2015 IEEE International Congress on Big Data

Enabling Privacy Mechanisms in Apache Storm

Christian Schaefer
Ericsson
Stockholm, Sweden
Email: christian.schaefer@ericsson.com

Manoj P M
Ericsson
Chennai, India
Email: manoj.m.p@ericsson.com

***Abstract*—To analyze data that is streamed into a real-time computation system has gained traction and is very useful in use cases where for example telecom networks should be optimized dynamically. For this analysis lots of data i.e., big data is used. This nevertheless also poses privacy risks as this data usually also contains personal data and not only core applications of an organization want to access the big data pool but also other applications. In such cases the goal is to control the access to the data and be able to impose conditions on the data access if it contains personal data.**

This paper achieves this goal by contributing a privacy policy framework which controls which data in a real-time computation system like Apache Storm can be accessed and under which conditions. It allows customers to specify their privacy policies on how their private data is to be treated and organizations to specify their policies according to law and their business needs. Additionally, it is able to check that the programs submitted to this real-time computation system are able to access only those data streams having been approved by the organization running the real-time computation system.

I. Introduction

Big data gained more traction in recent years and especially analytics of big data. Streaming analytics tries to provide insights that benefit business and society in a timely manner but it also brings big privacy concerns as the data being streamed in for analytics also contains personal data.

In any enterprise, ensuring privacy on critical data is extremely essential. In the telecom world, there is a strong need for security and privacy. For e.g., assume an enterprise has collected data on expenditure, income and profit/loss of various products owned by the enterprise and various teams employed by the enterprise. This information should be available as such in the raw form to the executive management as such for financial forecast of the next upcoming financial year and to decide on investments. It is extremely important that this information should be selectively available for the middle management. For e.g., a team manager should be able to access only the financial status of his team, but he should be able to see basic information of performance of other teams/products. Whereas, the facilities team should completely have no access to this financial data as there is a very high risk of critical information leakage, but should have access certain fields like employee name to run queries to fetch the headcount of a floor in the office, to better plan infrastructure and facilities. Queries can come to the database containing the information, at real time from all the above mentioned departments. It is extremely critical to transform the results based on the organizational policies and expose transformed query results to respective departments.

Similarly in telecom there could be multiple applications like customer care, network optimization and third party marketing applications, querying the database or crunching data from the database or live data from the network. For e.g., In a typical case of recharging a prepaid mobile account, a subscriber will call the customer care in case the recharge voucher which the subscriber purchased, didn't work. In such cases, it is extremely important that the customer care representative has no access to the voucher code. In case, the voucher code is exposed as such to the customer care representative, there are high chances that he might himself use the code or leak the code to friends. In this case, the customer care representative forwards the complaint to the operator. The operator should have full access to the voucher code to be able to fix errors in voucher generation. There are huge number of such support calls received by every customer care office of an operator everyday and each call will make the customer care agent send a real time query to the subscriber database. This typical case portrays the volume of real time queries from applications and the necessity in ensuring security and privacy. Similarly, in the case of a third party marketing application, it is extremely vital that it should not have access to critical fields like address, age and sex in the subscriber database or location data coming directly from the network. In case, the third party marketing application should have access to these fields, the fields should be anonymized as much as possible not to risk the privacy of individuals. The network optimization case is a little bit different as it requires access to location data which is coming directly from the network but does not require identifiers of individual users thus these identifiers should be anonymized. All these use cases show that it is extremely vital in telecom to enable and ensure security and privacy.

For all the above mentioned use cases, it is extremely important that the results are processed at real time. For example, in the customer care use case, when a subscriber calls the customer care immediately after a recharge failure to know about the status, the customer care database should already be updated with the recharge event of the customer which happened just 2 minutes back. Similarly, in the network optimization use case, supposing an alarm is raised in a network element, the alarms and resolution mechanisms should be processed instantaneously before the alarm creates an effect in the adjacent network elements thereby causing

978-1-4673-7278-7/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataCongress.2015.24

more network problems. In the marketing use case, most third party applications try to query the subscriber database to get a report with details of the top 'n' influential users. The top 'n' influential users always change with time. Hence it is extremely vital in all the above mentioned use cases that the data should be processed almost instantaneously at real time as soon as the input is generated. Apache Storm is a well known open source framework which can be used to process data at real time with low latency.

The use cases introduced above have relative simple conditions for access to data but there might be use cases with combinations of several conditions or different conditions not mentioned above. These different conditions can also be seen as different policies that are representing end user policies or business rules respective legal requirements.

A. Problem description

The problems that the above use cases show and which are dealt with in this paper are that it is desirable to restrict access to data sources and make sure that only authorized users can access these data sources. This authorization problem includes that currently anyone can submit topologies to compute data without restrictions when Apache Storm is being used as the real-time computation system. Thus it would be helpful to be able to check the submitted topology before it is being executed.

Another problem mentioned in the use cases above is that also conditional access to data should be possible. This means that depending on a policy it should be possible to anonymize, filter, aggregate, encrypt or do other modifications with sensitive data before it is being used in the computation. Despite these modifications to sensitive data it should be abstracted away to the topology developer so that there are no changes necessary in how the topology is written.

B. Contribution

To solve the problems mentioned above the policy framework introduced in this paper adds to an Apache Storm cluster:

- privacy policies,
- a security gateway,
- and a modified Apache Storm.

The privacy policies specify the privacy rules that have to be enforced in the Apache Storm cluster. The security gateway added to the Apache Storm cluster enforces the privacy policies by controlling which (pre-approved) topologies can be submitted to Storm and which users are allowed to submit which topologies. Additionally, Apache Storm is modified to enforce the privacy policies by accepting topologies only from the security gateway and by checking if a topology is allowed to access the data it wants to use. It can also filter out data for individual users if requested. Furthermore, privacy enhancing technologies like anonymization, encryption and others can be applied before the data is used for computation in the Storm cluster to enable conditional access to data.

The paper is organized as follows. In Section II some background information is given and the assumptions for the paper. This is followed by a description of the solution suggested in Section III. Implementation details for the solution are detailed in Section IV with some performance figures given in Section V. After the discussion of the solution in Section VI the related work in this are is discussed in Section VII. Finally Section VIII summarizes the paper.

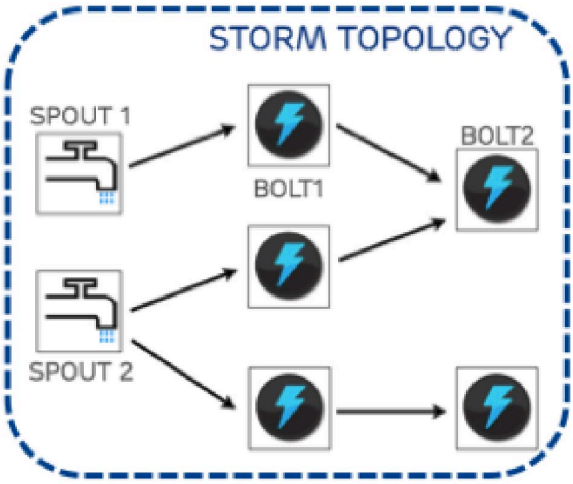

Fig. 1. Storm overview

II. Background and Assumptions

Apache Storm is a distributed real-time computation system. It provides a set of general primitives for real-time computation. Using these primitives a user can create so called topologies to do real-time computation.

Figure 1 shows an example Storm topology. A topology describes the overall computation being done in this topology and is represented as a network of spouts and bolts showing the data path. Spouts provide the input streams for the topology. Bolts process these input streams and can be itself input to other bolts due to the production of an output stream.

Due to its distributed nature Storm also needs something that manages the distributed components. This management task is done by Apache Zookeeper [1] which exposes common services such as naming or configuration management.

Figure 2 shows how a Storm cluster works together with Zookeeper. Storm's Nimbus server is responsible for accepting topologies and distributing the computation over the whole cluster. For the distribution of the computation it uses Zookeeper which communicates with Storm's supervisor nodes to distribute the computation tasks.

In this paper it is assumed that a Storm cluster has access to different kind of data which also includes personal data. Especially due to the personal data an organization wants to make sure that this data is accessed only in a controlled way.

In a standard Storm setup topologies can be submitted by any user even by users which are not fully trusted. This means it cannot be guaranteed that a topology submitted by a user adheres to rules set by the organization. Thus controls need to be in place which users cannot influence and which make it

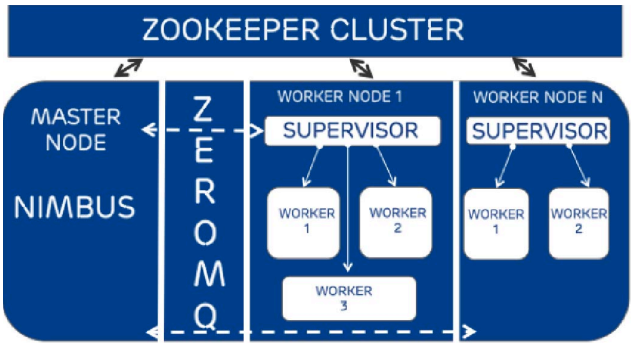

Fig. 2. Storm cluster

possible that users dont have to cooperate. How these controls could be setup is described in the following.

Furthermore, it is assumed that different privacy laws exist in different jurisdictions and different organizations have different business rules. Additionally, end users i.e., those users whose personal data is being handled, have the right to at least opt-out of their personal data being processed. But they also might have the right to be included only if they opt-in. Thus a flexible framework is needed which is able to deal with all these options.

III. Privacy Policy Framework

This paper suggests to use a privacy policy framework which gives a flexible way to accommodate different privacy laws, business rules and users privacy requirements and at the same time enables to control diverse aspects of the computation engine. For the privacy policy framework it is assumed that the organization provides a set of policies which cover the privacy laws in its jurisdiction and their business rules. Customers of this organization can specify their own privacy policies and they will be enforced by the privacy policy framework.

The focus on the proposed privacy policy framework is on three parts namely a security gateway, so called trusted spouts, and a XACML PDP [2] engine. The security gateway ensures that only approved topologies (a decision is made based on privacy laws and business rules) by authorized users are deployed in Storm. The trusted spout is a custom made interface in storm which enforces various policies in events before they are emitted into the bolts of an application topology. The XACML PDP decides if an action is allowed or not using user defined privacy policies or privacy policies defined by the organization.

A. Privacy Policies

Before talking about the other components it is introduced how the XACML PDP engine is used in this framework.

The policy framework contains policies which describe applicable privacy laws and business rules of the organization. Additionally, privacy policies exist which have been specified by an end user. It could be for example specified that the end user's personal data is not to be used for some marketing application.

XACML policies are specified using the Privacy Policy Profile [3]. This allows to specify the "usage" of data. Thus a topology requesting data has to specify for what it wants to use the data (using the AttributeId "urn:oasis:names:tc:xacml:2.0:action:purpose") and the data source has specified for which usages (using the AttributeId "urn:oasis:names:tc:xacml:2.0:resource:purpose") access is allowed. An example for requesting data and specifying its usage is shown in the listing below. This policy excerpt shows that when an application wants to use data with the purpose of using it for "NetworkOptimization" (i.e., this is the "Target") then it is checked against the policy containing this excerpt as target.

```
1  <Target>
2    <AnyOf>
3      <AllOf>
4        <Match MatchId="urn:oasis:names:tc:xacml:1.0↩
               :function:string-equal">
5          <AttributeValue
6            DataType="http://www.w3.org/2001/XMLSchema#↩
                 string">NetworkOptimization
7          </AttributeValue>
8      <AttributeDesignator
9            AttributeId="urn:oasis:names:tc:xacml:1.0↩
                 :action:purpose"
10           Category="urn:oasis:names:tc:xacml:3.0:attribute ↩
                 -category:action"
11           DataType="http://www.w3.org/2001/XMLSchema#↩
                 string" MustBePresent="false"/>
12       </Match>
13     </AllOf>
14   </AnyOf>
15 <\Target>
```

Obligations are also a feature of XACML which are used by the policy framework as they allow to specify that for example data has to be anonymized before it is being processed in the topology. An excerpt of an example policy is shown in the listing below where the obligation is to pseudo-anonymize (see line 14) the IMSI (International Mobile Subscriber Identity) (see line 10). This means the access to the data is allowed under the condition that the IMSI is first pseudo-anonymized before it is read by the application.

```
1  <ObligationExpressions>
2     <ObligationExpression ObligationId=
3     "urn:oasis:names:tc:xacml:example:obligation:
4     anonymization"
5     FulfillOn="Permit">
6      <AttributeAssignmentExpression AttributeId="↩
           urn:oasis:names:tc:xacml:3.0↩
           :example:attribute:anonymize">
7          <AttributeSelector
8              MustBePresent="true"
9              Category="urn:oasis:names:tc:xacml:3.0↩
                   :attribute-category:resource"
10             Path="md:record/IMSI"
11             DataType="http://www.w3.org/2001/XMLSchema#↩
                   string"/>
12         </AttributeAssignmentExpression>
13     <AttributeAssignmentExpression AttributeId="↩
           urn:oasis:names:tc:xacml:3.0↩
           :example:attribute:anonymization-algorithm">
14         <AttributeValue DataType="http://www.w3.org/2001/↩
               XMLSchema#string">pseudo-anonymization</↩
               AttributeValue>
15     </AttributeAssignmentExpression>
16     </ObligationExpression>
17 </ObligationExpressions>
```

Users submitting topologies are assumed to be part of a group like for example the marketing department. This

means that these users can only submit topologies which are producing data for the marketing department.

It is also assumed that the purpose for which a topology is submitted is part of the topology name. The exact naming convention and semantics of the purpose is up to the organization to decide. Thus this information can be used in the PDP for the decision process.

B. Security Gateway

The security gateway provides the perimeter protection functionality thus all requests to Apache Storm like submitting a topology have to go through the security gateway. The security gateway is an extension of Apache Knox [4] which already provides perimeter protection to services like Apache Hadoop and others.

Instead of submitting a topology directly to Apache Storm the user submits the topology to the security gateway. This is the only possible way to submit a topology as the default way to deploy topologies in Storm is disabled. The gateway authenticates the user submitting a topology. It also checks which spouts the submitted topology uses by analyzing the byte code of the submitted topology. The list of the found spouts in the analysis (it is assumed that every data source has its own specific spout implementation), the topology name (or some other meta information which describes for what the topology uses the data i.e., the purpose) and the user name is send as a request to the XACML PDP. The PDP decides if the topology is accepted or not (for example accepting only specific spouts for the requested usage) and sends the decision to the gateway. If the Knox gateway receives the answer "Permit" it submits the topology to Storm. If the answer is "Deny", a response is sent to the client stating that the topology is not allowed to be submitted. Thus the gateway acts as a policy enforcement point (PEP) for the XACML part.

Optionally the security gateway can also maintain meta-data about approved topologies in the past to speed up the decision process. Submitted topologies are then compared to the stored meta-data. The stored meta-data contains:

- User id
- Topology name
- Names of spouts
- Size of each spout
- Average CPU consumption
- Average Memory consumption
- Submission time
- Input and output location

With this meta-data it is detectable if the user tries to fool the system, by for example changing the application logic in spouts, as there will be a difference in the size of the spout file. If the sizes do not match with the values in the past, the gateway server does not accept the topology.

C. Apache Storm

In Apache Storm there are various types of spouts provided to users. One such type is a BaseRichSpout which implements the Ispout interface and has logic to send events as batches to various bolts in the topology. A trusted spouts is a new concept being introduced in this paper. Trusted spouts are special spouts provided and being approved by an organization. There exists a trusted spout for every data source.

All spouts use the SpoutOuputCollector class to identify the destination of every event/batch, append a message-id to each event/batch and push them to the corresponding destinations i.e. bolts. The trusted spouts makes use of a modified SpoutOuputCollector class which acts as a PEP. This PEP calls the XACML PDP and asks for example if "events with certain field values should be filtered and blocked" and receives obligations like "certain fields in the events should be sent in an anonymized way to bolts". If events have to be filtered out then this is done before the events are forwarded to the next bolt. The same holds for obligations like anonymization where fields are anonymized before the event is forwarded to the next bolt.

Users of Storm are not aware of this as the policy enforcement is completely done in the backend classes of storm. Users can continue to create their own spout implementations by extending a trusted spout without any changes in the topology. The only difference is that the backend classes of Storm ensure the policy enforcement in events at run time without the user being able to change it.

D. New Topology Submission Procedure

The sequence diagram in Figure 3 shows how the different components work together to enhance the privacy in the system.

In a first phase (1. - 10.), it is checked if a topology submitted to Apache Storm is allowed to be executed. To do this check the user has to authenticate against the security gateway and submit the topology there (1. - 3.). The gateway extracts from the topology the information which spouts the topology uses (4.) and sends this list together with the user ID and the topology name to the PDP for authorization (5.). The PDP retrieves the policies from Zookeeper and decides if the user is permitted to submit this topology (6. - 8.). If the user is permitted, the gateway forwards the topology to Apache Storm (10.).

In the second phase (11. - 20.), the modified Apache Storm asks the PDP if some end user data needs to be filtered out for this topology (11.). After retrieving the policies from Zookeeper, the PDP decides and sends Storm the information about which user data has to be filtered out (12. - 14.). After this, Storm asks the PDP again but this time if the topology has access to the requested data sources (16.). The PDP again retrieves the appropriate policies from Zookeeper and sends a "Permit" or "Deny". In the case of a "Permit" there can also be an obligation attached that specifies, for example, that personal data has to be anonymized in the spout before being forwarded to the next bolt (19.). When some data has to be anonymized, Storm anonymizes the data before it is forwarded to a bolt (20.).

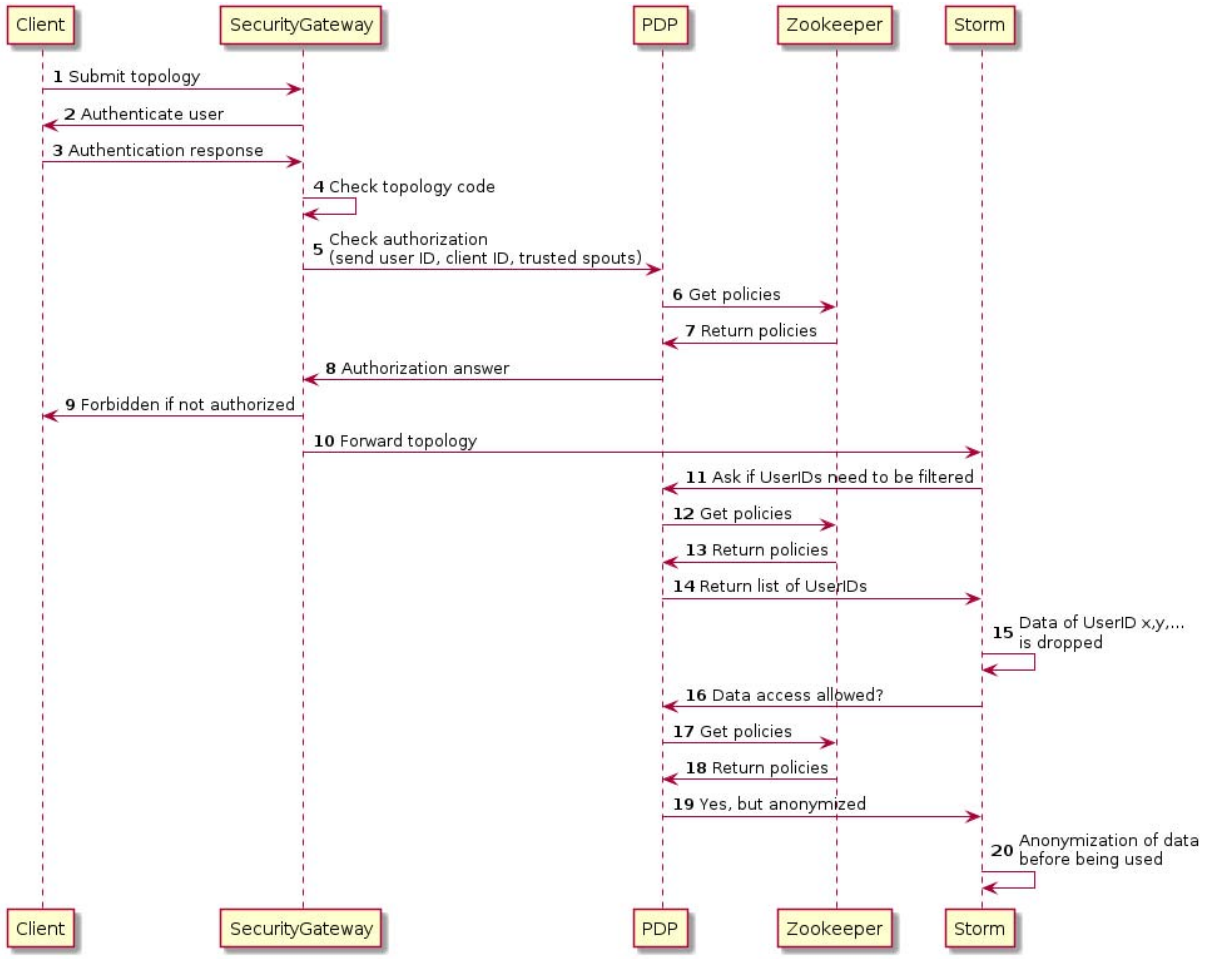

Fig. 3. Sequence Diagram

IV. Implementation

The implementation of the concept explained above is using existing software and extending it with specific functionality needed which is described in more detail in the following.

A. Security Gateway

The security gateway is based on Apache Knox [4] using version 0.5. A so called Knox service has been implemented that forwards the submitted topology to the Nimbus server of Apache Storm in version 0.9.1. Additionally a so called Knox provider was developed which implements the authorization part of the security gateway and acts as a PEP for the XACML part. The provider reads the incoming POST messages for the Storm cluster and extracts the topology from it. Using the Apache BCEL [5] library in version 5.2 the topology is analyzed and the list of spouts being used is extracted. To extract the list of spouts if there are classes implementing a trusted spout and if there are classes implementing "normal" spouts. The Knox provider also uses the existing Ldap implementation of Knox and sends the via Ldap authenticated User ID together with the list of spouts and the topology name to the PDP. If the PDP's answer is "Permit" the message is authorized and forwarded to Storm otherwise it is discarded and the user is notified that this submission is not allowed.

B. XACML PDP

The XACML PDP itself is based on the Balana [6] XACML PDP. Functionality was added to retrieve policies from the Z-nodes of the Zookeeper cluster. In the key of a Z-node is an array for each type of policies with every array separate by a field separator like ":". For example an array for application specific policies and an array for user specific policies would be: [marketing, networkOptimization]:[user1,user2, . . . ,usern]. Thus it is possible to query for application specific policies like marketing and for policies for a range of users which is helpful for blocking data. The communication with the PEPs is implemented using 0mq [7] which is easy to use in a distributed setup.

The policies could also be stored externally in places like HDFS, HBase etc., and the XACML PDP would have to implement then different methods for accessing policies. Nevertheless the current method of storing policies is useful if there are a huge number of policies and a lot of requests for policies to be expected, with very few policies instead this

method is not required. For a few policies only the standard file based module is useful which reads the policies from files in the local file system.

C. Apache Storm

In Apache Storm, there are various types of spouts provided to users. One such type is a BaseRichSpout. The BaseRichSpout implements the Ispout interface and has logic to send events as batches to various bolts in the topology. All spouts, use the SpoutOuputCollector class to identify the destination of every event/batch, append a message-id to each event/batch and push them to the corresponding destinations. To make the spouts trustable, a new interface called ITrustedSpout has been developed.

A new TrustedSpoutOutputCollector class was created. The TrustedSpoutOutputCollector class requests a decision from the XACML PDP related to the events and thus acts as a PEP. For e.g., the PEP could receive an obligation to anonymize "certain fields in the events", or it has to filter and block "events with certain field values". Such instructions are returned by the XACML PDP to the TrustedSpoutOutputCollector which then blocks/anonymizes events before pushing them to their corresponding destinations.

A TrustedBaseRichSpout was created, implementing the ITrustedSpout interface and using the TrustedSpoutOutputCollector class for event routing. Users are not aware of this, as the policy enforcement is completely done in the backend classes of storm. Users can continue to create their own spout implementations by extending TrustedBaseRichSpout, in a similar fashion as they used to do while using the default BaseRichSpout, without any changes in the topology. The only difference is that, by using the TrustedBaseRichSpout, the backend classes of storm (trustedSpoutOutputCollector and ITrustedSpout) ensures policy enforcement in events at run time, without the user being able to change it.

V. Performance

An experimental topology was used to run performance tests to arrive at the performance comparisons between the default Storm version and the customized trusted Storm version. No performance measures were done for the security gateway as the delay created there is not seen as critical for the overall performance.

The topology used for performance tests consisted of one spout, one aggregator bolt and one socket emitter bolt. In the trusted Storm part the spout is replaced by a trusted spout. For the experiments, a topology with a single bolt was used, as the operation was just aggregation. For other complex operations, multiple bolts with each performing a subset of the operation can be created. Though we used only 1 bolt for the experimental purpose, we spawned 5 instances of that bolt, to check the scalability.

The input data consisted of 153 fields depicting various parameters of a phone call done by a subscriber. The topology reads input from a file using a spout. The aggregator bolt aggregates the number of calls made and total minutes spent in outgoing calls. The Socket Emitter bolt is used to emit the output aggregated values to a socket. The topology was run on default Storm (in a single machine) and on trusted Storm (single machine) to observe the difference in performance.

TABLE I. Performance Figures

Feature	Default Storm	Trusted Storm
No. of events processed per second	3800 events (with batch of 10,000 events)	2700 events (with batch of 10,000 events for blocking) 2,200 events (with batch of 10,000 events for anonymization) 2,200 events (with batch of 10,000 events for blocking and anonymization)
Time taken to process 10,000 events.	2.8 seconds	4 seconds (blocking) 4.8 seconds (anonymization) 5 seconds (blocking and anonymization)
Delay in processing	21 seconds for 100,000 events	23 seconds for 100,000 events (blocking) 25 seconds for 100,000 events (anonymization) 25 seconds for 100,000 events (blocking and anonymization)

The machine used for performance tests had 8GB RAM and two 2.4 GHz Quad Core processors i.e., totally 8 cores.

Table I shows the comparison of results with five spout instances and five instances of each bolt for the above given topologies in default Storm and trusted Storm. It is to be noted that all figures in the table have been rounded off to the nearest integer.

Events have to be modified for the anonymization part thus it gives the worst performance. It is interesting to notice that the addition of blocking events to anonymization does not decrease the performance further.

The average initiation time of a topology is increased by 6 seconds by adding the privacy policy framework. But this is acceptable in real time computation systems as in such systems, a topology once deployed never ends and endlessly waits for new data to be processed. The addition in the time taken to initialise a topology is a one time delay which occurs only when the topology is deployed for the first time. Moreover, with the current implementation of the prototype, for every input event/record, the XACML PDP is queried to get the appropriate policies for the user. This can be further optimised by building a cache in the bolt so that redundant queries for policies pertaining to the same query field (in our case IMSI) need not be performed and the XACML PDP can be queried only for points with distinct field (in our case IMSI) values.This can further be optimised by proactively querying for policies applicable to a range of IMSI's and updating the cache with the results. So that, when a new record comes, the XACML PDP need not be queried. Storm can do policy enforcement based on the results in the cache. This will optimise the run time complexity to a significant extent.

The same experiment was done using three virtual machines created in the test machine. Storm was configured to run as a cluster in the three virtual machines. For a topology, 5 spouts and 5 bolts were spawned across the three machines. Default Storm could process 8600 events per second using the three machine cluster. The scale up is not as linear as expected because of the fact that the spouts are now distributed across machines and depends on the network speed and there are more inter process communications across bolts and spouts.

Modified Storm could process 6450 events (with a batch of 10,000 events for blocking), 5700 events per second (with a batch of 10,000 events for anonymization) and 5600 events per second (with a batch of 10,000 events for blocking and anonymization). Since, the overhead added to storm is mostly while initiating a topology, there is no performance differences between default Storm and modified Storm, when scaled up to three machines. Furthermore, the current design does not perform caching of policies. For every batch of input records (in our case 10,000 records) the XACML PDP is queried to get the matching policies. If the policies are cached, unnecessary requests to the XACML PDP are reduced. In such cases, the performance of modified and default storm might be approximately equal. But still, the time taken to initialise a topology with the modified storm will be higher when compared to the default storm.

In case of storm, it is truly efficient to scale only when the input data generated is of higher volume that demands scalability due to performance loss. In case of low input rate, there is no difference in terms of performance if storm is scaled up to more machines.

VI. Discussion

This work assumes that an important feature for privacy is to control the access to data sources and in what form this data can be accessed. It also assumes that an organization can implement trusted spouts for its data sources. Additionally, there are ways for end users to define their privacy policies i.e., to define what is allowed to be done with their personal data and what is not allowed.

Through the concept of the trusted spouts it is ensured that topologies have only access to data they are allowed to access. To correctly enforce this some organizational processes have to be in place. It is assumed that a user submitting topologies is only doing so for a specific area for example marketing or customer care. Thus topologies submitted by this user are not allowed to access data which is not to be used in the user's area. If this user tries to use data sources which are not allowed then the topology is already stopped at the security gateway as some of the spouts in the topology are not in the list of allowed spouts for this user. Additionally if a user tries to submit a topology which has nothing to do with the user's work area than the topology is already stopped at the security gateway.

Through the modification of Apache Storm a user also gets only access to data which is filtered (according to end user wishes) or is pre-processed according to the type of data usage like for example the IP addresses in the data are pseudo-anonymized. The modifications of Apache Storm ensure that the policies are enforced even when a topology creator is overwriting the trusted spout implementation with an own version. A weakness nevertheless remains as Apache Storm is implemented in Java. This means a user creating a topology can overwrite our modifications by re-implementing several Apache Storm classes and thus removing the policy checks.

The security gateway provides perimeter protection and it can be also used to control the traffic coming from the big data cluster. Thus it can be ensured that the data generated in the big data cluster is staying there or is only being received by the one starting the computation. As the gateway also makes sure that the access to the cluster is limited, allowing access only to authorized users, it would be necessary that two users have to cooperate if only one user is allowed to see the output of a program running in the cluster.

On the other hand having one security gateway also means that one single point of failure was introduced. If the gateway stops working then no topologies can be submitted and no user has access to the generated data anymore. Multiple instances of the gateway could be created and a Zookeeper cluster can be configured, to route the request to different security gateway instances (similar to routing to a different web server procedure followed by famous websites like Yahoo) to balance the load and ensure that the gateway doesn't become a single point of failure.

The security gateway itself also introduces some performance penalties as a code checking of topologies is performed and some other information is collected for the policy decision process. Furthermore the gateway has to wait for a decision from the PDP. Nevertheless this affects the system only during topology submission and is thus negligible. Using information about how and for what a topology wants to use the requested data is important for privacy as one aspect is to control how personal data is being used. The policy enforcement is built around this data usage aspect but it is not completely clear yet how this usage information is derived from the system. Currently the usage information is derived from the topology file name but it could also be solved by sending meta-data with the submitted topology which contains usage information. In the end this usage information has to be trusted as it is part of the policy decision. A topology submitting user could be allowed to submit topologies only for a specific purpose thus limiting the possibilities of a user to provide wrong information.

The policies are not guarded by signatures which are validated by the PDP as we assume access to the trusted environment is only available to the security gateway for those users submitting topologies. The PDP, which is inside the trusted environment, is evaluating policies only stored in the trusted environment. A limited number of users, which has full access to the trusted environment, can insert and modify business policies. Policies of customers are also assumed to be inserted in the trusted environment via a special channel where a user submitting topologies does not have access.

We assume that data sources for Apache Storm are not only databases but also streaming data coming directly from nodes in the network. Thus it is not possible to reuse or modify security/privacy features of databases instead it is reasonable to add security/privacy features to Storm.

In Apache Storm, there are numerous types of spouts. All the spout types are inherited from a basic spout template. The basic spout template uses the SpoutOutputCollector class to identify the bolts to which the incoming events are to be forwarded. The SpoutOutputCollector is modified with calls to the policy framework and the logic to enforce policies based on responses from the policy framework. In case new spout types are added to Storm, they will still continue to inherit the basic spout template which in turn uses the SpoutOutputCollector class. Hence, with such new updates, the proposed design

of privacy enforced storm will continue to work without problems. In case, there is a very major design change in storm and the basic spout template and SpoutOutputCollector class are scrapped, then it requires some minimal changes to be done i.e., coding calls to the privacy framework in appropriate places as per the new design of storm. But the changes are minimal as the privacy framework is a stand alone module to Storm and the logic to enforce policies based on results from the policy framework can be reused as such with the only change that they should be placed in the appropriate place as per the new design of Storm.

In the case that the spout code of the topologies is being updated and thus not being recognized as a valid spout in the security gateway anymore there are two ways to deal with it. One way is to update the information about allowed spouts in the security gateway quickly and thus an organization is needed that handles these tasks timely. Another way is to improve the automatic code checking of spouts in the security gateway and decide based on this code check if the spout in the topology can be trusted or not. This last way is future research.

In the current implementation the filtering out of individual user data is a performance bottle neck as the PDP is asked for every user id if it should be filtered out or not. Introducing caching and keeping filter lists locally at the trusted spouts should give some performance boost. It is also partly a business decision on how long such a cache entry is valid so an enterprise could decide that the entry is valid for 24 hours and inform their customers that a change to their individual policy might be effective only after a maximum of 24 hours.

VII. Related Work

We are not aware of much related work in the big data or real-time computation area with respect to privacy. Several projects, like the Intel Rhino project [8] work, on increasing security within big data frameworks like Apache Hadoop. The main work items are authentication infrastructure, encryption of data in rest or transit and similar work. There is no work about conditional access or protection from programs (e.g. MapReduce jobs or Apache Storm topologies) which try to access data they should not be allowed to access.

To our knowledge only one project exists (Storm Yarn [9]) which adds security features like authentication to Storm but relies on Yarn. It does not provide any control over the submitted topologies and how they use data respectively which data they access.

In related work to deal with rules (combined to policies) there exists related work for Storm. The project Clara [10] adds distributed rules processing by adding clara-bolts to a topology. These bolts have a subset of the current working memory of the rules engine. Clara rules target business logic and the system was not designed with security/privacy in mind but shows a way of how distributed rules processing could be done.

Policy languages are an active research field and a lot of different languages exist which also are targeting specifically privacy like [11], [12] or [13]. The latter one is also based on XACML but adding more privacy features like attribute based access control and negotiation of privacy policies between the involved parties. As this work is not designing any policy languages the pragmatic decision was to take a language where software implementations are freely available.

VIII. Summary

This paper introduced some mechanisms to improve privacy for Apache Storm. A security gateway was introduced to prevent the submission of unauthorized topologies such that only authorized users are allowed to submit topologies and those topologies are checked for accessing only allowed data sources. Additionally Apache Storm has been modified in such a way that topologies only can use data which has been treated according to some privacy policies like some personal data might have been filtered out and some data fields might have been anonymized.

The future work looks into how to integrate or apply similar techniques to other components in Apache Hadoop so that conditional access is possible and also programs are checked before submission. The policy framework which stores policies and fetches matching policies based on requests, is a stand alone module. It can be integrated to other existing frameworks, e.g., the job scheduler of Apache Hadoop can be modified, to give calls to the policy framework and enforce policies based on the results from policy framework. This will ensure that map reduce jobs run by the modified scheduler enforces policies for security/privacy in the Hadoop cluster. Likewise the policy framework can be reused with other available open source frameworks as well like Apache Spark. Most components (security gateway and PDP) can be reused as only the trusted spout concept (being unique to Apache Storm) has to be adapted for Apache Spark. Performance is also a topic that needs to be investigated further, here it is probably useful to look at firewall techniques.

References

[1] "Apache Zookeeper," http://zookeeper.apache.org, last accessed November 2014.

[2] E. Rissanen, "eXtensible Access Control Markup Language (XACML) Version 3.0," August 2010.

[3] ——, "XACML v3.0 Privacy Policy Profile," March 2010.

[4] "Apache Knox," http://knox.apache.org/, last accessed November 2014.

[5] "Apache Commons BCEL," http://commons.apache.org/proper/commons-bcel, last accessed November 2014.

[6] "Balana xacml implementation." [Online]. Available: http://xacmlinfo.com/

[7] iMatix Corporation, "Code Connected - ZeroMQ," http://zeromq.org/, last accessed December 2014.

[8] Intel, "Project rhino," https://github.com/intel-hadoop/project-rhino/, last accessed November 2014.

[9] Yahoo, "storm-yarn," https://github.com/yahoo/storm-yarn, last accessed November 2014.

[10] "clara-storm," https://github.com/rbrush/clara-storm, last accessed November 2014.

[11] L. Cranor, B. Dobbs, S. Egelman, G. Hogben, J. Humphrey, M. Langheinrich, M. Marchiori, M. Presler-Marshall, J. Reagle, M. Schunter, D. A. Stampley, and R. Wenning, "The Platform for Privacy Preferences 1.1 (P3P1.1) Specification," November 2006, http://www.w3.org/TR/P3P11/.

[12] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise Privacy Authorization Language (EPAL 1.2)," November 2003.

[13] S. Trabelsi, G. Neven, and S. Ragett, "Report on Design and Implementation," PrimeLife Project, Tech. Rep. D5.3.4, May 2011.