DBStream: an Online Aggregation, Filtering and Processing System for Network Traffic Monitoring

Arian Bär, Pedro Casas
FTW - Telecommunications Research Center Vienna
{baer, casas}@ftw.at

Lukasz Golab
University of Waterloo
lgolab@uwaterloo.ca

Alessandro Finamore
Politecnico di Torino
finamore@tlc.polito.it

***Abstract*—Network traffic monitoring systems generate high volumes of heterogeneous data streams which have to be processed and analyzed with different time constraints for daily network management operations. Some monitoring applications such as anomaly detection, performance tracking and alerting require fast processing of specific incoming real-time data. Other applications like fault diagnosis and trend analysis need to process historical data and perform deep analysis on generally heterogeneous sources of data. The Data Stream Warehousing (DSW) paradigm provides the means to handle both types of monitoring applications within a single system, providing fast and rich data analysis capabilities as well as data persistence. In this paper, we introduce DBStream, a novel online traffic monitoring system based on the DSW paradigm, which allows fast and flexible analysis across multiple heterogeneous data sources. DBStream provides a novel stream processing language for implementing data processing modules, as well as aggregation, filtering, and storage capabilities for further data analysis. We show multiple traffic monitoring applications running on DBStream, processing real traffic from operational ISPs.**

***Keywords*—DBStream; Data Stream Warehousing; Network Traffic Monitoring and Analysis**

I. Introduction

The complexity of large-scale, Internet-like networks is constantly increasing. With more and more services being offered on the Internet, the massive adoption of Content Delivery Networks (CDNs) for traffic hosting and delivery, and the continuous growth of bandwidth-hungry video-streaming services, network and server infrastructures are becoming extremely difficult to understand and to track. Network Traffic Monitoring and Analysis (NTMA) has taken an important role to understand the functioning of such networks, especially to get a broader and clearer visibility of unexpected events. The evolution of the Internet calls for better and more flexible measurement and monitoring systems to pinpoint problems and optimize service quality.

A variety of methodologies and tools have been devised by the research community to passively monitor network links. Technologies such as NetFlow and more advanced monitoring solutions [1]–[3] enable the monitoring of high-speed links using off-the-shelf hardware. Similarly, several solutions are available for active measurements, from simple command-line tools such as the standard `ping` to more advanced frameworks for topology discovery such as TopHat [4]. All these tools are stand-alone solutions, capable of extracting large amounts of detailed information from live networks. What is sorely missing is a flexible system able to store and process such rich and heterogeneous sources of network monitoring data in order to understand the complicated dynamics of nowadays Internet. Such a system should be capable of handling different types of NTMA applications, from real-time or near real-time data processing applications such as service performance tracking and anomaly detection and alerting, to more complex big data analysis tasks involving the processing of large amounts of stored historical data. The Data Stream Warehousing (DSW) paradigm [9] provides the means to handle both types of monitoring applications within a single system, combining the real-time data processing of data stream management systems with the deep analytics of long historical data of traditional warehouses.

978-1-4799-0959-9/14/$31.00 © 2014 IEEE

In this paper we introduce DBStream, a flexible and scalable DSW system tailored to NTMA applications. DBStream is a repository system capable of ingesting data streams coming from a wide variety of sources (e.g., passive network traffic data, active measurements, router logs and alerts, etc.) and performing complex continuous analysis, aggregation and filtering jobs on them. DBStream can store tens of terabytes of heterogeneous data, and allows both real-time queries on recent data as well as deep analysis of historical data. Figure 1 shows a standard deployment of DBStream as part of a generic network monitoring architecture for current Internet-like networks, consisting of both passive and active probes, as well as external data sources provided by other data repositories.

One of the main assets of DBStream is the flexibility it provides to rapidly implement new NTMA applications, through the usage of a novel stream processing language tailored to continuous network analytics. Advanced analytics can be programmed to run in parallel and continuously over time, using just a few lines of code. The near real-time data analysis is performed through the online processing of time-length configurable batches of data (e.g., batches of one minute of passive traffic measurements), which are then combined with historical collections to keep a persistent collection of the output. Moreover, the processed data can then be easily integrated into visualization tools (e.g., web portals).

To exemplify the kind of NTMA applications which can run on top of DBStream, we present in this paper four different traffic processing applications, considering the incoming data from passive probes installed both at the core of a mobile network and at the edge of a fixed-line ADSL/FTTH network of major European ISPs. In addition, we evaluate the performance of DBStream in processing real traffic measurements, and compare it both with standard PostgreSQL data repositories,

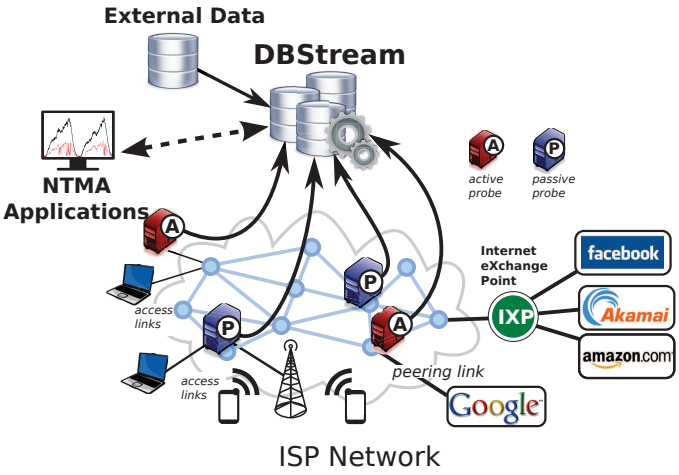


Fig. 1. A standard deployment of DBStream in an ISP network. DBStream is a data repository capable of processing data streams coming from a wide variety of sources.

as well as MapReduce-based frameworks.

The remainder of this paper is organized as follows: Related work is briefly discussed in Sec. II. Sec. III introduces DBStream, describing its design and implementation. System performance and scalability considerations are discussed in Sec. IV. Sec. V shows four different NTMA applications running on DBStream, processing traffic measurements from two different operational networks. Finally, Sec. VI concludes the paper.

II. Related Work

Several technologies are available to potentially implement a data repository system like DBStream, which we can coarsely divide into SQL and NoSQL systems [5]. The former class includes Database Management Systems (DBMSs), which are known to offer excellent performance when accessing the data, but suffer when new data have to be inserted continuously. The latter class makes its selling point by offering great horizontal scalability, but offers no guarantee on the response time. NoSQL systems include MapReduce [10] systems, supporting a simpler key-value interface rather than the relational/SQL model used by DBMSs. Hadoop [11] and Hive [12] are two popular MapReduce technologies. MapReduce systems are based on batch processing rather than on stream processing, which is specifically required in NTMA applications.

There has been a great deal of effort to improve traditional DBMSs in the last several years. Many data processing and storage systems have been developed to improve both performance and scalability. Still, a major limitation of such systems is the inability to cope with continuous analytics. Some new solutions have been proposed, including Data Stream Management Systems (DSMSs) and Data Stream Warehouses (DSW). DSMSs enable continuous processing of data over time; examples include Gigascope [6] and Borealis [7]. These systems consist of in-memory operations with no persistent data storage, which is a critical limitation for traffic analysis purposes. DSWs extend DBMSs with the ability to ingest new data in nearly real-time. DataCell [8] and DataDepot [9] are two examples, as well as the DBStream system presented in this paper. Finally, hybrid systems composed of a mix of SQL and NoSQL technologies have been proposed, for example HadoopDB [13].

None of these systems were designed to address continuous network monitoring applications. The only exception is DataDepot, which is a closed-source system based on proprietary technologies. Furthermore, to the best of our knowledge, only DBStream supports incremental queries defined through a declarative language such as SQL, which are particularly useful for tracking the status of a network.

III. System Overview

DBStream is a novel continuous analytics system. Its main purpose is to process and combine data from multiple sources as they are produced, create aggregations, and store query results for further processing by external analysis or visualization modules. The system targets continuous network monitoring but it is not limited to this context. For instance, smart grids, intelligent transportation systems, or any other use case that requires continuous process of large amounts of data over time can take advantage of DBStream.

DBStream combines on-the-fly data processing of DSMSs with the storage and analytic capabilities of DBMSs and typical big data analysis systems such as Hadoop. In contrast to DSMSs, data are stored persistently and are directly available for later visualization or further processing. As opposed to traditional data analytics systems, which typically import and transform data in large batches (e.g., days or weeks), DBStream imports and processes data in small batches (e.g., on the order of minutes). Therefore, DBStream resembles a DSMS in the sense that data can be processed quickly, but streams can be re-played from past data. The only limitation is the size of available storage. DBStream thus supports a native concept of time. At the same time DBStream provides a flexible interface for data loading and processing, based on the declarative SQL language used by all relational DBMSs.

Two salient features of DBStream are the following: first, it supports incremental queries defined through a declarative interface based on the SQL query language. Incremental queries are those which update their results by combining newly arrived data with previously generated results rather than being re-computed from scratch. This enables continuous time-series based data analysis, which is a strong requirement for real-time NTMA applications such as anomaly detection. Secondly, in contrast to many database system extensions, DBStream does not change the query processing engine. Instead, queries over data streams are evaluated as repeated invocations of a process that consumes a batch of newly arrived data and combines them with the previous result to compute the new result. Therefore, DBStream is able to reuse the full functionality of the underlying DBMS, including its query processing engine and query optimizer.

DBStream is built on top of a SQL DBMS back-end. We use the PostgreSQL database in our implementation, but the DBStream concept can easily be used with other databases and it is not dependent on any specific features of PostgreSQL.

A. System Architecture

In DBStream, *base tables* store the raw data imported into the system, and *materialized views* (or views for short) store

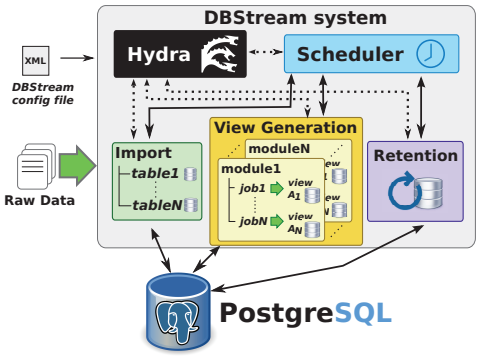


Fig. 2. General overview of the DBStream architecture. DBStream combines on-the-fly data processing of DSMSs with the storage and analytic capabilities of DBMSs and big data analysis systems such as Hadoop.

the results of queries such as aggregates and other analytics — which may then be accessed by ad hoc queries and applications in the same way as base tables. Base tables and materialized views are stored in a time-partitioned format inside the PostgreSQL database, which we refer to as Continuous Tables (CT). Time partitioning makes it possible to insert new data without modifying the entire table; instead, only the newest partition is modified, leading to a significant performance increase.

A *job* defines how data are processed in DBStream, having one or more CTs as input, a single CT as output and an SQL query defining the processing task. An example job could be: *"count the distinct destination IPs in the last 10 minutes"*. This job would be executed whenever 10 new minutes of data have been added to the input table (independently of the wall clock time) and stored in the corresponding CT.

Figure 2 gives a high-level overview of the DBStream architecture. DBStream consists of a set of modules running as separate operating system processes. The `Scheduler` defines the order in which jobs are executed, and besides avoiding resource contention, it ensures that data batches are processed in chronological order for any given table or view. `Import` modules may pre-process the raw data if necessary, and signal the availability of new data to the `Scheduler`. The scheduler then runs jobs that update the base tables with newly arrived data and create indices, followed by incrementally updating the materialized views. Each view update is done by running an SQL query that retrieves the previous state of the view and modifies it to account for newly arrived data; new results are then inserted into a new partition of the view, and indices are created for this partition. `View Generation` modules register jobs at the `Scheduler`. Finally, the `Retention` module is responsible for implementing data retention policies. It monitors base tables and views, deleting old data based on predefined storage size quotas and other data retention policies. Since each base table and view is partitioned by time, deleting old data is simple: it suffices to drop the oldest partition(s).

The DBStream system is operated by an application server process called `hydra`, which reads the DBStream configuration file, starts all modules, and monitors them over time. Status information is fetched from those modules and made available in a centralized location. Modules can be placed on separate machines, and external programs can connect directly to DBStream modules by issuing simple HTTP requests.

The DBstream system features a simple processing language. Below we show an example of a typical aggregation query, counting the number of rows per minute and device class. If the input table A has one flow in each row, the number of rows corresponds to the number of flows.

```
<job inputs="A (window 15min)"
     output="B (window 15min)"
  schema="time int4, dev_class int4, cnt int4">
  <query>
  select time - time%1min, dev_class, count(*)
    from A
    group by serial_time, dev_class
  </query>
</job>
```

In more detail, the XML attribute `inputs` is used to define one or more input streams. For each input stream, the batch size is specified with a `window` definition; in the example, the window size is 15 minutes. The `output` attribute is used to specify an output stream, which then can be used as input to other queries. The output stream also has a `window` definition. In addition, for the output stream, the `schema` is defined as the set of data types returned by the query. Note that the first column must be a monotonically increasing timestamp, which is used in the window definitions. Inside the `query` XML element, an SQL query defines how the input(s) should be processed. The result of this query is then stored in the new window of the output table. In the query, all features of PostgreSQL, including the very flexible User Defined Functions (UDF)s, can be used to process the data. Utilizing UDFs, it is easy to add code written in Python, Perl, C, R and other programming languages into the query.

In particular, when defining the query to compute a new window of an output table based on a new window of an input stream, it is possible to reference the *previous* window of the output table in addition to the new window of the input stream. This is useful when, e.g., computing cumulative counts and sums such as upload and download volumes over long periods of time. In this case, it suffices to add the volumes from the new input window to the cumulative sums maintained in the (previous window of the) output table. We call these queries *incremental queries* in the remainder of this paper.

IV. Performance Benchmarking

In this section, we compare DBStream implemented on top of PostgreSQL version 9.2 with standard PostgreSQL version 9.2. We perform three tests: a simple benchmark measuring the overhead of DBStream, a simple workload that illustrates the benefits of the scheduler, and a more complex query that illustrates the performance benefits of incremental processing. We use a 10 day-long data set collected at the vantage point of a major European ISP, which corresponds to 496 GB of data in plain text files and 703 GB in DBStream, containing 1.052 billion TCP connections. DBStream runs on a single server, equipped with a XEON E5 2640 2.5 GHz, 32 GB of RAM and four 2TB hard disks running in a RAID10 configuration.

The first test is a very simple query counting the number of flows in one day. It acts as a baseline for the hardware performance of the server as well as showing the overhead of DBStream compared to PostgreSQL. Note that running

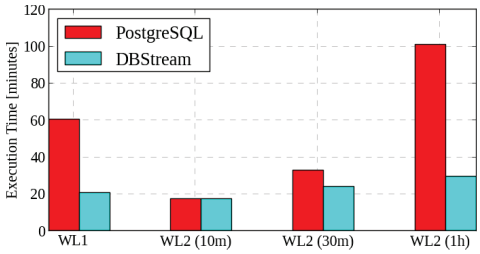


Fig. 3. DBStream performance vs. PostgreSQL. The `Scheduler` and the usage of incremental queries significantly improve performance.

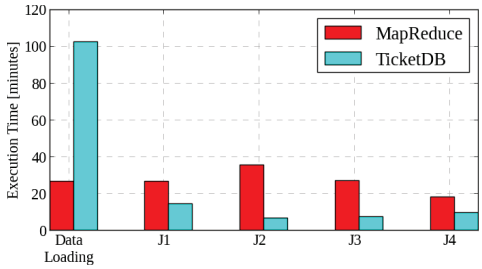


Fig. 4. Summary of processing job and import durations for TicketDB, the predecessor of DBStream.

a single query means that the DBStream scheduler is not necessary. In the considered day, the data amounts to 55 GB, corresponding to 82 million TCP connections. DBStream processed approximately 268 thousand rows per second and PostgreSQL 269 thousands; this is also reflected in the processed MB/s, which is 179 MB/s for DBStream and 180 MB/s for PostgreSQL. This test shows that the overhead of DBStream is minimal with respect to standard PostgreSQL.

The second test considers a more typical use case for DSWs and is meant to illustrate the need for a scheduler. Given one day of traffic data, we compute various aggregate statistics on HTTP traffic. We create 5 views, all with a window size of 1 minute. The first, call it A, contains only the interesting set of columns corresponding to HTTP flows. For example, we discard all P2P traffic and fetch organization names from the MaxMind database using a join query. This materialized view amounts to 4 GB. From view A, we generate four derived views, B, C, D and E, which can directly be used for visualization. These contain percentiles of the HTTP traffic statistics we are interested in: per-connection uploaded bytes in B, downloaded bytes in C, minimum Round Trip Times in D, and server elaboration time in E.

In PostgreSQL, we first load the whole day's worth of data into table A and then run the queries corresponding to the other views and save their results in the respective tables. In DBStream, all views are formulated as jobs with input windows of 10 minutes and created by "replaying" the same day of data letting the DBStream `Scheduler` propagate the changes to all the views. As shown in Figure 3 – Workload 1, the roughly three-times better performance of DBStream is achieved by parallelization. Since the `Scheduler` is aware of the precedence constraints among the views, whenever one 10 minutes window of A is loaded, the corresponding partitions of B, C, D and E can be computed in parallel, and at the same time, processing of the next window of A can start.

In the last experiment, we evaluate the efficiency of incremental queries, as compared to re-computing the results from scratch, which is done by default in PostgreSQL. We define a job which computes all the active IPs over a moving window. We assume that each window is 1 minute long, and test three variants of this query: finding all the active IPs within the past 10 minutes, 30 minutes and 60 minutes. Intuitively, as the length of the sliding window referenced by the query increases, we expect the performance advantage of incremental processing to increase. As shown in Figure 3 – Workload 2, the advantage of DBStream for 10-minute windows is only marginal; however, for 30-minute windows DBStream is noticeably faster, and for 60-minute windows it is over three times as fast as PostgreSQL.

To conclude the performance evaluation of DBStream, and for the sake of completeness, we present in Table 4 the performance comparison of TicketDB against the popular MapReduce system Hadoop, reported in our previous work [15]. TicketDB is the predecessor of the DBStream system, which does not include the `Scheduler` and `View Generation` modules, but already uses similar table partitioning techniques. Besides the new modules, DBStream uses de-normalized tables and is more optimized than TicketDB. Figure 4 reports the duration of four different jobs running both on a Hadoop cluster and on TicketDB. The dataset used in this evaluation consists of real network traces captured at the well-known WIDE network[1]. A total of 100 GB of data is imported. The four jobs cover typical network monitoring tasks such as: byte counts per IP (J1), ranking IPs by volume (J2), connection counts (J3) and counts of unanswered TCP handshakes (J4). The hardware characteristics of both environments are comparable in terms of equipment costs, but the main difference is that TicketDB runs on a single machine, whereas Hadoop runs on a cluster of 11 machines deployed on Amazon EC2; see [15] for the complete details. The reported results show that once the data are loaded into the system, TicketDB clearly outperforms the Hadoop cluster. This suggests a potentially much higher performance of DBStream compared to Hadoop, even if further benchmarking should be conducted to be conclusive.

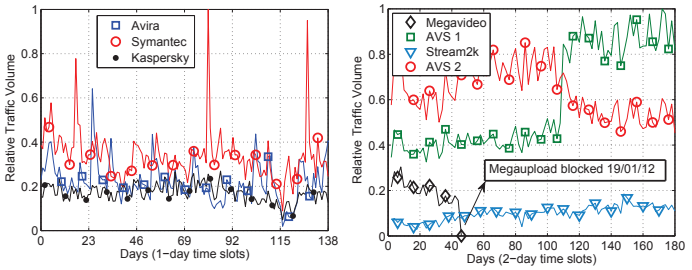
V. NTMA Applications in DBStream

We now briefly overview four different NTMA applications currently running on top of DBStream, which exemplify the kind of analysis it targets. The four applications analyze traffic packets passively observed at two different operational ISPs. The first ISP corresponds to a mobile network operator, and the traffic is captured at the standard Gn data interface. The second ISP corresponds to a fixed-line operator, and traffic is captured at a Point-of-Presence aggregating about 45,000 end-users connected to the Internet, either through ADSL or FTTH access technologies. The four applications include: (i) HTTP traffic classification [16], (ii) YouTube Quality of Experience (QoE) monitoring [17], (iii) CDN-based traffic tracking and analysis and (iv) on-line anomaly detection in end-user performance.

A. Traffic Classification for Trend Analysis

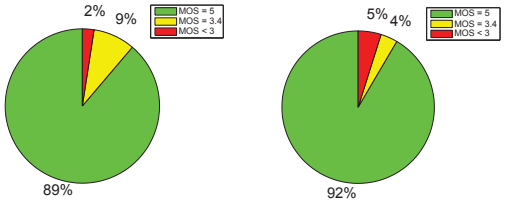
Figure 5 depicts two tracking applications of HTTPTag, an on-line HTTP traffic classification system running on

[1] http://www.wide.ad.jp/



(a) 5-months anti-virus services tracking (b) 6-months video services tracking

Fig. 5. HTTPTag classification coverage and some long-term tracking examples revealing different events of interest in an operational 3G network.



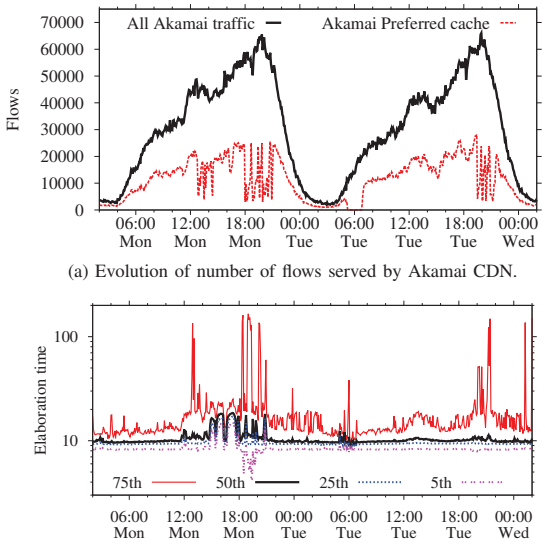
(a) MOS distribution (tickets). (b) MOS distribution (video time).

Fig. 6. YouTube QoE monitoring on a mobile network.

DBStream in a mobile network. HTTPTag classifies HTTP flows by pattern matching, applied to the hostname field of the HTTP requests. Figure 5(a) reports the traffic generated by three popular anti-virus services (Symantec, Kaspersky, and Avira) over a period of four months (from 26/05/12 to 15/10/12). Analyzing the traffic patterns over a sufficiently long period gives for example a good image of the different approaches the three companies use to manage software and virus-definition updates. Figure 5(b) depicts a comparison of four video streaming services over a 6-month period (from 1/12/11 to 25/05/12): Megavideo, Stream2k, and two adult video services (AVS 1 and 2). After 46 days from the starting day, `Megavideo` traffic completely disappears, which correlates to the well-known shut-down of the Megaupload services on 19/01/12. Having visibility over such variations and trends allows the network operator to better optimize his network, by defining for example specific content caching policies to reduce the load on the core links, different routing, load balancing, or prioritization/shaping policies.

B. YouTube QoE Monitoring

Figure 6 depicts the QoE experienced by users watching YouTube videos on the aforementioned mobile network, as reported by YOUQMON during one hour of traffic monitoring. YOUQMON is an on-line monitoring module running on DBStream, capable of assessing the QoE experienced by users watching YouTube videos, using network-layer measurements processing. Every 60 seconds, and for every YouTube video detected in a stream of packets, DBStream generates a ticket with the estimated QoE of the user, using a standard 5-points MOS scale, where 5 means perfect QoE, and 1 means unacceptable quality. The charts in Figs. 6(a) and 6(b) depict the distribution of number of tickets and total video played time at three different QoE classes: MOS = 5 (perfect QoE), MOS = 3.4 (poor QoE) and MOS $<$ 3 (bad QoE). The charts show that YouTube QoE in this network is excellent for about 90% of the issued tickets and of the video time consumed



(a) Evolution of number of flows served by Akamai CDN.

(b) Evolution of server elaboration time percentiles for the preferred cache.

Fig. 7. Tracking the behavior of Akamai. Strong variations in the number of flows served by the preferred cache might be caused by overloaded Akamai servers.

during the analyzed hour. For 9% of the issued tickets and 4% of the total video time, the quality achieved was rather poor, and bad for about 2% of the generated tickets. Using this monitoring application, the operator can have a clear view of the performance of the mobile network with respect to the satisfaction of the customers watching YouTube videos.

C. Tracking CDNs

Understanding the way traffic is served by large CDNs such as Akamai is useful for ISPs because of the dynamic server/cache selection policies they use, which might cause important traffic shifts inside the ISP boundaries in just a few minutes. Figure 7 reports the on-line tracking of the number of flows served by Akamai in the fixed-line network, using 5-minute time windows. Akamai flows are identified in DBStream by using the information provided at the aforementioned MaxMind database. Figure 7(a) focuses on a single /25 subnet hosting Akamai serves (the "preferred cache"), which provides the majority of the flows in this network. The preferred cache serves about 30% of the traffic at peak time. Surprisingly, traffic served by the preferred cache incurs occasional drops. These are the effects of the CDN server selection policies shifting traffic back and forth among CDN nodes. Figure 7(b) reports the evolution of the 5th, 25th, 50th, and 75th percentiles of the elaboration time[2] for the preferred cache servers. The abrupt increase in the elaboration time might indicate that the sudden drops in the number of served flows are caused by overloaded Akamai servers.

DBStream is used to group rows by service names[3]. As before, we consider time bins of 5 minutes, and for each service name we compute the fraction of requests served by the preferred and other caches. The obtained values are represented by the heatmap shown in Fig. 8. We selected the most popular

[2]The time between the client first packet with payload, and the server first packet with payload.

[3]Some string pre-processing is applied to group together FQDN such as *a1.da1.facebook.akamai.net*, *a2.da1.fecebook.akamai.net*, etc.

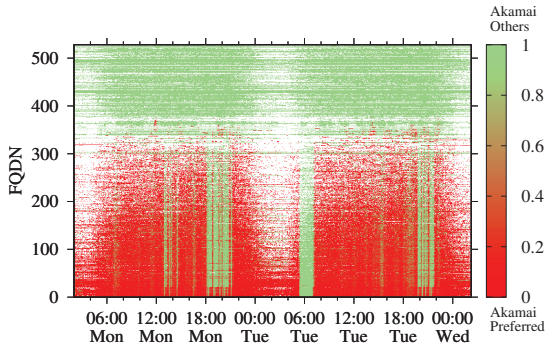


Fig. 8. Evolution of the volume of requests per service name (62 seconds of query execution time).

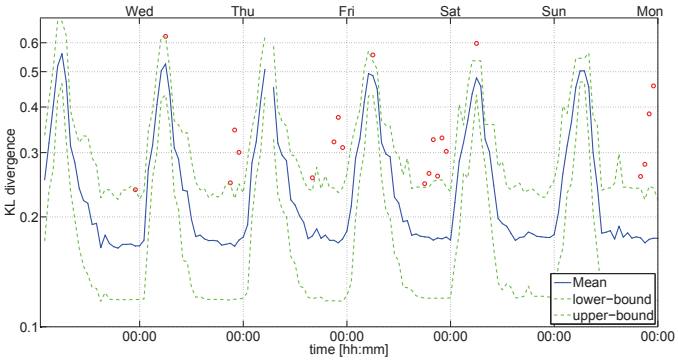


Fig. 9. Detection of anomalies in YouTube traffic. Alarms and acceptance region for the distribution of video flows average download rate. The red markers correspond to the flagged anomalies.

services, and sorted them by the probability of being served by the preferred cache. The results clearly show two groups: the bottom 300 services are normally served by some server at the preferred cache (red dots). The other 200 services are served exclusively by other Akamai CDN servers (green dots). Services not accessed any more during off peak time are left white. At the same time as the traffic shifts occur, practically all services are migrated to other caches, indicated by the green vertical bars in the plot. Only a group of about 20 services is never migrated, except during the aforementioned 2 hours gap on Tuesday, where all traffic is shifted to other caches.

D. Anomaly Detection

The last NTMA application consists of an on-line Anomaly Detection (AD) module. Figure 9 reports the output of the AD module when monitoring the downlink rate of users watching YouTube videos at the fixed-line network. The red markers in the plot correspond to alarms, which flag a large drop in the overall distribution of the per-user downlink throughput. Only YouTube flows are analyzed by the AD module, which in this case are pre-filtered by the passive probe capturing the traffic [2]. From Wednesday onward, the AD module systematically rises alarms at peak hours, between 21:00 and 23:00. Further analysis of these flagged anomalies revealed that the throughput drops might have been caused by the server selection policies used by Google, as the selected servers were apparently not correctly dimensioned to handle the traffic load at peak time.

VI. Concluding Remarks

In this paper we introduced DBStream, a novel Data Stream Warehouse (DSW) for online processing of data streams, typical for network monitoring environments. DBStream offers a flexible language that not only allows easy creation of materialized views, but also incremental queries that can merge their own past output with newly arrived data. This simplifies the writing of typical jobs for network measurement analysis and can improve processing performance threefold. The DBStream scheduler automatically resolves precedence constraints and can therefore run independent queries in parallel to speed-up the processing time. We have evaluated and benchmarked the performance of DBStream against standard SQL and NoSQL systems, showing promising results. Finally, we have presented different NTMA applications running on DBStream, processing traffic measurements from two different operational networks.

Acknowledgments

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n.318627 (Integrated Project mPlane). All members of FTW are supported by the Austrian Government and by the City of Vienna within the competence center program COMET.

References

[1] K. Keys et al., “The architecture of CoralReef: an Internet traffic monitoring software suite”, in *PAM*, 2001.

[2] A. Finamore et al., “Experiences of internet traffic monitoring with tstat”, in *IEEE Network*, vol 25(3), pp. 8–14, 2011.

[3] F. Fusco et al., “High speed network traffic analysis with commodity multi-core systems”, in *ACM IMC*, 2010.

[4] T. Bourgeau et al., “TopHat: supporting experiments through measurement infrastructure federation”, in *TridentCom*, 2010.

[5] R. Cattell, “Scalable SQL and NoSQL data stores”, in *ACM SIGMOD*, 2011.

[6] C. Cranor et al., “Gigascope: a stream database for network applications”, in *ACM SIGMOD*, 2003.

[7] D. Abadi et al., “Aurora: a new model and architecture for data stream management”, in *VLDB Journal*, vol. 12(2), pp. 120–139, 2003.

[8] E. Liarou et al., “MonetDB/Datacell: online analytics in a streaming column-store”, in *VLDB*, 2012.

[9] L. Golab et al., “Stream Warehousing with DataDepot”, in *ACM SIGMOD*, 2009.

[10] J. Dean et al., “Mapreduce: simplified data processing on large clusters”, in *Commun. ACM*, vol. 51(1), pp. 107–113, 2008.

[11] T. White, “Hadoop: the definitive guide”, *O’Reilly*, 2012.

[12] A. Thusoo et al., “Hive - a petabyte scale data warehouse using hadoop”, in *ICDE*, 2010.

[13] A. Abouzeid et al., “HadoopDB: an architectural hybrid of mapreduce and dbms technologies for analytical workloads”, in *VLDB*, 2009.

[14] A. Arasu et al., “The CQL continuous query language: semantic foundations and query execution”, in *VLDB Journal*, vol. 15(2), pp. 121–142, 2006.

[15] A. Bär et al., “Two parallel approaches to network data analysis”, in *LADIS*, 2011.

[16] P. Fiadino et al., “HTTPTag: A Flexible On-line HTTP Classification System for Operational 3G Networks”, in *INFOCOM*, 2013.

[17] P. Casas et al., “YOUQMON: A System for On-line Monitoring of YouTube QoE in Operational 3G Networks”, in *IFIP Performance*, 2013.