source or destination port have great probability to carry bursty load. Shark and Impala also contain signatures like this.

Flow payload signatures. Some applications do not use pre-defined ports for bursty data. Instead, their nodes negotiate the port number before bursty flows are established.

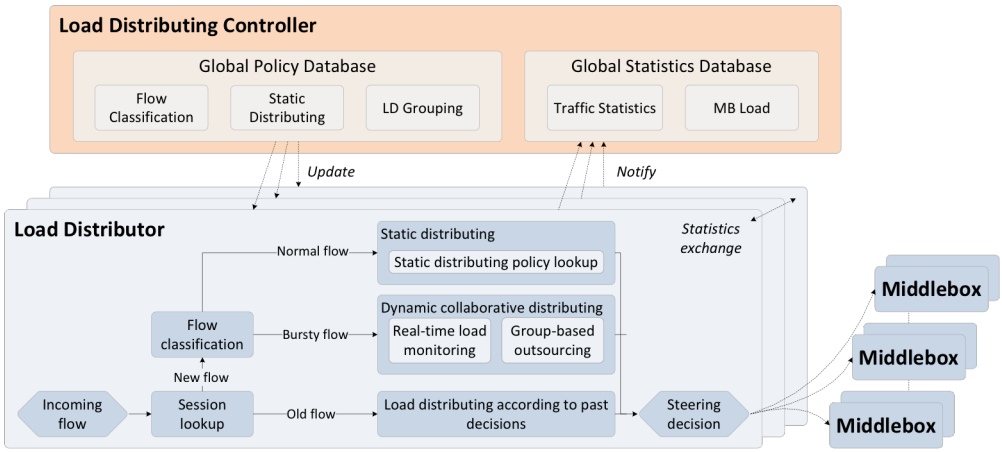

Figure 3. Hybrid Load Distributing Overview

Nodes of Spark, for instance, will negotiate the port number of the scheduling bursty flows in format of "spark.fileserver.urit..http://[ip]:[port]", from which the bursty port number can be extracted by real-time string matching and further added to port number policies for flow classification. Therefore, bursty flows can be classified by inspecting packet headers and payloads. The evaluations in Section 5 will show that our classification module is able to identify bursty flows with high accuracy.

III. HYBRID LOAD DISTRIBUTING

In order to handle the traffic load spikes more efficiently, we propose the hybrid load distributing, which combines ***static distributing*** and ***dynamic collaborative distributing*** to deal with normal flows and bursty flows, respectively. This section starts with introducing the components of the system, followed by elaborating the proposed hybrid load distributing strategy.

A. System Overview

Figure 3 shows the high-level flowchart of hybrid load distributing system, which includes ***Load Distributor*** and ***Load Distributing Controller***.

Load Distributor (LD) is responsible for directing each incoming flow to the appropriate MB instance. LDs are implemented in data plane using VMs, and conducts real-time load monitoring for a group of closely located MB instances. Thus, each LD maintains the real-time capacity of its monitoring MBs. Also, LDs are grouped into "Collaborative Load Distributor Groups", where they exchange MB load statistics and distribute load to each other within the group.

Load Distributing Controller (LDC) conducts centralized scheduling for load distributing. It collects the traffic and MB load statistics periodically from the LDs and decides how LDs are grouped as the "Collaborative Load Distributor Group". Besides, LDC also performs policy updating for flow classification as well as static distributing.

B. Load Distributing Strategy

MB Load distributing is performed in per-flow granularity, that is, packets of the same flow should be forwarded to the same MB. Therefore, each incoming packet is firstly inspected in header (e.g. 5 tuples) to see whether it belongs to an existing flow in the session table. If so, an entry in the session table will be searched out, which contains a MB instance id indicating where to steer the packets of this flow. If the incoming packet belongs to a new flow, it will be further classified by flow classification module as either "normal" or "bursty". Different load distributing strategy will be then conducted accordingly.

Static distributing, is conducted for "Normal flows". Normal flows have less impact to traffic spikes and usually maintain a stable amount of traffic rate during their lifetime. Therefore, they are statically assigned to appropriate MBs and provisioned with enough MB capacity. The static distributing policies, which specify all static flow assignments, are generated by the centralized controller. The policies are optimized based on physical topology, link bandwidth, and other static network properties, trying to minimize average communication overhead. The information above is periodically collected and updated when there is modification in topology, MB location and capacity. Recently, there are also works addressing this topic. In [4], each VM to MB path is assigned with corresponding costs, and traffic traversing the same MB is split into fractions corresponding to different MB instances. The traffic distributing task is then formulized as a linear programming to calculate minimum value of the overall cost.

Dynamic collaborative distributing, on the other hand, copes with the bursty flows by real-time load monitoring and flow steering. Temporary flows account for large proportion of the bursts, so they are possible to be scattered among more MB instances. Each LD conducts real-time load monitoring for a number of MBs. In order to share the load statistics, DPs are grouped into separate "collaborative load distributing groups". In the group, DPs are able to

exchange load statistics and outsource load to others. The grouping result is generated and periodically updated by centralized controller. The steering decision of dynamic collaborative distributing is to direct new arrival bursty flows to the least loaded MB instance. Therefore, it makes better utilization of overall MB capacity and helps to eliminate bottleneck. Traffic spikes of different tasks have short durations and they are likely to interleave with each other, in favor of resource multiplexing among multiple flows.

After the MB instance is determined for a new flow, the 5-tuples of flow and the id of designated MB will then be stored in a session entry. Subsequent packets of an existing flow will match the same session entry and be forwarded to the MB associated with it.

The hybrid load distributing strategy guarantees low latency for normal flows and relieve the bursty pressure. The evaluation result in section 5 shows that, the differentiation of flows will make better utilization of overall MB capacity and dramatically reduce the probability of MBs being overwhelmed by load spikes. Also, it eliminates unnecessary flow redirection and maintains low flow latencies.

IV. IMPLEMENTATION DISCUSSIONS

A. Load Distributing Control Plane and Data Plane

To accomplish real-time load distributing, especially for spike flows, the system should be able to achieve both fast load distributing decision and optimized load assignment. Therefore, the control plane and data plane should be carefully designed to support traffic burst with intensive flow set up rate. Aster*x [12] proposed a framework to assign flows by setting up flow entries reactively. And [13] accomplished further improvement by introducing wildcard flow entries and provide more efficient policy implementation and updating. However, when dealing with bursty traffic, the inevitable latency of reactive solutions will end in high performance penalty, which is fatal for real-time load distributing. Therefore, we propose a proactive solution and decouple the fine-grained operations from coarse-grained operations.

In the hierarchical architecture, the lower layer LDs are responsible for fine-grained operations, including packet classifying, forwarding, load monitoring and load statistics exchanging. On the other hand, the upper layer LDC conducts coarse-grained periodical policy generation and updating. Therefore, the high complexity, time-consuming higher layer computation will not affect the real-time operations of the lower layer, which ensures the scalability of the entire system with low overhead.

B. Grouping Strategy

The grouping strategy of LDC determines how Load Distributors share their load statistics and capacity. Basically, the larger the Load Distributor Group is, the more available capacity it provides. However, larger Load Distributor Group also introduces more load statistics exchanging overhead. Therefore, the grouping algorithm largely depends on the datacenter network conditions. For instance, inter-rack traffic overhead in some datacenters is prohibitively high. In this case, the grouping could be performed separately for Load Distributors in each server rack. Currently, a simple global grouping algorithm is adopted in our design. There are two parameters for the algorithm: the ***capacity threshold*** specifies the expected average ratio of available processing capacity for each group; the ***maximum group size*** prevents getting bulky groups, thus limits the intra-group communication overhead. In the grouping algorithm, all the LDs are initially treated as a size-1 group and sorted by their capacity (the total capacity of MBs the LD monitors). If the LD with least resource is below ***capacity threshold***, it will be grouped with the most capable one. If the size of new group stays below ***maximum group size***, it will continue the grouping circulation with other LDs, otherwise it will be excluded from the following grouping. Evaluation result shows that our algorithm is adequate for effective LD collaboration.

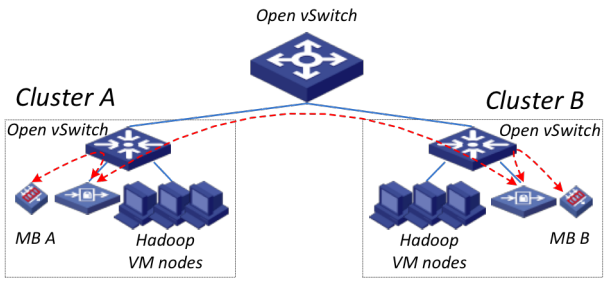

Figure 4. Prototype System Implementation

V. EVALUATION

A. Flow Classification Performance

We first evaluated the performance of bursty flow classification. The flow classification in LD consists of a packet classification module for tuple signatures and a string matching module for payload signatures. The tests include 4 traces from popular big data applications, i.e. “Spark- π calculation” (51 flows), “Hadoop-sort” (40 flows), “Shark-count (608 flows)” and “Impala-count” (75 flows). Flows with higher rate than 1Mbps are manually tagged as bursty flows, and compared against the output of the flow classification module. Table II shows the result for each trace measured by *precision* and *recall*.

It is shown that, the precision of all the traces are higher than 90%, which means that at least 90% of the flows classified as bursty are correctly identified. In addition, the recalls achieve 100% for all the traces, meaning that our classification method guarantees a zero misclassification for bursty flows.

B. Load Distributing Performance

To verify the effectiveness of hybrid distributing strategy, we built a prototype system and compare hybrid load

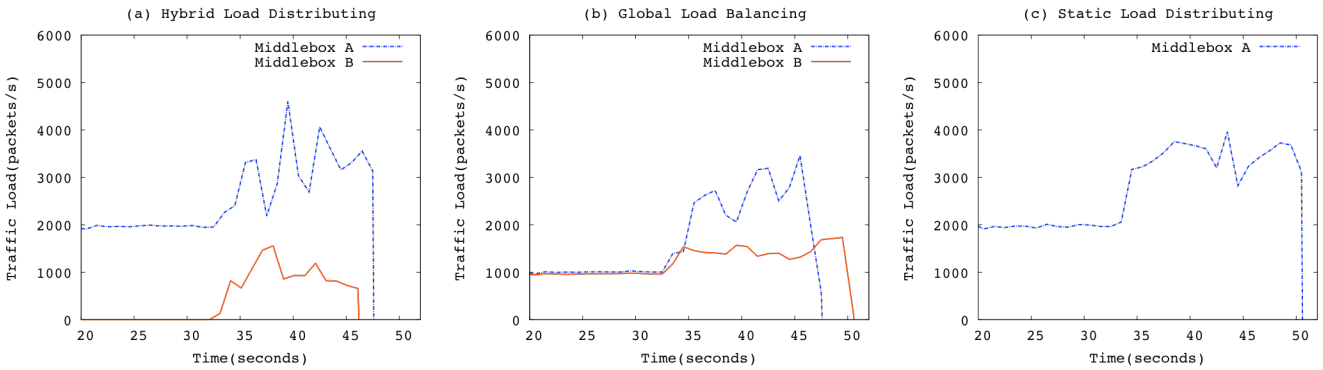

Figure 5. Comparison of Three Load Distributing Strategies on Prototype System

distributing with two common load distributing strategies, i.e. static load distributing and global load balancing.

Figure 4 shows the topology of our prototype system. It is implemented in 2 HP Z220 hosts, each with a 4-core Intel Xeon E3-1225 processor and 10G memory. Each host contains a total of 5 VMs, including a cluster of 3 nodes, 1 middlebox and 1 Load Distributor. The VMs are interconnected by Open vSwitch within the host. A third host running Open vSwitch acts as a core switch, interconnecting the two clusters. The maximum bandwidth of the core switch is configured by QoS to simulate the overhead of load outsourcing.

To emulate the situation that MB might be overwhelmed under high load pressure, we provision MB A with capacity of ~3800 packets per second, which is lower than the peak throughput of task traffic. Note that this capacity is not a hard limitation since MB processing cost of each packet is not unified. MB B does not process local traffic load, and possesses spare capacity. In our test, we only run "Hadoop Sort" on Cluster A and repeated the task with each of three distributing strategies, in order to compare their flow distributing effects. For simplicity, it is assumed that all the flows need to traverse its local LD (i.e. LD in Cluster A) and then should be steered to either MB A or MB B according to the distributing strategy.

For static load distributing, LD in Cluster A distributes the incoming load to MB A and do not outsource any traffic. LDs with strategy of global load balancing, though, steer each flow to the least loaded instance of the two MBs.

Table II. Evaluation of flow classification module

Trace	Precision	Recall
Hadoop-sort	95.4%	100.0%
Impala-count	92.0%	100.0%
Shark-count	90.0%	100.0%
Spark-π calculation	100.0%	100.0%

For hybrid load distributing strategy, LD in Cluster A only outsources the bursts to MB B when there is insufficient local capacity.

Figure 5 shows the load output of middlebox A and B. Among three solutions, hybrid load distributing achieves shortest task execution time (47.8 seconds). As shown in Figure 5(a), the Load Distributor in cluster A outsources much of the exceeding bursty flows to MB B, which eliminates the capacity bottleneck. Compared with static load distributing in Figure 5(c), hybrid load distributing reduces the execution time by 3 seconds. In Figure 5(b), global load balancing also outsourced load from MB A to MB B. However, without differentiating flows, global load balancing redirected both ordinary and spike traffic, introducing extra latency overhead and resulting longer execution time than hybrid distributing.

The study above shows that the hybrid distributing strategy is superior to static load distributing and global load balancing in terms of time efficiency.

C. Large Scale Simulation

We also evaluate our solution for large scale networks through simulation. The simulation is implemented by python, and mainly aimed at studying how static load distributing, global load balancing and hybrid load distributing perform in terms of resource efficiency. During the simulation, we used the term "logic round", to simulate time-related processes, such as traffic fluctuation, flow establishment and vanish. The simulating system consists of 600 middleboxes and 30LDs, with each LD monitoring 20 middleboxes and distributing the flows according to the real-time load statistics.

1) Traffic Generation

To generate both normal flows and bursty flows for each MB, we use several parameters to specify the traffic generator and control the pattern of generated flows. Each parameter acts as a random variable for each logical round, controlling the resulting traffic patterns. Some of the parameters are listed in Table III.

2) Middlebox Capacity Calculation

Table III. Parameters of Different Traffic Generators

	Normal flow Generator	Bursty flow Generator
Flow duration	Long	Short
New flow number	Low	High
Burst probability	Low	High
Average burst amplitude	Low	High
Burst amplitude variance	Low	High

The capacity of each MB is then calculated by simulator based on flow rates of both normal flows and bursty flows. For each middlebox MB_i, the $Capacity_i$ is defined by following formula:

$$Capacity_i = \max(PersistentTrafficRate_i) + r \times max(TemporaryTrafficRate_i)$$

Normal flows, according to our strategy, should be forwarded to its default MB. Therefore normal flows are regarded as partial workload of its corresponding MB. For bursty flows, pre-defined rate *r* is used to represent the extent of resource over-provision for traffic bursts.

It is worth noting that *r* is always less than 0.5 since bursts are sparse over time and traffic peaks tends to interleave with each other. Moreover, higher over-provision rate also brings about higher costs and lower resource utilization.

3) Evaluating Results

Overall, the simulator compares three load distributing strategies in terms of "packet loss rate" and also compare with different over-provision rate. Each load distributing strategy runs 200 testing traces with *r* ranging from 0.0 to 0.4. And for hybrid load distributing, the ***maximum group size*** is set to 5.

From the evaluating results in Figure 6, it is shown that the proposed hybrid load distributing solution achieves far less loss rate than static load distributing, which is less than 1% in all cases. Compared with global load balancing, hybrid load distributing is proved to achieve approximately optimal load distributing, with less overhead of redirecting flows. Meanwhile, as VM capacity declines, there is only minor increase in the packet loss rate. Therefore, the evaluation results showed that our solution is beneficial to reducing packet loss with less resource provision.

VI. CONCLUSION

In summary, we initially proposed hybrid load distributing solution, which differentiates normal flows from bursty flows and conducts load distributing strategies accordingly. Evaluations with prototype system have proved that hybrid load distributing can distribute the exceeding spike traffic with better time efficiency. Moreover, large-scale simulation results showed that the hybrid load distributing is able to achieve approximately optimal load distributing while dramatically reducing the packet loss with low over-provision rate.

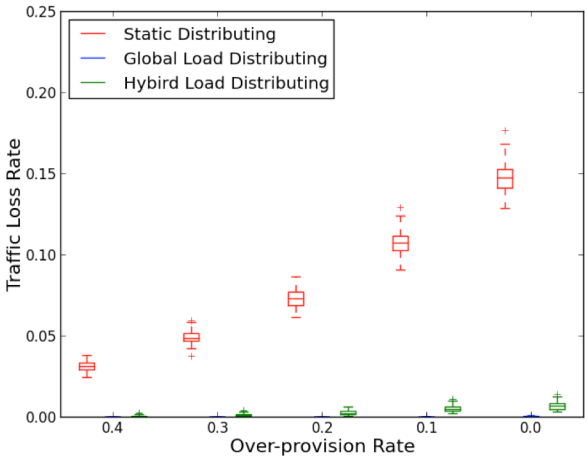

Figure 6. Comparison of three load distributing strategies

ACKNOWLEDGMENT

This work has been supported by 2013 IBM Global Shared University Research (SUR)

REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, et al. "Data center tcp (dctcp)". In Proc. of SIGCOMM, 2010.
[2] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service". In Proc. of SIGCOMM, 2012.
[3] Z. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, *M. Yu*, "SIMPLE-fying Middlebox Policy Enforcement Using SDN". In Proc. of SIGCOMM, 2013.
[4] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. TR. Stratos: a network-aware orchestration layer for middleboxes in the cloud, 2013.
[5] S. Kandula, S. Sengupta, A. Greenberg, "The Nature of Datacenter Traffic: Measurements & Analysis", In Proc. of IMC, 2009
[6] T. Benson, A. Akella, D.A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild", In Proc. of IMC, 2010
[7] Apache Hadoop. http://hadoop.apache.org/
[8] Apache Spark. http://spark.apache.org/
[9] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System". MSST, 2010
[10] A. Metwally, D. Agrawal, and A. Abbadi, "An Integrated Efficient Solution for Computing Frequent and Top-k Elements in Data Streams," ACM Transactions on Database Systems, vol. 31, no. 3, pp. 1095–1133, 2006
[11] Y. Shao, B. Yang, J. Jang, Y. Xue and J. Li, "Emilie: Enhance the Power of Traffic Identification", in Proc. of ICNC, 2014.
[12] N. Handigol, M. Flajslik, S. Seetharaman, Nick McKeown, "Aster*x: Load-Balancing as a Network Primitive", 2010
[13] R. Wang, D. Butnariu, J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild". In Proc. of Hot-ICE, 2011.
[14] Shark. http://shark.cs.berkeley.edu/
[15] Impala. http://impala.io/