Toward Scalable Internet Traffic Measurement and Analysis with Hadoop

Yeonhee Lee
Dept. of Computer Engineering
Chungnam National University, Korea
yhlee06@cnu.ac.kr

Youngseok Lee
Dept. of Computer Engineering
Chungnam National University, Korea
lee@cnu.ac.kr

ABSTRACT

Internet traffic measurement and analysis has long been used to characterize network usage and user behaviors, but faces the problem of scalability under the explosive growth of Internet traffic and high-speed access. Scalable Internet traffic measurement and analysis is difficult because a large data set requires matching computing and storage resources. Hadoop, an open-source computing platform of MapReduce and a distributed file system, has become a popular infrastructure for massive data analytics because it facilitates scalable data processing and storage services on a distributed computing system consisting of commodity hardware. In this paper, we present a Hadoop-based traffic monitoring system that performs IP, TCP, HTTP, and NetFlow analysis of multi-terabytes of Internet traffic in a scalable manner. From experiments with a 200-node testbed, we achieved 14 Gbps throughput for 5 TB files with IP and HTTP-layer analysis MapReduce jobs. We also explain the performance issues related with traffic analysis MapReduce jobs.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Management, Network Monitoring

Keywords

Hadoop, Hive, MapReduce, NetFlow, pcap, packet, traffic measurement, analysis

1. INTRODUCTION

We live in an era of "big data" produced by skyrocketing Internet population and extensive data-intensive applications. According to Cisco [1], for example, global IP traffic has multiplied eightfold over the past five years and annual global IP traffic will exceed 1.3 zettabytes by the end of 2016. It is also reported that the aggregated traffic volume in Japan has been doubling roughly every two years since 2005 [2]. As the number of network elements, such as routers, switches, and user devices, has increased and their performance has improved rapidly, it has become more and more difficult for Internet Service Providers (ISPs) to collect and analyze efficiently a large data set of raw packet dumps, flow records, activity logs, or SNMP MIBs for accounting, management, and security.

To satisfy demands for the deep analysis of ever-growing Internet traffic data, ISPs need a traffic measurement and analysis system where the computing and storage resources can be scaled out. Google has shown that search engines can easily scale out with MapReduce and GFS [3, 4]. MapReduce allows users to harness tens of thousands of commodity machines in parallel to process massive amounts of data in a distributed manner simply defining map and reduce functions. Apache Hadoop [5], sponsored by Yahoo!, is an open-source distributed computing framework implemented in Java to provide with MapReduce as the programming model and the Hadoop Distributed File System (HDFS) as its distributed file system. Hadoop offers fault-tolerant computing and storage mechanisms for the large-scale cluster environemt. Hadoop was orginally designed for batch-oriented processing jobs, such as creating web page indices or analyzing log data. Currently, Hadoop is widely used by Yahoo!, Facebook, IBM, Netflix, and Twitter to develop and execute large-scale analytics or applications for huge data sets [6].

Generally, it is not straightforward to perform network management or business intelligence analytics on large amounts of data: traffic classification of packet and netflow files; quick investigation of anomalies such as global Internet worm outbreaks or DDoS attacks; long-term network trends or user behaviors. For instance, the volume of traffic data captured at a 10 Gbps directional link of 50% utilization becomes 2.3 TB per hour. However, there is no analysis tool that can afford this amount of data at once. It is then expected that the major advantage of using Hadoop to measure Internet traffic is the scale-out feature, which improves the analysis performance and storage capacity in proportion to the computing and storage resources with commodity hardware. Hadoop for its scalability in storage and computing power is a suitable platform for Internet traffic measurement and analysis but brings about several research issues.

In this work we develop a Hadoop-based scalable Internet traffic measurement and analysis system that can manage packets and NetFlow data on HDFS. The challenges of applying Hadoop to Internet measurement and analysis are 1) to parallelize MapReduce I/O of packet dumps and netflow records in HDFS-aware manner, 2) to devise traffic analysis algorithms especially for TCP flows dispersed in HDFS, and 3) to design and implementation an integrated Hadoop-based Internet traffic monitoring and analysis system practically useful to operators and researchers. To this end, we propose a binary input format for reading packet and NetFlow records concurrently in HDFS. Then, we present MapReduce analysis algorithms for NetFlow, IP, TCP, and HTTP traffic. In particular, we elucidate how to analyze efficiently the TCP performance metrics in MapReduce in the distributed computing environment. Finally, we create a web-based agile traffic warehousing system using Hive [7] which is useful for creating versatile operational analysis queries on massive amounts of Internet traffic data. We also explain how to increase the performance of Hadoop when running traffic analysis MapReduce jobs on a large-scale cluster. From experiments on a large-scale Hadoop testbed consisting of 200 nodes, we have achieved 14 Gbps throughput for 5 TB packet files with IP and HTTP-layer analysis MapReduce jobs.

The main contribution of our work is twofold. First, this work presents a Hadoop-based tool that offers not only Internet traffic measurement utilities to network reseachers and operators but also various analysis capabilities on a huge amount of packet data to network researchers and analysts. Second, this work establishes a guideline for researchers on how to write network measurement applications with MapReduce and adopt high-level data flow language, such as Hive and Pig.

The paper is organized as follows. In Section 2, we describe the related work on traffic measurement and analysis. The Hadoop-based traffic measurement and analysis system is explained in Section 3, and the experimental results are presented in Section 4. Finally, Section 5 concludes this paper.

2. RELATED WORK

Over the past few decades, a lot of tools have been developed and widely used for Internet traffic monitoring. Tcpdump [8] is the most popular tool for capturing and analyzing packet traces with `libpcap`. Wireshark [9] is a popular traffic analyzer that offers user-friendly graphic interfaces and statistics functions. CoralReef [10], developed by CAIDA, provides flexible traffic capture, analysis, and report functions. Snort [11] is an open source signature-based intrusion detection tool designed to support real-time analysis. Bro [12], which is a network security monitoring system, has been extended to support the cluster environment [13]. However, it provides only independent packet processing at each node for live packet streaming, so it cannot analyze a large file in the cluster filesystem. Tstat [14] is a passive analysis tool which elaborates `tcptrace`, and it offers various analysis capabilities with regard to TCP performance metrics, application classification, and VoIP characteristics. On the other hand, Cisco NetFlow [15] is a well-known flow monitoring format for observing traffic through routers or switches. Many open-source or commercial flow analyzing tools exist, including flow-tools [16], flowscan [17], argus [18], and Peakflow [19]. Yet, in general, the majority Internet traffic measurement and analysis tools run on a single server and they are not capable of coping with a large amount of traffic captured at high-speed links of routers in a scalable manner.

Most MapReduce applications on Hadoop are developed to analyze large text, web, or log files. In our preliminary work [20], we have devised the first packet processing method for Hadoop that analyzes packet trace files in a parallel manner by reading packets across multiple HDFS blocks. Recently, RIPE [21] has announced a similar packet library for Hadoop, but it does not consider the parallel-processing capability of reading packet records from HDFS blocks of a file so that its performance is not scalable and its recovery capability against task failures is not efficient. In this paper, we present a comprehensive Internet traffic analysis system with Hadoop that can quickly process IP packets as well as NetFlow data through scalable MapReduce-based analysis algorithms for large IP, TCP, and HTTP data. We also show that the data warehousing tool Hive is useful for providing an agile and elastic traffic analysis framework.

3. TRAFFIC MEASUREMENT AND ANALYSIS SYSTEM WITH HADOOP

In this section, we describe the components of the traffic measurement and analysis system with Hadoop and traffic analysis MapReduce algorithms. As shown in Fig. 1, our system[1] consists of a traffic collector; new packet input formats; MapReduce analysis algorithms for NetFlow, IP, TCP, and HTTP traffic; and a web-based interface with Hive. Additional user-defined MapReduce algorithms and queries can be extended.

3.1 Traffic collector

The traffic collector receives either IP packet and NetFlow data from probes or trace files on the disk, and writes them to HDFS. NetFlow exported in UDP datagram can be considered as IP packet data. Traffic collection is carried out by a load balancer and HDFS DataNodes. In online traffic monitoring, the load balancer probes packet streams using a high-speed packet capture driver, such as PF_RING and TNAPI [22], and it forwards packets to multiple DataNodes evenly with a flow-level hashing function. Then, HDFS DataNodes capture forwarded packets and write them to HDFS files concurrently. Since disk I/O performance may be a bottleneck to a Hadoop cluster, each DataNode uses a parallel disk I/O function, such as RAID0 (data striping mode), to boost the overall performance of HDFS. Due to the distributed traffic collecting architecture, a traffic collector can achieve the scale-out feature for storing the increased input traffic volume. However, we focus on the offline traffic collection and analysis system because our system currently does not guarantee the end-to-end performance from online traffic collection to real-time traffic analysis, which requires a job scheduling discipline capable of provisioning cluster resources for the given traffic analysis load.

3.2 IP packet and NetFlow reader in Hadoop

In Hadoop, text files are common as an input format because data mining of text files such as web documents or log files is popular. IP packets and NetFlow data are usually stored in the binary format of `libpcap`. Hadoop in itself supports a built-in sequence file format for binary input/output. In order to upload the packet trace files captured by existing probes to HDFS, however, we have to convert them into HDFS-specific sequence files. This procedure will result in the computation overhead of reading every packet record sequentially from a packet trace file and saving each one in the format of the sequence file. Though the sequence file format can be used for online packet collection, it will incur the additional space requirements for the header, sync, record length, and key length fields. Moreover, this sequence file format in HDFS is not compatible with widely used `libpcap` tools, which causes the data lock-in problem. Therefore, it is not efficient to use the built-in sequence file format of HDFS for handling NetFlow and packet trace files. In this work, we developed new Hadoop APIs that can read or write IP packets and NetFlow v5 data in the native `libpcap` format

[1] The source code of our tool is available in [24]

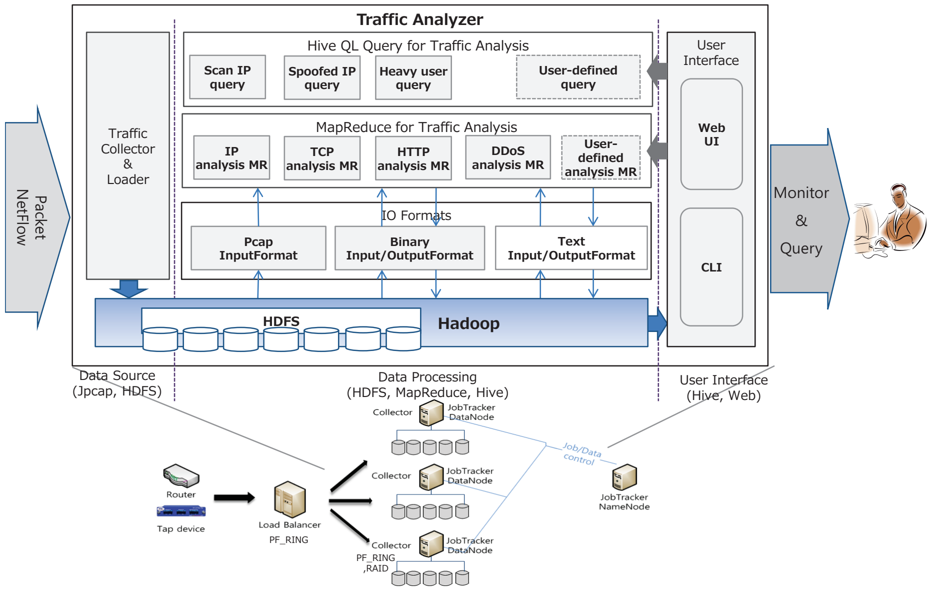


Figure 1: Overview of the traffic measurement and analysis architecture with Hadoop.

on HDFS. The new Hadoop API makes it possible to directly save the `libpcap` streams or files to HDFS and run MapReduce analysis jobs on the given `libpcap` files.

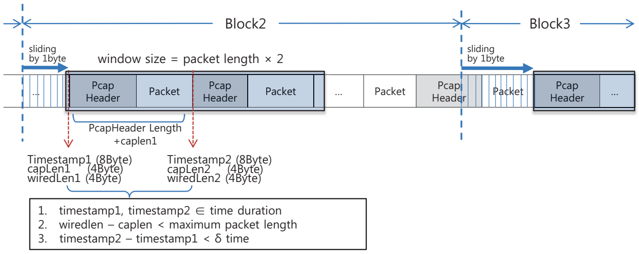


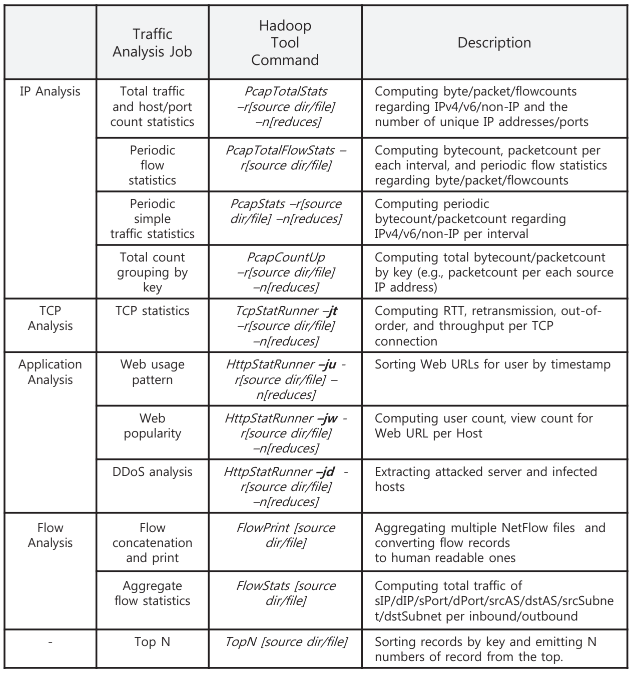
Figure 2: Reading packet records in `libpcap` on HDFS blocks for parallel processing.

When a MapReduce job runs on `libpcap` files in HDFS, each map task reads its assigned HDFS block to parse packet records independently, which is imperative for parallel processing. Since an HDFS block is chunked in a fixed size (e.g., 64MB), a variable-sized packet record is usually located across two consecutive HDFS blocks. In addition, in contrast to the text file including a carriage-return character at the end of each line, there is no distinct mark between two packet records in a `libpcap` file. Therefore, it is difficult for a map task to parse packet records from its HDFS block because of the variable packet size and no explicit packet separator. In this case, a single map task can process a whole `libpcap` file consisting of multiple HDFS blocks in a sequential way, but this file-based processing method, used in RIPE `pcap` [21], is not appropriate for parallel computing. With RIPE `pcap`, each trace should be saved in a sufficiently fine-grained size to fully utilize the map task slots of all cluster nodes. Otherwise, the overall performance will be degraded due to the large file size. Moreover, if a map or reduce task fails, the file-based MapReduce job will roll back to the beginning of a file.

In order for multiple map tasks to read packet records of HDFS blocks in parallel, we propose a heuristic algorithm [20] that can process packet records per block by using the timestamp-based bit pattern of a packet header in `libpcap`. Figure 2 shows how each map task delineates the boundary of a packet record in HDFS. Our assumption is that the timestamp fields of two continuous packets stored in the `libpcap` trace file are similar. When a MapReduce job begins, it invokes multiple map tasks to process their allocated HDFS blocks. Each map task equipped with the packet record-search algorithm considers the first 16 bytes as a candidate for a `libpcap` packet header, reads the related fields of two packets using the caplen field, and verifies each field value of the two packet records. First, the timestamp value should be within the time duration of the captured packet traces. Then, we look into the integrity of the captured length and the wired length values of a packet header, which should be less than the maximum packet length. Third, the timestamp difference between two contiguous packet records should be less than threshold. The packet-record search algorithm moves a fixed-size (e.g., 2× maximum packet size) window by a single byte to find a packet. On top of the heuristic algorithm, we have implemented `PcapInputFormat` as the packet input module in Hadoop, which can manipulate IP packets and NetFlow v5 data in a native manner in HDFS. `BinaryInputFormat` is the input module for managing binary records such as flows and calculating flow statistics from IP packets.

3.3 Network-layer analysis in MapReduce

For network-layer analysis, we have developed IP flow statistics tools in MapReduce, as shown in Fig. 3. Computing IP flow statistics certainly fits the MapReduce framework because it is a simple counting job for the given key and it can be independently performed per HDFS block. With this tool, we can retrieve IP packet and flow records and determine IP flow statistics that are similarly provided by well-known tools [10, 16]. For the packet trace file, we can tally the IP packet statistics, such as byte/packet/IP



	Traffic Analysis Job	Hadoop Tool Command	Description
IP Analysis	Total traffic and host/port count statistics	*PcapTotalStats –r[source dir/file] –n[reduces]*	Computing byte/packet/flowcounts regarding IPv4/v6/non-IP and the number of unique IP addresses/ports
	Periodic flow statistics	*PcapTotalFlowStats –r[source dir/file]*	Computing bytecount, packetcount per each interval, and periodic flow statistics regarding byte/packet/flowcounts
	Periodic simple traffic statistics	*PcapStats –r[source dir/file] –n[reduces]*	Computing periodic bytecount/packetcount regarding IPv4/v6/non-IP per interval
	Total count grouping by key	*PcapCountUp –r[source dir/file] –n[reduces]*	Computing total bytecount/packetcount by key (e.g., packetcount per each source IP address)
TCP Analysis	TCP statistics	*TcpStatRunner **–jt** –r[source dir/file] –n[reduces]*	Computing RTT, retransmission, out-of-order, and throughput per TCP connection
Application Analysis	Web usage pattern	*HttpStatRunner **–ju** -r[source dir/file] –n[reduces]*	Sorting Web URLs for user by timestamp
	Web popularity	*HttpStatRunner **–jw** -r[source dir/file] –n[reduces]*	Computing user count, view count for Web URL per Host
	DDoS analysis	*HttpStatRunner **–jd** -r[source dir/file] –n[reduces]*	Extracting attacked server and infected hosts
Flow Analysis	Flow concatenation and print	*FlowPrint [source dir/file]*	Aggregating multiple NetFlow files and converting flow records to human readable ones
	Aggregate flow statistics	*FlowStats [source dir/file]*	Computing total traffic of sIP/dIP/sPort/dPort/srcAS/dstAS/srcSubnet/dstSubnet per inbound/outbound
-	Top N	*TopN [source dir/file]*	Sorting records by key and emitting N numbers of record from the top.

Figure 3: Overview of IP and NetFlow analysis tools in MapReduce.

flow counting, IP flow statistics, periodic traffic statistics, and top N, as shown in Fig. 3. The IP analysis is mostly an aggregation work that summarizes the corresponding count values associated with IP addresses and port numbers. `PcapTotalFlowStats` periodically calculates flow information, which consists of IP addresses and ports from packet trace files. This analysis requires two MapReduce jobs for flow statistics and aggregated flow information, respectively. `PcapStats` tallies the total byte/packet and bit/packet rate per time window for the given packet trace files. `PcapCountUp` summarizes the total byte/packet count for given packet trace files regardless of any time window. `TopN` finds the most popular statistics, such as top 10 IP or TCP flow information.

3.4 Transport-layer analysis in MapReduce

In contrast to IP analysis, TCP performance metrics such as round-trip time (RTT), retransmission rate, and out-of-order cannot be computed per HDFS block, because the computation of these metrics is not commutative and associative across TCP flow parts on several HDFS blocks. In TCP analysis with MapReduce, we face two challenges: constructing a TCP connection by stitching directional TCP flow parts spread out across multiple HDFS blocks; optimizing the TCP performance metric calculation work in MapReduce.

3.4.1 Constructing a TCP connection by stitching flow parts

Before performing TCP-layer analysis in MapReduce, we have to extract a full TCP connection containing both client-server (C2S) and server-client (S2C) directional flows that are often stored across several HDFS blocks. In order to find a TCP connection, we have to map multiple C2S and S2C directional TCP flow parts on different HDFS blocks into the same reduce task. Each map task associated with a HDFS block pulls out the packet streams of a single TCP connection and passes them to a reduce task that will merge flow parts into a TCP connection and will compute the TCP performance metrics. Figure 4 illustrates how two C2S and S2C directional TCP flows that belong to a single TCP connection, are assigned to the same reducer. If a map task uses a key of five tuples, two directional TCP flows of a TCP connection ① may be mapped to different reduce tasks (①') because the hashing function will assign sourcedestination or destinationsource flows to different domains. Then, neither of the two reduce tasks can carry out accurate TCP performance analysis.

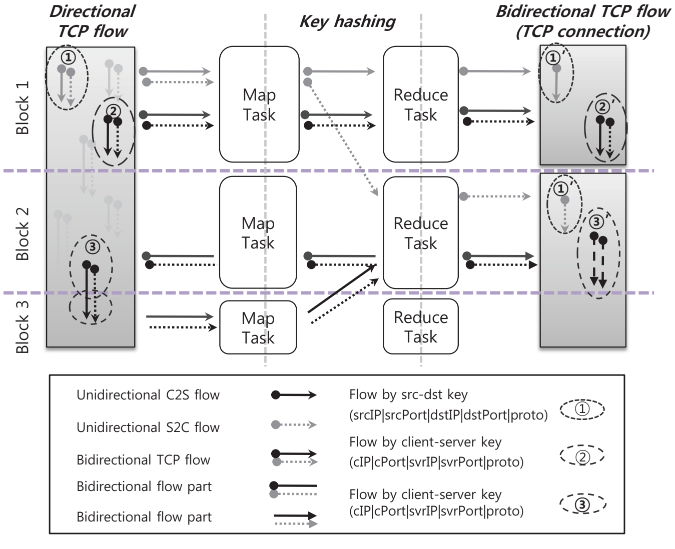


Figure 4: Two directional TCP flows, that belong to a single TCP connection: two flows of TCP connection ① are mapped to different reducers; two flows of TCP connection ② and ③ are mapped to a single reducer due to C2S and S2C flow classification.

This problem has been solved using a simple rule for classifying two directional IP flows of a single TCP connection into C2S or S2C flows with the port number comparison. If a packet contains a source port number smaller than the destination port number, it is classified into a S2C flow. Otherwise, packets with the larger source port number will be categorized into a C2S flow. After classifying IP flows, a map task can emit two directional TCP flow parts (②) to the same reduce task. In addition, multiple map tasks on different HDFS blocks can emit their own parts of a TCP connection to the same reduce task by using the client-server flow classification function with the port number comparison. For example, in Fig. 4, C2S and S2C TCP flows (③) on HDFS block 2 and 3 will be recognized for the same TCP connection with the port number comparison. Then, two map tasks processing TCP flow parts on their HDFS blocks emit the packets of both C2S and S2C TCP flows to the same reduce task (③').

3.4.2 Optimizing TCP metric calculation at the reduce task

After pulling packet streams of a TCP connection from multiple map tasks, a reduce task computes the performance metrics for a TCP connection. For example, when calculating the RTT values of a TCP connection, a reduce task has to find a pair of a TCP segment and its corresponding ACK from a series of unsorted packet records. In this case, a lot of packet records may be kept at the heap space of a reduce task, which often results in the failure of the MapReduce job. Particularly, a reduce task processing a long fat TCP flow cannot maintain many packet records at the heap memory to traverse unsorted TCP segments and their corresponding ACKs.

To solve this problem, we have simplified the computation process at the reduce task by sending a TCP segment and its corresponding ACK together from the map task. That is, a map task extracts a TCP segment from a packet and couples it with the corresponding ACK with the key of TCP sequence and ACK numbers. This coupling process is also applied to ACK segments. Since both a TCP segment and its corresponding ACK use the same key, they will be grouped during the shuffle and sort phase of MapReduce. Then, a reduce task can sequentially compute RTT, retransmission rate, and out-of-order from a coupled pair of a TCP segment and its corresponding ACK, which are sorted with the TCP sequence number. In the reduce task, we can add up various TCP metrics according to the analysis purposes. In this paper, we considered four representative metrics of throughput, RTT, retransmission rate, and out-of-order for each TCP connection, as shown in Fig. 3.

3.5 Application-layer analysis in MapReduce

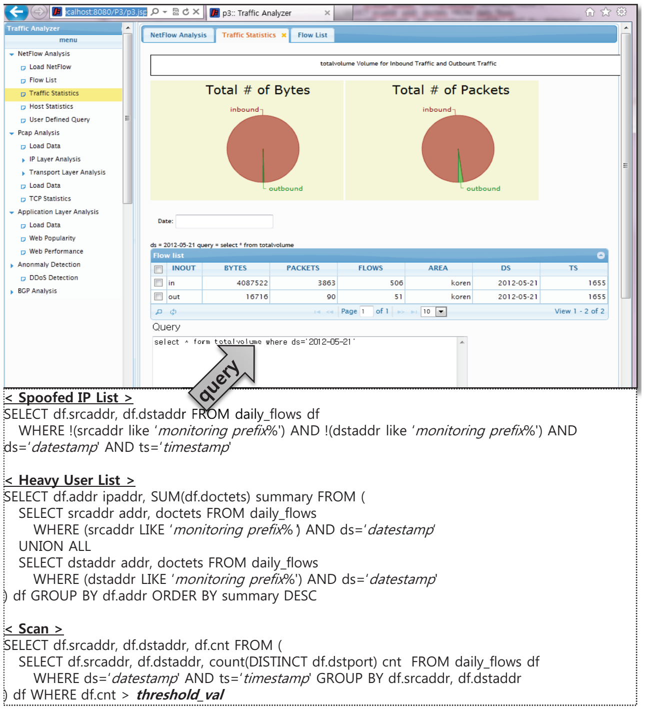
`libpcap` input format in HDFS makes it possible to build up application-specific analysis MapReduce modules for web, multimedia, file sharing, and anomalies. In this work, we focus on the HTTP-based application analysis in MapReduce, because HTTP is popular in many Internet applications.

3.5.1 Web traffic analysis

For web traffic analysis, we have developed a MapReduce algorithm that can investigate website popularity and user behavior by examining HTTP packets. We look into the header information of the first N (e.g., 3) HTTP packets because we are interested only in the header fields, not the user content. In the MapReduce algorithm, first, the map task extracts several fields out from the header of a HTTP request message, such as a Uniform Resource Identifier (URI), content-related fields, user agent, and host values. Then, the reduce task summarizes the statistics or popularity per website (or domain/host), webpage (or URL), content URI, or user. A website is considered as a unique host name field at the HTTP message. Each web page, specified as a URL, consists of several content URIs that share the same referrer field. For web traffic analysis, we have implemented two MapReduce jobs: the first job sorts the URL list accessed by each user; the second job summarizes the user and view counts per host and URL. HTTP analysis can be easily extended to other Internet applications, such as SIP for video or VoIP, file-sharing, and anomalies.

3.5.2 DDoS traffic analysis

In order to show the effectiveness of MapReduce-based application-layer traffic analysis, we present a HTTP-based distributed denial-of-service (DDoS) attack detection method implemented in MapReduce. We employ a counter-based DDoS detection algorithm in MapReduce [23]. The map task generates keys to classify the requests and response HTTP messages. Then, the reduce task summarizes the HTTP request messages and marks the abnormal traffic load by comparing it with the threshold. Though the DDoS traffic analysis MapReduce algorithm is used for offline forensic analysis, it can be extended to real-time analysis.



ds = 2012-05-21 query = select * from totalvolume

Flow list

	INOUT	BYTES	PACKETS	FLOWS	AREA	DS	TS
□	in	4087522	3863	506	koren	2012-05-21	1655
□	out	16716	90	51	koren	2012-05-21	1655

Page 1 of 1 10 View 1 - 2 of 2

Query

```
select * form totalvolume where ds='2012-05-21'
```

< Spoofed IP List >

```
SELECT df.srcaddr, df.dstaddr FROM daily_flows df
   WHERE !(srcaddr like 'monitoring prefix%') AND !(dstaddr like 'monitoring prefix%') AND
ds='datestamp' AND ts='timestamp'
```

< Heavy User List >

```
SELECT df.addr ipaddr, SUM(df.doctets) summary FROM (
   SELECT srcaddr addr, doctets FROM daily_flows
      WHERE (srcaddr LIKE 'monitoring prefix%') AND ds='datestamp'
   UNION ALL
   SELECT dstaddr addr, doctets FROM daily_flows
      WHERE (dstaddr LIKE 'monitoring prefix%') AND ds='datestamp'
) df GROUP BY df.addr ORDER BY summary DESC
```

< Scan >

```
SELECT df.srcaddr, df.dstaddr, df.cnt FROM (
   SELECT df.srcaddr, df.dstaddr, count(DISTINCT df.dstport) cnt  FROM daily_flows df
      WHERE ds='datestamp' AND ts='timestamp' GROUP BY df.srcaddr, df.dstaddr
) df WHERE df.cnt > threshold_val
```

Figure 5: Web user interface supporting Hive queries on NetFlow data.

3.6 Interactive query interface with Hive

Since the MapReduce framework is useful for processing a large amount of IP packets and NetFlow data in parallel, researchers capable of developing the measurement and analysis function can invent their own MapReduce modules for fast response time. However, it may take long time for analysts to write application-specific analysis programs in MapReduce, though analysis modules implemented in MapReduce can bring a better throughput. Therefore, a simple query interface is more convenient than MapReduce programming to users interested in agile traffic analysis. Moreover, it is more expressive and extensive for users to ask versatile questions on the traffic data through the query interface. Hive provides the ability to generate MapReduce codes through the SQL-like query interface, Hive Query Language (HiveQL). Therefore, we harness the Hive query interface for easy operational Internet traffic analysis.

We implemented a NetFlow monitoring system with our MapReduce flow analysis algorithms and Hive, as shown in Fig. 5. The traffic collector receives the NetFlow streams from a router and writes them to a single HDFS file every five minutes. After a NetFlow file is closed every five minutes, the job scheduler invokes IP analysis MapReduce jobs to process NetFlow data and uploads flow records and IP statistics to Hive tables. Then, operators can issue user-defined queries to the Hive table over the characteristics of IP flow data. For example, users can monitor anomalous flows related with spoofed IP addresses, scanning, or heavy users with the HiveQL, as shown in Fig. 5. Since the queries will be translated into MapReduce codes, their performance depends on query optimization and the Hive platform.

4. EXPERIMENTS

4.1 Hadoop testbed

For the experiments, we used two local Hadoop testbeds[2]. As shown in Table 1, the first testbed is a high-performance 30-node cluster in a single rack where 30 nodes are connected in 1 GE, and the second is a large-scale 200-node cluster where nodes are connected in 1 GE and six racks are interconnected with 4 Gbps links. The 30-node cluster is configured with the Hadoop 1.0.3 version with an HDFS block size of 128 MB and a replication of two, and the 200-node cluster with Hadoop 0.20.203 version with a block size of 64 MB and a replication of two.

Table 1: Hadoop testbed

Nodes	CPU	RAM	HDD
30	2.93 GHz (8 core)	16GB	4TB
200	2.66 GHz (2 core)	2GB	0.5TB

4.2 Accuracy

Before performing the scalability experiments, we investigated the accuracy of the packet search algorithm to find a packet record from a HDFS block. While we can achieve the ability of accessing packet records fast in parallel, we may experience packet losses because timestamp values in `libpcap` used by the pattern matching algorithm might appear at other fields. From 10 GB to 1 TB input data, we examined the accuracy using CoralReef, an IP-layer analysis tool, by comparing our IP analysis command and observed 100% of packet processing.

4.3 Scalability

For the scalability test, we probed the performance of IP, TCP, and HTTP-layer analysis MapReduce jobs for 1 TB `libpcap` files on the 30-node Hadoop testbed as the number of Hadoop worker nodes varies. Figure 6 shows how the job completion time and the throughput are enhanced when we add five nodes to the Hadoop cluster. From five to 30 Hadoop worker nodes, it is apparently observed that resource-proportional performance improvement is possible. For example, the job completion time of IP analysis (`PcapTotalStats`) is decreased from 71 minutes with five nodes to 11 minutes with 30 nodes. The throughput of IP analysis was 1.9 Gbps on five nodes and 12.8 Gbps on 30 nodes (6.7× improvement). The most complicated TCP analysis job was finished within 121 minutes on 5 nodes and 15 minutes on 30 nodes The throughput of TCP analysis was enhanced from 1.1 Gbps on five nodes to 8.8 Gbps on 30 nodes (8× increase). As TCP analysis MapReduce job is complicated, the performance of TCP analysis is worse than that of IP and HTTP analysis. TCP analysis map and reduce tasks require more CPU, storage, and network resources to reassemble TCP connections and to compute metrics per TCP connection.

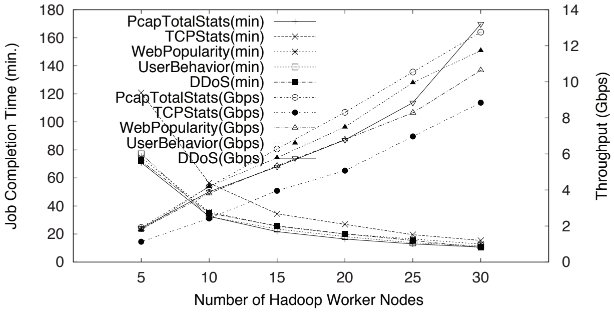


Figure 6: Completion time and throughput of traffic analysis MapReduce jobs on the 30-node Hadoop testbed for 1 TB `libpcap` files.

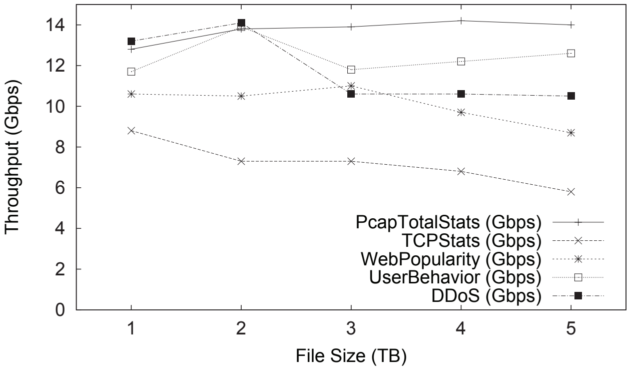


Figure 7: Size-up of traffic analysis MapReduce jobs on the 30-node Hadoop testbed for 1 - 5 TB `libpcap` files.

In order to examine the influence of the varying input file size, we executed IP, TCP, and HTTP analysis jobs under 1 to 5 TB packet trace files on the 30-node Hadoop testbed. Figure 7 shows the performance results of analysis jobs as the input file increases from 1 to 5 TB. The IP analysis job (`PcapTotalStats`) achieves the improved throughput of 14.0 Gbps for 5 TB. However, TCP/HTTP (`WebPopularity`) analysis jobs show a slightly decreased throughput of 5.8/8.7 for 5 TB from 8.8/10.6 for 1 TB because they cannot use the combiner to aggregate the intermediate data, which results in the overhead at reduce tasks. When we used the 200-node testbed for the size-up experiment, it is seen in Fig. 8 that our tool maintains the high throughput with the rising pattern for the increased file size. In particular, IP/TCP/HTTP(`WebPopularity`) analysis jobs accom-

[2] Though Amazon Web Service (AWS) of EC2 is a good candidate for a large experimental cluster, it is not easy to move multi-tera bytes of data to the Amazon datacenter ("Amazon Import service" is usually used for transferring a large amount of data on the physical HDD by FedEx.). Running many high-performance AWS EC2 "Extra Large" machines is not cheap. In addition, we are not aware of the physical environment of the AWS EC2 virtual machines. Therefore, we have used local Hadoop testbeds.

plish 14.6/8.5/9.7 Gbps for 5 TB. In the 200-node testbed, the increased number of total reduce tasks has mitigated the overhead of processing the non-aggregated intermediate data from map tasks.

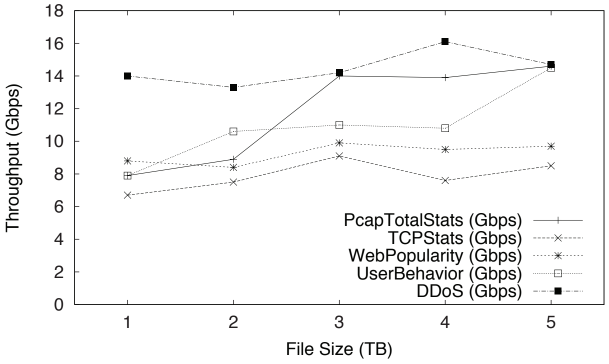


Figure 8: Size-up of traffic analysis MapReduce jobs on the 200-node Hadoop testbed for 1 - 5 TB `libpcap` files.

4.4 Comparison with CoralReef and RIPE's Pcap

We have compared our tool with the well-known CoralReef tool on a single server to explain how the parallel computation on a cluster improves job performance in proportion to cluster size. As shown in Fig. 9, the IP-layer analysis job (`PcapTotalStats`) on the 30-node cluster achieves up to 20.3× increased throughput for 1 TB. In CoralReef, the limited number of input files also hinders the aggregation job of processing many packet trace files.

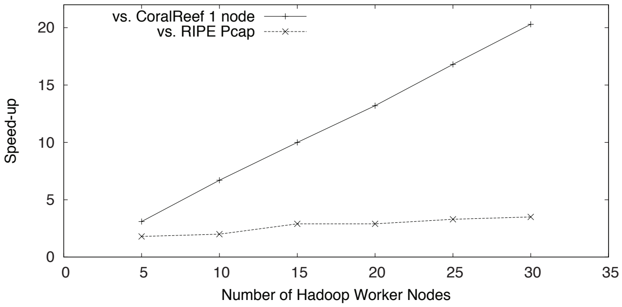


Figure 9: Speed-up of traffic analysis MapReduce jobs vs. CoralReef and vs. RIPE pcap on the 30-node Hadoop testbed for 1 TB `libpcap` files.

We then evaluated how our packet-search algorithm for reading `libpcap` files on HDFS enhances the performance of MapReduce jobs. Our `PcapInputFormat` supports the parallel processing capability per HDFS block, whereas RIPE's packet library [21] employs the file-based parallel processing approach. As shown in Fig. 9, our `PcapInputFormat` outperforms RIPE's one by 3.5× for 1 TB on 30 nodes (1.8× on five nodes). The file-based processing method cannot use the advantage of data locality in Hadoop, because DataNodes often have to receive remote HDFS blocks. In the case of task failures, the overall performance of a file-based MapReduce job will be deteriorated. When map/reduce tasks failed during the experiment, our tool shows 3.9/4.3× improved performance on 30 nodes. In order to emulate our block-based MapReduce job, the input file should be chunked by the HDFS block size (e.g., 64 MB) in advance.

4.5 Breakdown of performance bottlenecks

When running traffic analysis MapReduce jobs in Hadoop, we are confronted with several performance bottlenecks of Hadoop components: one issue is related with hardware such as CPU, memory, hard disk, and network, and the other with MapReduce algorithm optimization.

In Hadoop, map and reduce tasks share CPU resources together. When the maximum number of tasks are assigned and run, CPU-intensive jobs may experience performance saturation. In our case, the TCP-analysis MapReduce job is extremely CPU-bound so that its performance improvement is eminent as the number of CPU cores is increased thanks to the scale-out feature of Hadoop. We might meet the memory issue of the master node when many files are loaded to HDFS because the master node of HDFS should retain the location information regarding a file and its related HDFS blocks at the memory[3]. In a large-scale Hadoop testbed, the network among worker nodes is also a critical bottleneck. The inter-rack link may experience congestion if its bandwidth is not provisioned to satisfy the inter-rack traffic load.

Another performance issue is related with the hard disk. Map tasks of the traffic analysis job write the intermediate results to the local disk, and reduce tasks pull the data by RPC. As the volume of the input traffic file increases, the hard disk capacity should be large enough to accommodate the temporary data from map tasks. By default, the HDFS replicated each block three times for backup, which requires more storage. The TCP analysis MapReduce job requires additional capacity for the intermediate data because its map task has to emit each TCP segment information per connection to the reduce task without using the combiner to reduce the intermediate data. Though high-speed access to large data with an expensive solid-state disk (SSD) is possible, this should be determined by considering the cost-performance tradeoff or return-on-investment (ROI) on a large-scale cluster. A popular solution to improve the hard disk access speed is to use multiple hard disks in RAID0 (data striping mode), which normally increases I/O performance in proportion to the number of hard disks.

The software issue is related to the MapReduce algorithm optimization. In MapReduce algorithms, the combiner is useful for diminishing the traffic between map and reduce tasks as well as the computation overhead at the reduce task. Applying the combiner helps the reduce task relieved of frequent disk I/O's. In IP analysis MapReduce algorithms, the value lists for the same key of five tuples can be merged by the combiner. In TCP analysis, however, the reduce task has to search every TCP segment and its corresponding ACK for the performance metrics such as RTT, retransmissions, and out-of-order. This prevents the MapReduce job from using the combiner at the map task, so that a TCP analysis job causes a large overhead in memory, network, and disk. That is, a TCP analysis reduce task requires a large amount of

[3] In [6], as a rule of thumb, it is reported that the metadata for a file in HDFS takes 150 bytes and 1 million files will take 300 MB of memory on the name node.

memory to contain the TCP records, and CPU resources to search the TCP segments and their corresponding ACKs. Thus, we have optimized the TCP analysis MapReduce job by emitting a group of a TCP segment and its corresponding ACK information to the reduce task, which decreases the running time of the TCP analysis MapReduce algorithm by 3.2 times without errors of memory shortage.

There are many tuning parameters in Hadoop for performance improvement, such as the number of map or reduce tasks and buffer size. Since map and reduce tasks share CPU resources, the maximum number of tasks in execution should be deliberately configured. Though the maximum number of map tasks are usually two times more than that of reduce tasks, increasing the number of reduce tasks often improves the performance of reduce-side complex computation. The I/O buffer to sequence files is also an important factor in delivering the intermediate data between two MapReduce jobs, like in the website popularity analysis. In HTTP analysis, we enhanced the job completion time by 40% by setting 4 KB of the default I/O buffer size to 1 MB.

5. CONCLUSION

In this paper, we presented a scalable Internet traffic measurement and analysis scheme with Hadoop that can process multi-terabytes of `libpcap` files. Based on the distributed computing platform, Hadoop, we have devised IP, TCP, and HTTP traffic analysis MapReduce algorithms with a new input format capable of manipulating `libpcap` files in parallel. Moreover, for the agile operation of large data, we have added the web-based query interface to our tool with Hive. From experiments on large-scale Hadoop testbeds with 30 and 200 nodes, we have demonstrated that the MapReduce-based traffic analysis method achieves up to 14 Gbps of throughput for 5 TB input files. We believe our approach to the distributed traffic measurement and analysis with Hadoop can provide the scale-out feature for handling the skyrocketing traffic data. In future work, we plan to support real-time traffic monitoring in high-speed networks.

Acknowledgment

We thank the anonymous reviewers for their helpful feedback and Sue Moon for her kind guidance in developing the final draft. This research was supported partially by the KCC under the R&D program supervised by the KCA(KCA-2012-(08-911-05-002)) and partially by the Basic Science Research Program through the NRF, funded by the Ministry of Education, Science and Technology (NRF 2010-0021087). The 200-node Hadoop experiment was supported by Super computing Center/KISTI. The corresponding author of this paper is Youngseok Lee.

6. REFERENCES

[1] Cisco White Paper, *Cicso Visual Networking Index: Forecast and Methodology, 2011-2016*, May 2012.

[2] K. Cho, K. Fukuda, H. Esaki, and A. Kato, *Observing slow crustal movement in residential user traffic*, ACM CoNEXT2008.

[3] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Cluster*, USENIX OSDI, 2004.

[4] S. Ghemawat, H. Gobioff, and S. Leung, *The Google file system*, ACM SOSP, 2003.

[5] Hadoop, http://hadoop.apache.org/.

[6] T. White, *Hadoop: the Definitive Guide*, O'Reilly, 3rd ed., 2012

[7] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, *Hive: a warehousing solution over a map-reduce framework*, VLDB, August 2009.

[8] Tcpdump, http://www.tcpdump.org.

[9] Wireshark, http://www.wireshark.org.

[10] CAIDA CoralReef Software Suite, http://www.caida.org/tools/measurement/coralreef.

[11] M. Roesch, *Snort - Lightweight Intrusion Detection for Networks*, USENIX LISA, 1999.

[12] Bro, http://www.bro-ids.org.

[13] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, *The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware*, International Conference on Recent Advances in Intrusion Detection (RAID), 2007.

[14] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, and D. Rossi, *Live traffic monitoring with tstat: Capabilities and experiences*, 8th International Conference on Wired/Wireless Internet Communication, 2010.

[15] Cisco NetFlow, http://www.cisco.com/web/go/netflow.

[16] M. Fullmer and S. Romig, *The OSU Flow-tools Package and Cisco NetFlow Logs*, USENIX LISA, 2000.

[17] D. Plonka, *FlowScan: a Network Traffic Flow Reporting and Visualizing Tool*, USENIX Conference on System Administration, 2000.

[18] QoSient, LLC, *argus: newtork audit record generation and utilization system*, http://www.qosient.com/argus/.

[19] Arbor Networks, http://www.arbornetworks.com.

[20] Y. Lee, W. Kang, and Y. Lee, *A Hadoop-based Packet Trace Processing Tool*, International Workshop on Traffic Monitoring and Analysis (TMA 2011), April 2011.

[21] RIPE Hadoop PCAP, https://labs.ripe.net/Members/wnagele/large-scale-pcap-data-analysis-using-apache-hadoop, Nov. 2011.

[22] F. Fusco and L. Deri, *High Speed Network Traffic Analysis with Commodity Multi-core Systems*, ACM IMC 2010, Nov. 2010.

[23] Y. Lee and Y. Lee, *Detecting DDoS Attacks with Hadoop*, ACM CoNEXT Student Workshop, Dec. 2011.

[24] CNU Project on traffic analysis in Hadoop, https://sites.google.com/a/networks.cnu.ac.kr/dnlab/research/hadoop.