A Novel Approach to Netflow Monitoring in Data Center Networks

Chaitanya Balantrapu, Anupama Potluri, Nirmoy Das
School of Computer and Information Sciences
University of Hyderabad, Hyderabad, India
Email: bnsk.uohyd@gmail.com, apcs@uohyd.ernet.in, nirmoy.aiemd@gmail.com

***Abstract*—Scalable netflow monitoring is one of the challenges in data center networks which have thousands of Virtual Machines (VMs) running on hundreds of physical servers. The typical architecture for netflow monitoring using a centralized netflow collector is not scalable in data center networks. The solution has been to use a cluster of netflow collectors and include a load balancing element in the network. In this paper, we propose a different architecture wherein the netflow clients directly write to a scalable NoSQL database such as Cassandra. We modified Open vSwitch (OVS) to directly write netflow packets to Cassandra. We extended *ntop-ng*, a free open source netflow collector software, to store netflow records in Cassandra and compared its performance with that of OVS, a netflow client, writing to Cassandra directly. We found that using Cassandra from OVS leads to a significant performance improvement over *ntop-ng* in terms of the average number of netflow packets written per second. We modified *ntop-ng* to store the netflow statistics to Cassandra and compared the performance of Cassandra against RRD, which is the default non-volatile storage in *ntop-ng*. We found that Cassandra is more scalable than RRD as the time to store the records increases linearly with number of writes for RRD whereas Cassandra has constant time in our experimental setup.**

I. Introduction

Data center networks consist of thousands or tens of thousands of Virtual Machines (VMs) running on hundreds or thousands of physical servers. The VMs are connected using software switches such as vSwitch of VMWare or Open vSwitch. Network statistics collection and monitoring are essential to management of networks. VMs may be started, migrated to save energy by shutting down physical servers that are lightly loaded or rogue VMs may be shutdown based on the statistics collected. Data analytics software on top of Netflow [4] records that alerts data center administrators of these events is also of high importance today.

The typical architecture for netflow monitoring so far has always been to use a centralized collector that collects netflow records from all the netflow clients in various network elements. Software switches such as Open vSwitch [9] are also netflow clients. Typically, they are configured with the IP address and port of the netflow collector. The netflow collector software collects the records, may compute statistics from the records and stores them in a non-volatile storage for further use. When dealing with the thousands of netflow clients as in the case of data center networks, the centralized collector has to be highly scalable. This is difficult to achieve with a single server. An obvious solution would be to have a cluster of collectors with a load balancer to balance the traffic between them. There are two issues with this solution. Firstly, it replicates the solutions that have been already implemented in scalable NoSQL databases such as Cassandra [3] etc. Secondly, despite load balancing, the amount of information that needs to be stored becomes unmanageable if historical information needs to be maintained. Thus, there is a definite need to have a highly scalable database. Storing the netflow records to a scalable database can help overcome these limitations.

In this paper, we discuss our solution to the problem of scalable netflow monitoring in data center networks. Our contributions can be stated as follows:

1) We extended a well-known netflow collector, *ntop-ng* [8], to store netflow records and/or statistics to Cassandra, a NoSQL database. We experimented to understand the performance of Cassandra as compared to Round-Robin Database (RRD) with *rrdtool* [10] which is the default storage in *ntop-ng*.
2) We extended Open vSwitch to store netflow records directly to Cassandra instead of sending them to a netflow collector. We compared the performance of *ntop-ng* and Open vSwitch for the same netflow traffic to determine the speed up achieved.

The rest of this paper is organized as follows: we review a couple of papers that deal with scalable network monitoring in Section II. We present our architecture for netflow monitoring and modifications to *ntop-ng* and Open vSwitch in Section III. We present our experimental setup and initial results in Section IV. We conclude with Section V.

II. Related Work

In [13], the authors propose a scalable netflow collection for cloud data centers. The EMC2 architecture consists of two threads in the collector – one for netflow and the second for sflow records. The collector uses flat files to store the records obtained. If the communicating VMs are all present within the same data center,the netflow records will come from both VMs for the same data. Hence, EMC2 does de-duplication to eliminate such duplicate records before storing the data. EMC2 is a centralized netflow collector similar to *ntop-ng* but includes de-duplication. However, its storage model of flat files is not scalable as it is limited by the file system being used.

In [12], the authors propose scalable network traffic measurement and analysis using Hadoop [1], an open-source implementation of MapReduce [5] and the Hadoop distributed file system (HDFS). Hadoop primarily supports text fields or binary format data in a sequence file format. In [12], packet trace files generated by **libpcap** tools such as *tcpdump* are used as input to Hadoop for offline processing. Since this format is not compatible with the Hadoop format for binary data, the authors build a Hadoop API to handle this data. While text data has an end-of-the-record marker in a carriage return, packet data has no such marker. Packet data is also not of fixed length. Thus, differentiating packet records in Hadoop is a challenge. The authors propose the use of the timestamp field and the fact that consecutive packets do not differ too much in their timestamp to distinguish between packets in a HDFS block. They propose a heuristic algorithm using this property which is used by the *map* jobs to parallelize the packet analysis jobs.

III. Scalable NetFlow Monitoring using Cassandra

EMC2 is a netflow collector that stores netflow records in flat files instead of statistics in RRD as done by *ntop-ng*. In [12], standard **libpcap** data stored in flat files on different systems is read and converted to Hadoop-compatible format. This is not netflow data but standard packet trace information. In this paper, we propose the extension of netflow clients to store netflow records directly to the scalable database, Cassandra, using simple data models. This is different from the earlier solutions in two respects: firstly, it is not the netflow collector that stores data into flat files/databases but the netflow client. Secondly, it is an online algorithm unlike in [12], where they read the offline data from flat files and store this data in Hadoop.

We look at scalability in netflow monitoring from two perspectives – scalability in terms of storage time and the handling of the amount of netflow traffic generated in data center networks.

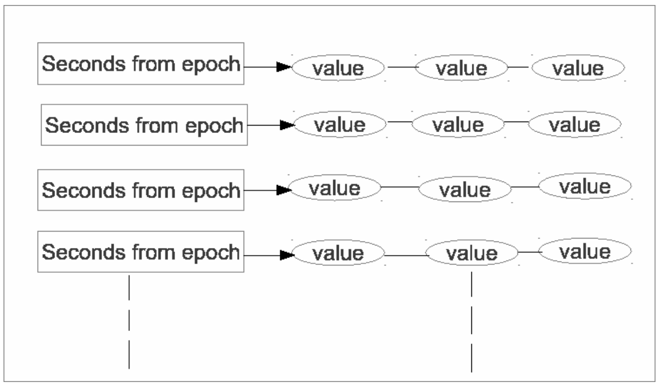

Fig. 1. Cassandra Data Model for Storing NetFlow Records

A. Scalability in Storage Time:

When we look at typical netflow collectors such as the EMC2 [13] or *ntop-ng* which uses RRD or other netflow collectors such as *nfdump* [7] etc., none of them have a scalable storage capacity. While RRD is built specially for time series data, it has limited storage space. If historical information needs to be stored and data analytics software needs to use this data to predict network performance of different tenants of the data center, RRD cannot be the database of choice. We propose instead the use of Cassandra and extend *ntop-ng* to achieve this.

ntop-ng computes statistics from the netflow records received and stores these in RRD. We store the exact same information in Cassandra and compare the performance of these two databases. Since the rate of generation of netflow records will be very high in a data center network, computing statistics and then storing the information may lead to loss of scalability. We propose storing raw netflow records into Cassandra without any processing in *ntop-ng*. The data model we have used to store the raw netflow records in Cassandra is given in Fig. 1.

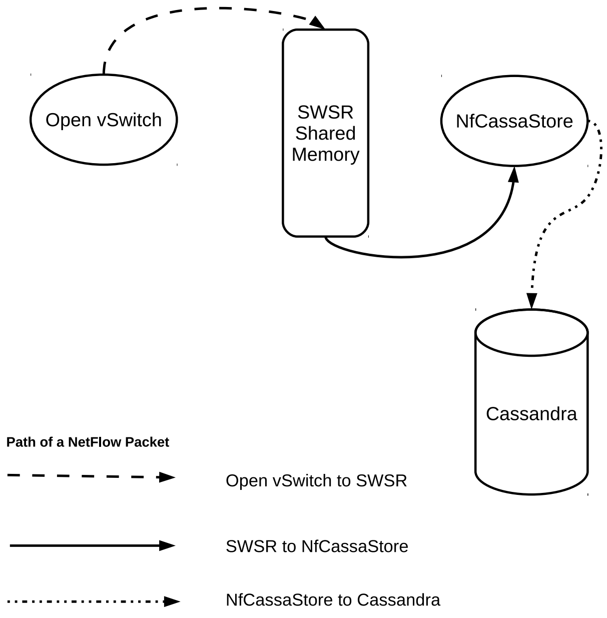

Fig. 2. Architecture of OVS storing netflow records into Cassandra

B. Scalability in NetFlow Record Collection:

While netflow collectors that use a scalable database to store information overcome the issue of scalability in storage, they may still not be able to handle the rate of netflow traffic destined to them. Centralized servers suffer from limitations of the socket interface they are generally using to receive netflow records. The solution then is to use netflow collector clusters to achieve scalability. This leads to replication of inherent scalability features of NoSQL databases in netflow collectors.

To overcome the above limitations, we propose that netflow clients store their netflow records directly in a scalable database. These databases already consist of clusters and are built to handle large amounts of data at high speeds and also do not have a limitation of storage space. Another reason to choose a scalable database is the fast retrieval of data needed for a monitoring tool which uses data analytics software. This is well supported by the NoSQL databases. We chose Cassandra for our implementation since it has been proven to be better than *Redis* and *Hbase* by a test done by DataStax [2].

To determine the difference in performance between netflow clients writing raw netflow records directly to Cassandra and a centralized netflow collector doing the same, we chose one netflow client – Open vSwitch – and extended it to store the netflow records to a Single Writer Single Reader (SWSR) shared memory. A daemon (*NfCassastore* in Fig. 2) reads this shared memory and stores them in the Cassandra database. This prevents a tight coupling of a specific database with OVS at the cost of a slight loss of performance. The architecture of our proposed solution is shown in Fig. 2. *NfCassastore* will not become a bottleneck as it is limited to data coming only from the *vswitch* of that physical machine. We can also increase the size of the shared memory segment to be sufficient to handle the rate of traffic of a single *vswitch* and therefore this is not a limitation of our architecture.

IV. Experimental Setup and Discussion of Results

The experimental setup used is shown in Fig. 3. We have one VM on a physical system communicating with its corresponding VM in another physical system. The VMs are connected via Open vSwitch to the physical Gigabit Ethernet card on the physical system. The POSIX shared memory segment used for the experimentation is 64KB in size which is the same as the TCP socket buffer size used by *ntop-ng*. The components used and the details are given in Table I. In each VM, we run 125 *netperf* [6] clients. Each *netperf* client generates TCP-CRR (Connection Request-Response) transaction. This consists of 10 TCP packets – three each for open and close connection and two for the data and its ack. Each such transaction generates two netflow records. Since Open vSwitch generates a netflow packet to the netflow collector when 30 netflow records are collected, 15 TCP-CRR transactions generate a netflow packet. In our experiment, Open vSwitch generates 220-250 netflow packets/second, each containing 30 netflow records.

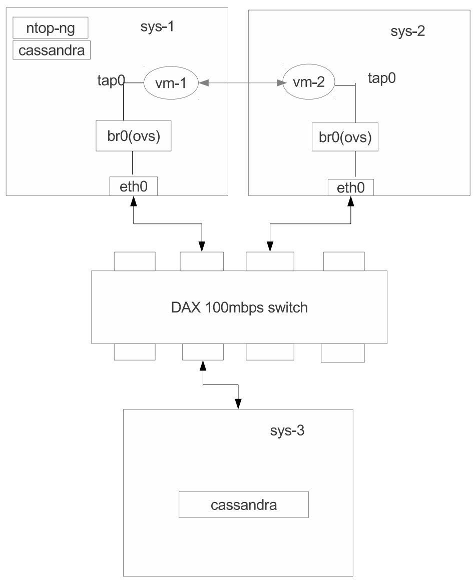

Fig. 3. Testbed used for the experimentation

TABLE I. Components of our testbed

Component Name	Details/Version
CPU	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz.
Operating System	OpenSuse 12.2
Linux Kernel	3.4
KVM	kvm-1.1.1-1.8.1
QEMU Emulator version	1.1.1
Open vSwitch	openvswitch-1.9.0(LTS)
netperf	netperf-2.6
ntop-ng	ntop-ng 1.0.1
Cassandra	Apache Cassandra-2.0.0
RRD	RRD version 1.4.7
nprobe	nprobe_6.14.130821_svn3645

A. ntop-ng *performance comparison with RRD and Cassandra:*

Fig. 4 shows the time taken to store the computed statistics in RRD and Cassandra. We compute the time by using the system call *gettimeofday* before and after the calls to RRD and Cassandra write operations. We ran the *netperf* clients for 60 minutes. A thread runs every minute and stores the computed statistics for that minute into the database. As the number of records written to RRD increases, the time taken by it to store similar amount of data, i.e., statistics/min increases. As reported in [11], RRD opens, writes and closes the file each time a write has to be done. Cassandra, on the other hand, is using *memtables* before commit and therefore is much faster. We see that when Cassandra is on a remote system, the time increases a little. Since our systems were isolated from the rest of the LAN and there was no other traffic the difference in time between Cassandra server on the same system as *ntop-ng* and on a remote system is not very high. This is the ideal difference and represents the cost of the socket interface and the limitation of the switch connecting the systems.

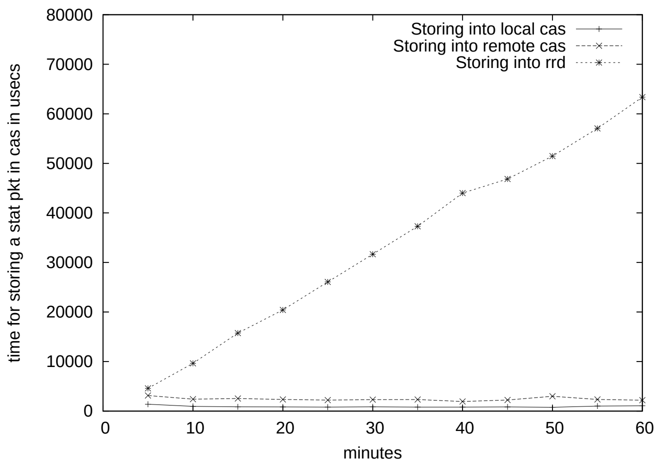

Fig. 4. Comparison of time taken by *ntop-ng* to store statistics in RRD and Cassandra (local and remote locations)

B. ntop-ng *versus Open vSwitch performance with Cassandra:*

Fig. 6 shows the number of netflow records stored in Cassandra from *ntop-ng* and Open vSwitch. The data model used in Cassandra is that each second from *epoch time* is the row key. Each netflow packet received by Cassandra (that consists of 30 netflow records) is stored as a different column. The netflow packets received every second need not

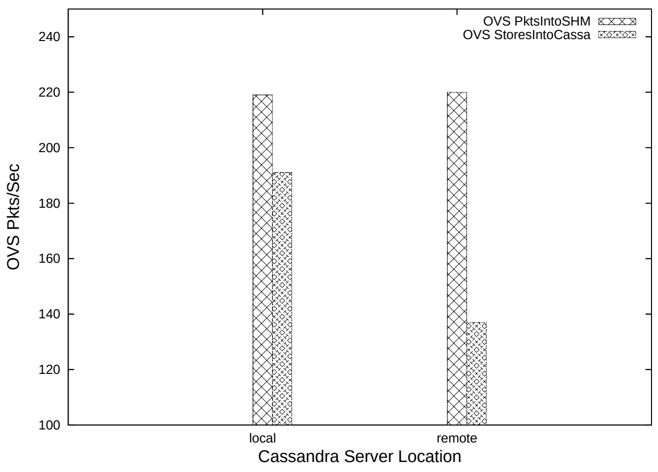

Fig. 5. Rate of Netflow Packets (1 Netflow Pkt = 30 Netflow Records) Stored/Minute by OVS into Local and Remote Cassandra Servers

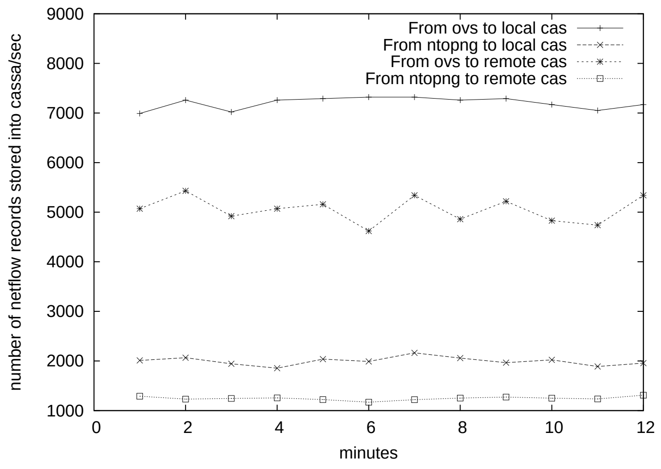

Fig. 6. Comparison of average number of Netflow Records stored by *ntop-ng* and OVS per minute in Cassandra

be constant. We average the packets received over a minute. We ran the experiment for 15 minutes and show the average number of netflow packets received every minute. The number of netflow packets stored per minute when OVS stores directly to Cassandra is significantly higher whether the Cassandra server is co-located on the same physical system as OVS/*ntop-ng* or is on a remote system.

When we tested OVS with a remote Cassandra server, we found that the performance dropped by 25%. We measured the average number of packets/second written to the shared memory by OVS and the number of stores into Cassandra server in that time. We ran the experiment three times for 12 minutes each and calculated the average across all the runs. We find that 87% of the records in the shared memory are stored in a local Cassandra server whereas with a remote Cassandra server only 62% of the records are stored. This is shown in Fig. 5. The rate at which the data is written to Cassandra is limited by the socket interface limitations.

V. Conclusion and Future Work

We have extended *ntop-ng*, a netflow collector and Open vSwitch (OVS), a netflow client, to store data to Cassandra, a scalable NoSQL database. We generated a high rate of netflow traffic by using 125 *netperf* clients per VM per physical system. Each *netperf* client generates TCP-CRR transactions. We have compared the time taken by *ntop-ng* and OVS when storing raw netflow data to Cassandra. We found that the number of netflow packets stored per minute by OVS is significantly higher than *ntop-ng*. We also found that the time taken to store statistics into Cassandra is constant whereas it increases linearly in RRD, which is the default database in *ntop-ng*. We conclude that our approach of netflow clients storing data directly to Cassandra is promising for high speed traffic.

In future, we plan to use a cluster of Cassandra servers and experiment with a much larger testbed with many VMs in many physical systems and the OVS in each physical system storing the netflow records in the Cassandra cluster. We also plan to extend our work to store the netflow records in HDFS and compare its performance with Cassandra.

Acknowledgment

The authors would like to thank Aravinda Potluri, Managing Director, Oneconvergence Devices India for her initial proposal of extending *ntop* to use Cassandra for netflow monitoring in data center networks. We also acknowledge A.Geetha Sowjanya for her help in the experimental setup.

References

[1] “Apache hadoop and hbase,” http://www.slideshare.net/cloudera/sf-nosql2011/58.

[2] “Cassandra performance review,” http://www.datastax.com/dev/blog/2012-in-review-performance.

[3] “Cassandra, the ideal foundation for big data.” http://www.datastax.com/what-we-offer/products-services/datastax-enterprise/apache-cassandra.

[4] “Cisco IOS NetFlow,” http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.

[5] “Mapreduce,” http://en.wikipedia.org/wiki/MapReduce.

[6] “Netperf homepage,” http://www.netperf.org/netperf/.

[7] “nfdump home page,” http://nfdump.sourceforge.net/.

[8] “ntop-ng: High-speed web-based traffic analysis and flow collection.” http://www.ntop.org/products/ntop/.

[9] “Open vswitch: A open virtual switch,” http://openvswitch.org/.

[10] “Rrdtool home page,” http://oss.oetiker.ch/rrdtool/.

[11] L. Deri, S. Mainardi, and F. Fusco, “tsdb: a compressed database for time series,” in *Proceedings of the 4th international conference on Traffic Monitoring and Analysis*, ser. TMA’12. Springer-Verlag, 2012, pp. 143–156.

[12] Y. Lee and Y. Lee, “Toward scalable internet traffic measurement and analysis with hadoop,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 5–13, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2427036.2427038

[13] V. Mann, A. Vishnoi, and S. Bidkar, “Living on the edge: Monitoring network flows at the edge in cloud data centers,” in *Fifth International Conference on Communication Systems and Networks, COMSNETS 2013*, 2013.