Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework

Aryan TaheriMonfared*, Tomasz Wiktor Wlodarczyk*, Chunming Rong*,
*Department of Electrical Engineering and Computer Science, University of Stavanger, Norway
{aryan.taherimonfared, tomasz.w.wlodarczyk, chunming.rong}@uis.no

***Abstract*—The proper operation and maintenance of a network requires a reliable and efficient monitoring mechanism. The mechanism should handle large amount of monitoring data which are generated by different protocols. In addition, the requirements (e.g. response time, accuracy) imposed by long-term planned queries and short-term ad-hoc queries should be satisfied for multi-tenant computing models.**

This paper proposes a novel mechanism for scalable storage and real-time processing of monitoring data. This mechanism takes advantage of a data-intensive framework for collecting network flow information records, as well as data points' indexes. The design is not limited to a particular monitoring protocol, since it employs a generic structure for data handling. Thus, it's applicable to a wide variety of monitoring solutions.

I. Introduction

Monitoring and measurement of the network is a crucial part of infrastructure operation and maintenance. A good understanding of the traffic passing through the network is required for both planned and ad-hoc tasks. Capacity planning and traffic matrix processing are planned, whereas traffic engineering, load-balancing, and intrusion detection are ad-hoc tasks which often require real-time behaviour.

1) Storage Requirements: Quite often, ad-hoc tools are used for analysing network properties [1]. Traffic dumps or flow information are common data type for an ad-hoc analysis. The data volume for these types can be extremely large.

2) Analytic Requirements: The storage should be distributed, reliable, and efficient to handle high data input rate and volume. Processing this large data set for an ad-hoc query should be near real-time. It should be possible to divide and distribute the query over the cluster storing the data.

3) Privacy Policies: Storing the packet payload which corresponds to the user data is restricted according to European data laws and regulations [2]. The same policy applies to the flow information as well.

A. Related Work

Li et al. [3] surveyed the state of the art in flow information applications. They identified several challenges in the fields such as: machine learning's feature selection for an effective analysis, real-time processing, and efficient storage of data sets. Lee et al. [4] proposed a mechanism for importing network dumps (i.e. libpcap files) and flow information to HDFS. They've implemented a set of statistical tools in MapReduce for processing libpcap files in HDFS. The tool set calculates statistical properties of IP, TCP, and HTTP protocols. Their solution copies recently collected NetFlow data to Hive tables in fixed intervals which doubles the storage capacity requirement. Andersen et al. [5] described the management of network monitoring datasets as a challenging task. They emphasized on the demand for a data management framework with the eventual consistency property and the real-time processing capability. The framework should facilitate search and discovery by means of an effective query definition and execution process. Balakrishnan et al. [6] and Cranor et al. [1] proposed solutions for the real-time analysis of network data streams. However, they may not be efficient for the analysis of high-speed streams in a long period [5].

B. Contributions

A flexible and efficient mechanism is designed and implemented for real-time storage and analysis of network flow information. In contrast to other solutions, which have analysed binary files on distributed storage systems, a NoSQL type of data store provides real-time access to a flexible data model. The data model flexibility makes it compatible with different monitoring protocols. Moreover, the structure leads to fast scanning of a small part of a large dataset. This property provides low latency responses which facilitate exploratory and ad-hoc queries for researchers and administrators. The solution provides a processing mechanism which is about 4000 times faster than the traditional one.

The study concentrates on flow information records, due to regulatory and practical limitations such as privacy directives and payload encryption. However, one can leverage the same solution for handling wider and richer datasets which contain application layer fields. This study is a part of our tenant-aware network monitoring solution for the cloud model.

The rest of the paper is organized as follows: Section II explains the background information about data-intensive processing frameworks and network monitoring approaches. Section III describes the Norwegian NREN backbone network as a case study. Dataset characteristics and monitoring requirements of a production network are explained in this section. Section IV introduces our approach toward solving data processing challenges for network monitoring. Section V discusses technical details of the implementation as well as performance tunings for improving the efficiency. Section VI evaluates the solution by performing common queries and Section VII concludes the paper and introduces future works.

II. Background

A. Framework for Data-Intensive Distributed Applications

Using commodity hardware for storing and processing large sets of data is becoming very common [7]. There are multiple proprietary, open-source frameworks and commercial services providing similar functionality such as: Apache's Hadoop[1] [8] and related projects, Google's File System (GFS) [9], BigTable [10], Microsoft's Scope [11], Dryad [12]. In the following, required components for the analysis and storage of our dataset is explained.

1) File System (Hadoop Distributed FS): The first building block of our solution, for handling network monitoring data, is a proper file system. The chosen file system must be reliable, distributed and efficient for large data sets. Several file systems can fulfil these requirements, such as Hadoop Distributed File System (HDFS) [8], MooseFS[2], GlusterFS[3], Lustre[13], Parallel Virtual File System (PVFS)[14]. Despite the variety, most of these file systems are missing an integrated processing framework, except HDFS. This capability in HDFS makes it a good choice as the underlying storage solution.

2) Data Store (HBase): Network monitoring data, and packet header information are semi-structured data. In a short period after their generation, they're accessed frequently, and a variety of information may be extracted from them. Apache HBase[4][15] is the most suitable non-relational data store for this specific use-case. HBase is an open-source implementation of a column-oriented distributed data source inspired by Google's BigTable [10], which can leverage the MapReduce processing framework of Apache. Data access in HBase is key-based. It means a specific key or a part of it can be used to retrieve a cell (i.e. a record), or a range of cells [15]. As a database system, HBase guarantees consistency and partition tolerance from the CAP theorem [16] (aka. Brewer's theorem).

3) Processing Framework (Hadoop MapReduce): Processing large data sets has demanding requirements. The processing framework should be able to partition the data across a large number of machines, and exposes computational facilities for these partitions. The framework should provide the abstraction for parallel processing of data partitions and tolerate machine failures. MapReduce [17] is a programming model with these specifications. Hadoop is an open source implementation by Apache Software Foundation, which will be used in our study.

B. Network Monitoring

This study focuses on the monitoring of backbone networks. The observation can be instrumented using Simple Network Management Protocol (SNMP) metrics, flow information (i.e. packet header), and packet payload. SNMP does not deliver the granularity demanded by our use-case; also storing packets payloads from a high capacity network is not feasible, because of both scalability issues [1] and privacy policies [2]. Thus, we are more interested in the packet header, and IP flow information. An IP flow is a set of packets passing through a network between two endpoints, and matching a certain set of criteria, such as one or more identical header fields [18]. In our study, a flow is a canonical five-tuple: source IP, source port, destination IP, destination port, and protocol. Flow information is flushed out of the network device after 15 seconds of inactivity, 30 minutes of persistent activity, TCP session termination, or when the flow buffer in the device is full. This makes the start and end time of a flow imprecise [19]. IP flow information is an efficient data source for the real-time analysis of network traffic.

IP flow information can be exported using different protocols, in different formats. NetFlow [20], sFlow [21], and IP Flow Information Export (IPFIX) [18] are designed to handle network monitoring data. Collected data have a variety of use-cases. They can be used for security purposes, audit, accountability, billing, traffic engineering, capacity planning, etc.

C. Testing Environment

We have implemented, optimized, and tested our suggested solution. The testing environment consists of 19 nodes, which deliver Hadoop, HDFS, HBase, ZooKeeper, Hive services. The configuration for these nodes is as follows: 6x core AMD Opteron(tm) Processor 4180, 4x 8GB DDR3 RAM, 2x 3 TB disks, 2x Gigabit NIC.

III. Case Study: Norwegian National Research and Education Network (NREN)

This study focuses on the storage and processing of IP flow information data for the Norwegian NREN backbone network. Two core routers, TRD_GW1 (in Trondheim) and OSLO_GW (in Oslo), are configured to export flow information. Flow information are collected using NetFlow [20] and sFlow [21].

A. Data Volume

Flow information is exported from networking devices at different intervals or events (e.g. 15 seconds of inactivity, 30 minutes of activity, TCP termination flag, cache exhaustion). The data are collected in observation points, and then the anonymized data are stored for experiments. Crypto-PAn [22] is used for the data anonymization. The mapping between the original and anonymized IP address is "one-to-one", "consistent across traces", and "preserves prefix".

Flow information is generated by processing a sampled set of packets. Although sampled data is not as accurate as not-sampled one, studies showed they can be used efficiently for network operation and anomaly detection, by means of right methods [23], [24].

There need to be a basic understanding of the dataset for designing the proper data store. Data characteristics, common queries and their acceptable response times are influential factors in the schema design. The identifier for accessing the data can be on one or more fields from the flow information record

[1] http://hadoop.apache.org/
[2] http://www.moosefs.org/
[3] http://www.gluster.org/
[4] http://hbase.apache.org/

TABLE I: Traffic Characteristics

Traffic Type	Statistics/day		
	Avg	Max	Min
Distinct Source IPs	987104	4740760	122266
Distinct Source IPs and Source ports	6083640	13188647	844898
Distinct Destination IPs	1613040	2488893	420686
Distinct Destination IPs and Destination ports	7010330	16379274	1113095
Distinct Bidirectional flows	10683200	21454096	1829854
NetFlow records	21962800	44036078	4373665

(e.g. source or destination IP addresses, ports, Autonomous Systems (AS), MACs, VLANs, interfaces, etc.). Figure 1 depicts number of unique *source*, *destination IP addresses*, unique *source IP:source port*, *destination IP:destination port* tuples, unique *bidirectional flows (biflows)*, and *flow information records* per day for the TRD_GW1 in a 5 month period. The summary of numeric values for TRD_GW1 and OSLO_GW is presented in Table I.

The average number of flow information records for both routers is 22 millions per day, which corresponds to 60 GBs of data in binary form. However, this number can become much bigger if flow informations are collected from more sources and the sampling rate is increased.

B. Data Access Methods

Monitoring data can be accessed for different purposes such as: billing information, traffic engineering, security monitoring, forensics. These purposes corresponds to a big set of possible queries. The schema can be design such that it performs very well for one group of queries. That may lead to a longer execution time for the other query groups. Our main goal is reaching the shortest execution time for security monitoring and forensics queries. Three types of queries are studied, *IP based*: requires fast IP address lookups (e.g. specific IPs, or subnets), *Port based*: requires fast Port address lookups (e.g. specific services), and *Time based*: requires fast lookup on a time period.

Network monitoring data, and packet header information are semi-structured data. They have arbitrary lengths and a various number of fields. Storing this type of data as binary files in a distributed file system is challenging. The next section discusses several storage schemas and their applicability to desired access methods.

IV. Solution

Two major stages in the life cycle of the monitoring data can be considered: short-term processing, and long-term archiving.

- Short-term processing: when collected monitoring data are imported into the data store, several jobs should be executed in real-time. These jobs generate real-time network statistics, check for anomaly patterns and routing issues, aggregate data based on desired criteria, and etc.
- Long-term archiving: Archived data can be accesses for security forensics, or on-demand statistical analysis.

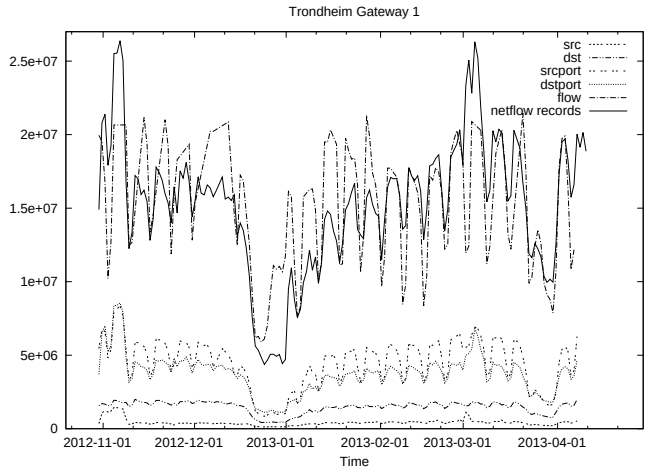


Fig. 1: Number of distinct source IPs, source IPs: source ports, destination IPs, destination IPs:destination ports, bidirectional flows, and raw netflow records collected from Trondheim gateway 1.

A. Choice of Technologies

Apache HBase satisfies our requirements (Section I) such as consistency and partition tolerance. Moreover, the data staging is affordable by proper configuration of cache feature, in-memory storage size, in-filesystem storage size, regions configuration and pre-splitting for each stage, and etc. For instance, short-term data can be stored in regions with large memory storage and enabled block cache. The block cache should be configured such that the Working Set Size (WSS) fits in memory [25]. While long-term archives are more suitable for storage in the filesystem.

Hive[5] is an alternative for HBase, which is not suitable for our application. It doesn't support binary key-values, and all parameters are stored as strings. This approach demands for more storage, and makes the implementation inefficient. While composite key structure is an important factor for fast data access in the design, it is not supported by Hive. Although Hive provides an enhanced query mechanisms for retrieving data, aforementioned issues make it inapplicable to our purpose.

B. Design Criteria

A table schema in HBase has three major components: rowkey, column-families, and columns structures.

1) Row Key: A rowkey is used for accessing a specific part of data or a sequence of them. It is a byte array which can have a complex structure such as a combination of several objects. Rowkey structure is one of the most important part of our study because it has a great impact on the data access time, and storage volume demand. The followings are our criteria for designing the rowkey:

- **Rowkey Size**: Rowkey is one of the fields stored in each cell, and is a part of a cell coordinate. Thus, it should

[5]http://hive.apache.org/

be as small as possible, while efficient enough for data access.

- **Rowkey Length (Variable versus Fixed)**: Fixed length rowkeys, and fields help us to leverage the lexicographically sorted rows in a deterministic way.
- **Rowkey Fields' Order (with respect to region load)**: Records are distributed over regions based on regions' key boundaries. Regions with high loads can be avoided by a uniform distribution of rowkeys. Thus, the position of each field in the rowkey structure is important. Statistical properties of a field's value domain are determining factors for the field position.
- **Rowkey Fields' Order (with respect to query time)**: Lexicographic order of rowkeys makes queries on the leading field of a rowkey much faster than the rest. This is the motivation for designing multiple tables with different fields order. Therefore, each table provides fast scanning functionality for a specific parameter.
- **Rowkey Fields' Type**: Fields of a rowkey are converted to byte arrays then concatenated to create the rowkey. Fields' types have significant effect on the byte array size. As an example number 32000 can be represented as a *short* data type or as a *string*. However, the *string* data type require two times more number of bytes.
- **Rowkey Timestamps vs. Cell Version**: It's not recommended to set the maximum number of permitted versions too high [25]. Thus, there should be a timestamp for the monitoring record as a part of the rowkey.
- **Timestamps vs. Reverse Timestamps**: In the first stage of data life cycle, recent records are frequently accessed. Therefore, revere timestamps are used in the rowkey.

2) Column Families: Column families are the fixed part of a table which must be defined while creating the schema. It's recommended to keep the number of families less than three, and those in the same table should have similar access patterns and size characteristics (e.g. number of rows) [15]. Column family's name must be of string type, with a short length. The family's name is also stored in the cell, as a part of the cell coordination. A table must have at least one column family, but it can have a dummy column with an empty byte array. We have used constant value D for our single column family across all tables.

3) Columns: Columns are the dynamic part of a table structure. Each row can have its own set of columns which may not be identical to other rows' columns. Monitoring data can be generated by different protocols, and they may not have similar formats/fields. Columns make the solution flexible and applicable to a variety of monitoring protocols.

There are several tables with different fields' orders in rowkeys, but not all of them have columns. Complete monitoring record is just inserted into the reference table, and others are used for fast queries on different rowkey fields.

C. Schemas

Section III-B explained desired query types and Section IV-B1 described required properties of a rowkey for fast scanning. Here, three table types are introduced, each addressing one query category: IP-based, Port-based, and Time-based tables.

1) IP Based Tables:

a) T1 (reference table), T2: The rowkey of this table consists of: *source IP address, source port, destination IP address, destination port,* and *reverse timestamp* (Table II). Columns in this family are flexible and any given set can be stored there. Column qualifiers identifier are derived from fields' names, and their values are corresponding values from flow information records. Other tables are designed as secondary indexes. They improve access time considerably for the corresponding query group. Table T1 is used for retrieving flow information parameters that are not in the rowkey (e.g. number of sent or received packets, bytes, flows)

Table T2 has destination address and port in the lead position. This is used in combination with T1 for the analysis of bidirectional flows.

b) T3, T4: are suitable when source and destination addresses are provided by the query (Table II). For instance, when two ends of a communication are known, and we want to analyse other parameters such as: communication ports, traffic volume, duration, etc.

2) Port Based Tables:

a) T5, T6: are appropriate tables for service discovery (Table II). As an example, when we want to discover all nodes delivering SSH service (on default port: 22), we can specify the lead fields on T5 and T6 (source and destination ports), and let the data store returns all service providers and their clients. If the client *c1* is communicating on the port *p1* with the server *s1* on the port 22 at time *ts*, then there is a record with the rowkey: *[22][s1][c1][p1][1-ts]* in the data store.

b) T7, T8: can fulfil the requirement for identifying clients who use a particular service (Table II). The same record from T5, T6 will have the rowkey: *[22][c1][s1][p1][1-ts].*

3) Time Based Tables: OpenTSDB[6] is used for storing time series data. This can be an efficient approach for accessing and processing flows of a specific time period. A rowkey in OpenTSDB consists of: a metric, a base timestamp, and a limited number of tags in the key-value format. Source and destination IP addresses and ports are represented as tags, and a set of metrics are defined. Five fields from the flow information record are chosen as metrics: number of input and output bytes, input and output packets, and flows.

D. Storage Requirement

The storage volume required for storing a single replication of a not-compressed record can be estimated using Equation (1) as depicted in Table III. However, this estimation may vary considerably if protocols other than NetFlow v5 and sFlow are used for collecting monitoring data (e.g. IPFIX raw record can be 250 bytes, containing 127-300 fields.)

Equation (2) is used for calculating the required capacity for tables T2-T8 (See Table III). These tables don't have columns and values, which makes them much smaller than Table 1.

[6]www.opentsdb.net

TABLE III: Storage requirements IPv4

	Est. # records	Storage for T1	Storage for T2-T8	Storage for OpenTSDB	Total
Single Record	1	$(37 * 23) + (133) \sim 1KB$	$7tables * 23B = 161B$	$5metrics * 2B = 10B$	$\sim 1KB$
Daily Import	$\sim 20million$	$1KB * 20 * 10^6 = 20GB$	$161B * 20 * 10^6 \sim 3GB$	$10B * 20 * 10^6 = 200MB$	$\sim 23GB$
Initial Import	$20m * 150days \sim 3 * 10^9$	$1KB * 3 * 10^9 = 3TB$	$161B * 3 * 10^9 \sim 500GB$	$10B * 3 * 10^9 = 30GB$	$\sim 3.5TB$

TABLE II: IP Based and Port Based Tables

Table	Row Key					Query Type
T1	[sa]	[sp]	[da]	[dp]	[1 - ts]	Extended queries
T2	[da]	[dp]	[sa]	[sp]	[1 - ts]	
T3	[sa]	[da]	[sp]	[dp]	[1 - ts]	Source-Destination address queries
T4	[da]	[sa]	[dp]	[sp]	[1 - ts]	Source-Destination address queries
T5	[sp]	[sa]	[da]	[dp]	[1 - ts]	Service server discovery queries
T6	[dp]	[da]	[sa]	[sp]	[1 - ts]	Service server discovery queries
T7	[sp]	[da]	[sa]	[dp]	[1 - ts]	Service client discovery queries
T8	[dp]	[sa]	[da]	[sp]	[1 - ts]	Service client discovery queries

$$|record_{T1}| = |cq| * (|rk| + |cfn| + |cn|) + \sum_{i \in cq} |cv_i|) \quad (1)$$

$$|record_{T2-T8}| = (|rk| + |cfn|) \quad (2)$$

where:

$|x|$ = x's size in byte(s)
rk = row key (size = 23B)
cfn = column family name
cq = column qualifiers set
cn = column qualifier name
cv = column value

V. Implementation

A. Data Collection

A set of MapReduce jobs and scripts are developed for collecting, storing, and processing data in HBase and OpenTSDB[7]. In the MapReduce job, the map task read flow information files and prepare rowkeys as well as columns for all tables. In the next step they are written into the corresponding tables. After that another task checks data integrity by a simple row counting job. This verification is not fully reliable, but it is a basic step for the integrity check without scarifying performance.

Performance evaluation was performed by processing records of a single day. The day is chosen randomly from working days of 2013. The statistical characteristics of the chosen day represents properties of any other working days. The performance of the implementation is not satisfactory in this stage. For HBase, the maximum number of operations per second is 50 with the maximum operation latency of 2.3 seconds. HDFS shows the same performance issue, the maximum number of written bytes per second is 81 MB/s. The task is finished after 45.46 minutes. Therefore, a performance tuning is required.

[7]Available at: https://github.com/aryantaheri/netflow-hbase

B. Performance Tuning

The initial implementation of the collection module was not optimized for storing large datasets. By investigating performance issues, seven steps are recognized as remedies [26], [25]. These improvements will also enhance query execution process, and are applied there as well.

a) Using LZO compression: Although compression demands more CPU time, the HDFS IO and network utilization are reduced considerably. Compression is applied to store files (HFiles) and the algorithm must be specified in the table schema for each column family. The compression ratio is dependent on the algorithm and the data type, and for our dataset with the LZO algorithm the ratio is about 4.

b) Disabling Swap: Swappiness is set to zero on data nodes, since there is enough free memory for the job to complete without moving memory pages to the swap [26].

c) Disabling Write Ahead Log (WAL): All updates in a region server are logged in WAL, for guaranteeing durable writes. However, the write operation performance is improved significantly by disabling it. This has the risk of data loss in case of a region server failure [25].

d) Enabling Deferred Log Flush (DLF): DLF is a table property, for deferring WAL's flushes. If WAL is not disabled (due to the data loss risk), this property can specify the flushing interval to moderate the WAL's overhead [25].

e) Increasing heap size: 20TB of the disk storage is planned to be used for storing monitoring data. The formula for calculating the estimated ratio of disk space to heap size is: $RegionSize/MemstoreSize * ReplicationFactor * HeapFractionForMemstores$ [27]. This leads to a heap size of 10GB per region server.

f) Specifying Concurrent-Mark-Sweep Garbage Collection (CMS-GC): Full garbage collection has tremendous overhead and it can be avoided by starting the CMS process earlier. Initial occupancy fraction is explicitly specified to be 70 percent. Thus, CMS starts when the old generation allocates more than 70 percent of the heap size [26].

g) Enabling MemStore-Local Allocation Buffers (MSLAB): MSLAB relaxes the issue with the old generation heap fragmentation for HBase, and makes garbage collection pauses shorter. Furthermore, it can improve cache locality by allocating memory for a region from a dedicated memory area [28].

h) Pre-Splitting Regions: The pre-splitting of regions has major impact on the performance of bulk load operations. It can rectify the hotspot region issue and distribute the work load

TABLE IV: Initial region splits for tables T1-T4 (Store file size in MBytes-Number of store files)

Region	Starting IP address	T1	T2	T3	T4
1		30-1	0-0	0-0	5-1
2	17.17.17.17	23-1	0-0	0-0	0-0
3	34.34.34.34	32-1	6-1	5-1	0-0
4	51.51.51.51	172-1	22-1	21-1	22-1
5	68.68.68.68	325-1	57-1	57-1	57-1
6	85.85.85.85	77-1	11-1	10-1	11-1
7	102.102.102.102	85-1	9-1	13-1	0-0
8	119.119.119.119	57-1	11-1	0-0	11-1
9	136.136.136.136	102-1	11-1	10-1	11-1
10	153.153.153.153	543-1	92-1	82-1	97-1
11	170.170.170.170	21-1	0-0	0-0	0-0
12	187.187.187.187	887-1	138-1	141-1	139-1
13	204.204.204.204	73-1	11-1	10-1	11-1
14	221.221.221.221	5-1	0-0	0-0	1-1
15	238.238.238.238	0-1	0-0	0-0	0-0

TABLE V: Initial region splits for tables T5-T8 (Store file size in MBytes-Number of store files)

Region	Starting Port number	T5	T6	T7	T8
1		197-1	137-1	198-1	137-1
2	4369	7-1	0-0	0-0	0-0
3	8738	0-0	0-0	0-0	0-0
4	13107	0-0	0-0	0-0	0-0
5	17476	0-0	0-0	0-0	0-0
6	21845	0-0	9-1	8-1	0-0
7	26214	0-0	0-0	0-0	0-0
8	30583	0-0	0-0	0-0	10-1
9	34952	0-0	12-1	0-0	12-1
10	39321	0-0	13-1	10-1	12-1
11	43690	9-1	12-1	0-0	12-1
12	48059	37-1	49-1	38-1	60-1
13	52428	25-1	49-1	26-1	50-1
14	56797	25-1	37-1	25-1	38-1
15	61166	26-1	24-1	25-1	25-1

among all region servers. Each region has a start and an end rowkeys, and only serves a consecutive subset of the dataset. The start and end rowkeys should be defined such that all regions will have a uniform load. The pre-splitting requires a good knowledge of the rowkey structure and its value domain.

Tables T1-T4 start with an IP address, and T4-T8 have a port number in the lead position. Thus, they demand different splitting criteria. The initial splitting uses a uniform distribution function, and later it's improved by an empirical study. IPv4 space has 2^{32} addresses, and the address space is split uniformly over 15 regions, as shown in Table IV. Furthermore, port number is a 16 bit field with 65535 values and the same splitting strategy is applied for it, Table V.

The performance gain for storing a single day of flow information is considerable. On average, 754 HBase operations are performed in a second (x30 more operations/s), the average operation latency is decreased to 27 ms (x14 faster), and the job is finished in 15 minutes (x3 sooner). Despite high efficiency improvement, there are some hotspot regions which should be investigated more.

Tables IV and V show regions' start keys,the number of store files, and their sizes. It can be observed that splitting regions using the uniform key distribution function doesn't lead to a uniform load in regions.

In tables T1-T4, regions R4, R5, R10, R12 have big store files compared to the rest of regions. Highly loaded regions serve entries within the following IP address spaces (anonymized) : *R4 → [51.51.51.51, 68.68.68.68), R5 → [68.68.68.68, 85.85.85.85), R10 → [153.153.153.153, 170.170.170.170), R12 → [187.187.187.187, 204.204.204.204)*

By investigating these IP address blocks, we identified that some of them contains Norwegian address blocks[8] and some others are popular services providers. In addition, empty regions contain special ranges such as: private networks and link-local addresses.

In tables T5-T8, regions R1, R12, R13, R14, R15 have high loads, and they serve the following port numbers: *R1 → [0, 4369), R12 → [48059, 52428), R13 → [52428, 56797), R14 → [56797, 61166), R15 → [61166, 65536),*

For tables T5-T8, R1 covers well known ports (both system ports and user ports) suggested by Internet Assigned Numbers Authority (IANA)[9], and R12-R15 contains short-lived ephemeral ports (i.e. dynamic/private ports). In the empirical splitting, the difference between system ports, user ports, and private/dynamic (ephemeral) ports will be taken into account.

A large fraction of records have port numbers of popular services (e.g. HTTP(S), SSH) or IP addresses of popular sources/destinations (e.g. Norwegian blocks, popular services). Therefore, regions should not be split using a uniform distribution over the port number range or the IP address space. The splitting is improved by taking these constrains into consideration and the result is significant. The average number of operations per second is 1600 (x64 more), the latency is 5ms (x80 less), and the job duration is reduced to 6.57 minutes (x7.5 faster). The results are depicted in Figure 2.

VI. Evaluation

This section analyses several query types and their response times.

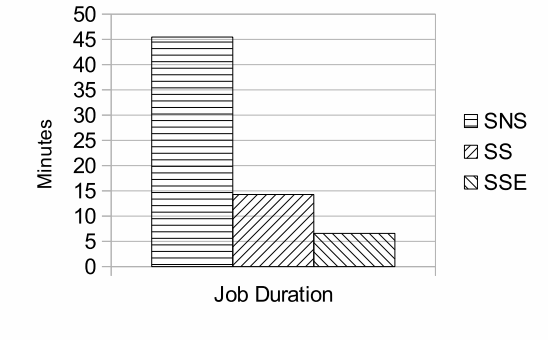
A. Top-N Host Pairs

Finding Top-N elements is a common query type for many datasets. In our dataset, elements can be IP addresses, host pairs, port numbers, etc. In the first evaluation, a query for finding Top-N host pairs is studied for a 150 days period. These pairs are hosts which have exchanged the most traffic on the network. The query requires processing of all records in the table T1, and aggregation of input and output bytes for all available host pairs, independent of the connection initiator and port numbers. Table T1 has 5 billion records.

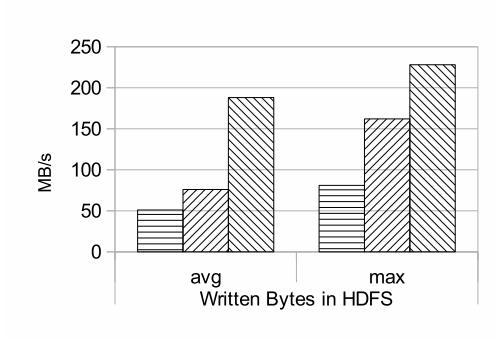
Traditional tools (e.g. NFdump) are not capable of answering this query, because the long period corresponds to an extremely large dataset. For this purpose, two chaining MapReduce jobs are written for HBase tables. The first one identifies host pairs and aggregates their exchanged traffic. The second one sorts pairs based on the exchanged traffic.

[8]http://drift.uninett.no/nett/ip-nett/ipv4-nett.html

[9]http://www.iana.org/



(a) Jobs finishing time



(b) HDFS IO

(c) Number of operations per seconds in HBase

(d) Operation latency in HBase

Fig. 2: Storage performance under different implementations (SNS: Single day processing without pre-splitting, SS: Single day processing with a uniform splitting function, SSE: Single day processing with an empirical pre-splitting function)

On average, the first job finishes after 26 minutes, and the second one after 19 seconds. These are reasonable duration for processing a large dataset, since, Top-N host pairs queries are not executed very frequently, and there is no real-time demand for them.

B. Service Server Discovery for a Given Period

This query type contains time filters which means a subset of the dataset in the given time range is of interest. The query can be executed using two methods. The first method uses HBase tables to retrieve records which satisfy non-time criteria (i.e. intermediate result). Then, compliant records with the time filter are returned as the target dataset. Finally, the target dataset is processed according to the query specifications. Since, time criteria can not be evaluated in each data node[10], the time range filtering is performed in a single node. Therefore, this method is inefficient when the intermediate result is large.

The second method benefits from OpenTSDB. OpenTSDB key structure simplifies accessing data within a time range, at the cost of storage volume and response time. This method retrieves records within the time frame, first. Then, other filters are evaluated and the final processing is performed. This approach is efficient for small periods when the estimated number of compliant records with all filters is high.

[10]Because the timestamp has a trailing position in the rowkey structure.

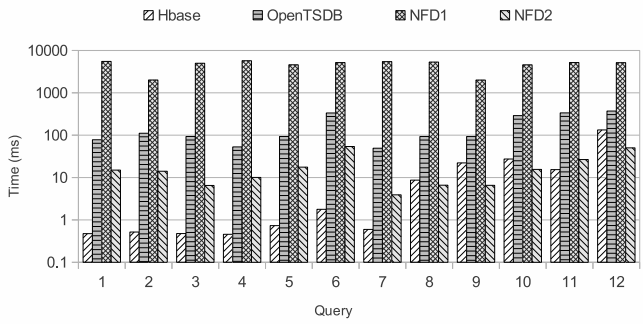


Fig. 3: Queries performance evaluation

Figure 3 depicts response times of several queries using multiple methods. The first two methods (i.e. *HBase*, and *OpenTSDB*) are explained earlier. The other methods use a traditional tool (i.e. NFdump) for retrieving and processing records. *NFD1* is executed over the complete dataset, and *NFD2* processes only a subset of the dataset which satisfies the time constrain. HBase outperforms OpenTSDB by an average factor of 87 and NFD1 by an average factor of 4472. Its performance is not comparable with NFD2, since NFD2 has a limited dataset.

VII. Conclusion

The paper proves the effectiveness of a data-intensive processing framework for delivering scalable and efficient network monitoring services. The proposed mechanism is not dependent on a specific network monitoring protocol, and it's applicable to any protocol as long as rowkey design criteria are satisfied. Data point structure is designed by careful analysis and conversion of monitoring record's fields. Therefore, the collection's process and storage volume overheads are reduced, and real-time data retrieval is accomplished.

Long-term queries are performed by MapReduce jobs and short-term queries are executed through available scanning APIs. These two accessing methods fulfil response time requirements of planned (e.g. statistical analysis, evidence gathering) and ad-hoc (e.g. forensics) activities.

Further Work

There are several areas which required further study and improvement such as: advanced query interface for network operators and researchers, embedded analytical engine for statistical studies, robust underlying infrastructure for enhanced availability, and integration with a cloud platform's network management service for a better real-time monitoring and security enforcement mechanisms using Software Defined Networking (SDN) technologies.

Acknowledgment

The authors would like to thank Olav Kvittem and Arne Oslebo from UNINETT and Martin Gilje Jaatun from SINTEF ICT, who provided valuable comments and assistance to the undertaking of this research.

References

[1] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope." ACM Press, 2003, p. 647. [Online]. Available: http://portal.acm.org/citation.cfm?doid=872757.872838

[2] European Parliament Council, "Directive 2002/58/EC," Jul. 2002. [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:EN:NOT

[3] B. Li, J. Springer, G. Bebis, and M. Hadi Gunes, "A survey of network flow applications," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, Mar. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1084804512002676

[4] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, p. 5, Jan. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2427036.2427038

[5] D. G. Andersen and N. Feamster, "Challenges and opportunities in internet data mining," *Parallel Data Laboratory, Carnegie Mellon University, Research Report CMU-PDL-06-102*, 2006.

[6] H. Balakrishnan, M. Balazinska, D. Carney, U. etintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik, "Retrospective on aurora," *The VLDB Journal*, vol. 13, no. 4, pp. 370–383, Dec. 2004. [Online]. Available: http://link.springer.com/10.1007/s00778-004-0133-5

[7] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic." ACM Press, 2009, p. 202. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1644893.1644918

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system." IEEE, May 2010, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5496972

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 29, Dec. 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1165389.945450

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, Jun. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1365815.1365816

[11] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "SCOPE: easy and efficient parallel processing of massive data sets," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, p. 12651276, 2008.

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad." ACM Press, 2007, p. 59. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1272996.1273005

[13] P. Schwan, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux Symposium*, vol. 2003, 2003.

[14] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, "PVFS: a parallel file system for linux clusters," in *In Proceedings of the 4th Annual Linux Showcase and Conference*. USENIX Association, 2000, p. 317327.

[15] L. George, *HBase: the definitive guide*. O'Reilly Media, Incorporated, 2011.

[16] E. A. Brewer, "Towards robust distributed systems," in *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, vol. 19, 2000, p. 710.

[17] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, p. 107, Jan. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1327452.1327492

[18] B. Claise, *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*, ser. Request for Comments. IETF, Jan. 2008, no. 5101, published: RFC 5101 (Proposed Standard). [Online]. Available: http://www.ietf.org/rfc/rfc5101.txt

[19] R. R. Kompella and C. Estan, "The power of slicing in internet flow measurement," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, ser. IMC '05. Berkeley, CA, USA: USENIX Association, 2005, p. 99. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251086.1251095

[20] B. Claise, *Cisco Systems NetFlow Services Export Version 9*, ser. Request for Comments. IETF, Oct. 2004, no. 3954, published: RFC 3954 (Informational). [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[21] P. Phaal and M. Lavine, "sFlow version 5," Tech. Rep., Jul. 2004. [Online]. Available: http://www.sflow.org/sflow_version_5.txt

[22] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon, "Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme," *Computer Networks*, vol. 46, no. 2, pp. 253–272, Oct. 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128604001197

[23] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics." ACM Press, 2006, p. 159. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1177080.1177101

[24] V. Carela-Espaol, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Sol-Pareta, "Analysis of the impact of sampling on NetFlow traffic classification," *Computer Networks*, vol. 55, no. 5, pp. 1083–1099, Apr. 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128610003439

[25] Apache Software Foundation, "The apache HBase reference guide," http://hbase.apache.org/book.html, 2012. [Online]. Available: http://hbase.apache.org/book.html

[26] Y. Jiang, *HBase Administration Cookbook*. Birmingham: Packt Publishing, Limited, 2012. [Online]. Available: https://ezproxy.siast.sk.ca:443/login?url=http://proquest.safaribooksonline.com/9781849517140

[27] Lars Hofhansl, "HBase region server memory sizing," Jan. 2013. [Online]. Available: http://hadoop-hbase.blogspot.no/2013/01/hbase-region-server-memory-sizing.html

[28] Todd Lipcon, "Avoiding full GCs in apache HBase with MemStore-Local allocation buffers," Mar. 2011. [Online]. Available: http://blog.cloudera.com/blog/2011/03/avoiding-full-gcs-in-hbase-with-memstore-local-allocation-buffers-part-3/