International Journal of Advances in Electronics and Computer Science, ISSN: 2393-2835 Volume-4, Issue-2, Feb.-2017
http://iraj.in

SECURITY OF REAL-TIME BIG DATA ANALYTICS PIPELINE

[1]KHIN ME ME THEIN, [2]THI THI SOE NYUNT, [3]KYAR NYO AYE

University of Computer Studies, Yangon
E-mail: [1]khinmemethein@gmail.com, [2]thithisn@gmail.com, [3]kyarnyoaye@gmail.com

Abstract- In today's world, real-time data or streaming data can be conceived as a continuous and changing sequence of data that continuously arrive at a system to store or process. Big Data is also one of the hottest research topics in big data computing and it requires different approaches: techniques, tools and architecture. Big data security also faces the need to effectively enforce security policies to protect sensitive data. Trying to satisfy this need, we proposed the secure big data pipeline architecture for the scalability and security. Throughout our work, we emphasize about the security of message. We use Apache Kafka and Apache Storm for real time streaming pipeline and also use sticky policies and encryption/decryption algorithm for security.

Keywords- scalable, durable, fault-tolerant, publish-subscribe messaging, aggregation, replication

I. INTRODUCTION

In today's world, real-time information is continuously getting generated by applications (business, social, or any other type), and this information needs easy ways to be reliably and quickly routed to multiple types of receivers. Most of the time, applications that are producing information and applications that are consuming this information are well apart and inaccessible to each other. This, at times, leads to redevelopment of information producers or consumers to provide an integration point between them. Therefore, a mechanism is required for seamless integration of information of producers and consumers to avoid any kind of rewriting of an application at either end [4].

In the present big data era, the very first challenge is to collect the data as it is a huge amount of data and the second challenge is to analyze it. This analysis typically includes following type of data and much more:

• User behavior data
• Application performance tracing
• Activity data in the form of logs
• Event messages

Kafka provides seamless integration between information of producers and consumers without blocking the producers of the information and without letting producers know who the final consumers are. Kafka can be used like a traditional message broker. It has high throughput, built-in partitioning, replication, and fault-tolerance, which makes it a good solution for large scale message processing applications.

Kafka can also be used for high volume website activity tracking. Site activity can be published, and can be processed real-time, or loaded into Hadoop or offline data warehousing systems. Kafka can also be used as a log aggregation solution. Instead of working with log files, logs can be treated a stream of messages.

A. Our Contributions

In this study, an efficient encryption scheme which is named AES – (Advanced Encryption Standard) 256bits encryption and decryption is used. The contributions of our schemes have three aspects.

- Firstly, we propose the big data pipeline architecture using Apache Kafka and Apache Storm framework for real time streaming applications.
- Secondly, we propose the sticky policy for the security of big data pipeline.
- Thirdly, we use visualization tool to analyze the output in user-understandable format.

The work presented in this paper is rough and incomplete, where some important aspects haven't been considered.

B. Related Work

In a modern data architecture that is built on YARN - enabled (Apache Hadoop NextGem MapReduce) Apache Hadoop [1], Kafka works in combination with Apache Storm, Apache Hbase and Apache Spark for real-time analysis and rendering of streaming data. Kafka can message geospatial data from fleet of long-haul trucks or sensor data from heating and cooling equipment in office buildings. Whatever the industry or use case, Kafka brokers massive message streams for low-latency analysis in Enterprise Apache Hadoop. Kafka is fully supported and included in HDP (Hortonworks Data Platform) today. Some of the companies that are using Apache Kafka in their respective use cases are as follow [4]:

• LinkedIn (www.linkedin.com): Apache Kafka is used at LinkedIn for the streaming of activity data and operational metrics. This data powers various products such as LinkedIn news feed and LinkedIn Today in addition to offline analytics systems such as Hadoop.
• DataSift (www.datasift.com)

• Twitter (www.twitter.com)
• Foursquare (www.foursquare.com/):
• Square (www.squareup.com/)

C. Organization

The rest of this paper is organized as follows. We show about Big Data in Section 2. Real-time analytics is described in Section 3.In the section 4, we describe the architecture of Kafka and its design and about the zookeeper which needs to run Kafka. We describe Apache Storm in Section 5. In Section 6, we describe sticky policies. Section 7 proposes a framework of our system and explains one use case for this proposed system. We discuss future work and conclude in Section 8.

II. BIG DATA

Big data means not only a large volume of data but also other features that differentiate it from the concepts of ‘‘massive data’’ and ‘‘very large data’’. In the present big data era, the very first challenge is to collect the data as it is a huge amount of data and the second challenge is to analyze it. This analysis typically includes following types of data and much more:

- User behavior data
- Application performance tracing
- Activity data in the form of logs
- Event messages

Big Data is high volume, high velocity, and high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization. Processing or analyzing the huge amount of data is a challenging task. It requires new infrastructure and a new way of thinking about the way business and IT industry works.

III. REAL-TIME ANALYTICS

Real-time analytics is an iterative process involving multiple tools and systems. Real-time analytics is the use of or the capacity to use, all available enterprise data and resources when they are needed. It consists of dynamic analysis and reporting, based on data entered into a system less than one minute before the actual time of use.

IV. KAFKA ARCHITECTURE AND DESIGN

Kafka is a distributed, partitioned, replicated commit log service. Kafka [3] maintains feeds of messages in categories called topics. We’ll call processes that publish messages to a Kafka topic are producers. And we’ll call processes that subscribe to topics and process the feed of published messages are consumers. Kafka is run as a cluster comprised of one or more servers each of which is called a broker. At a high level, producers send messages over the network to the Kafka cluster which in turn serves them up to consumers like this in Figure 1:

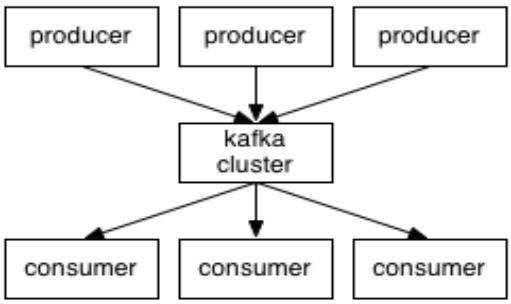

Figure 1: Basic architecture of Kafka

Producers publish messages to Kafka topics, and consumers subscribe to these topics and consume the messages. A server in a Kafka cluster is called a broker. For each topic, the Kafka cluster maintains a partition for scaling, parallelism and fault-tolerance. Each partition is an ordered, immutable sequence of messages that is continually appended to a commit log. The messages in the partitions are each assigned a sequential id number called the offset. Anatomy of a topic is described in Figure 2.

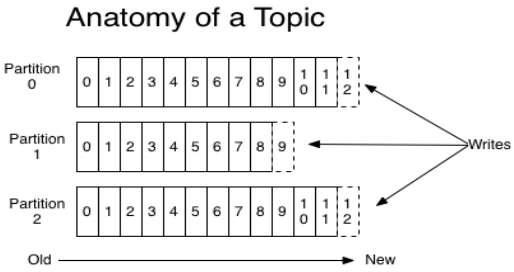

Figure 2: Anatomy of a topic

Kafka can also be used for high volume website activity tracking. Site activity can be published, and can be processed real-time, or loaded into Hadoop or offline data warehousing systems. Kafka can also be used as a log aggregation solution. Instead of working with log files, logs can be treated a stream of messages [5].

Kafka is also an open source, distributed publish-subscribe messaging system, mainly designed with the following characteristics:

• Persistent messaging: To derive the real value from big data, any kind of information loss cannot be afforded. Apache Kafka is designed with O(1) disk structures that provide constant-time performance even with very large volumes of stored messages, which is in order of TB.

• High throughput: Keeping big data in mind, Kafka is designed to work on commodity hardware and to support millions of messages per second.

• Distributed: Apache Kafka explicitly supports messages partitioning over Kafka servers and distributing consumption over a cluster of consumer

machines while maintaining per-partition ordering semantics.

• Multiple client support: Apache Kafka system supports easy integration of clients from different platforms such as Java, .NET, PHP, Ruby, and Python.

• Real time: Messages produced by the producer threads should be immediately visible to consumer threads; this feature is critical to event-based systems such as Complex Event Processing (CEP) systems.

Kafka also supports parallel data loading in the Hadoop systems [4]. Some of the important characteristics that make Kafka such an attractive option include the following table: [8]

Table 1: Characteristics of Kafka

Feature	Description
Scalability	Distributed system scales easily with no downtime
Durability	Persists messages on disk, and provides intra-cluster replication
Reliability	Replicates data, supports multiple subscribers, and automatically balances consumers in case of failure
Performance	High throughput for both publishing and subscribing, with disk structures that provide constant performance even with many terabytes of stored messages

The overall architecture of Kafka is shown in Figure 3. Since Kafka is distributed in nature, a Kafka cluster typically consists of multiple brokers. To balance load, a topic is divided into multiple partitions and each broker stores one or more of these partitions. Multiple producers and consumers can publish and retrieve messages at the same time [2].

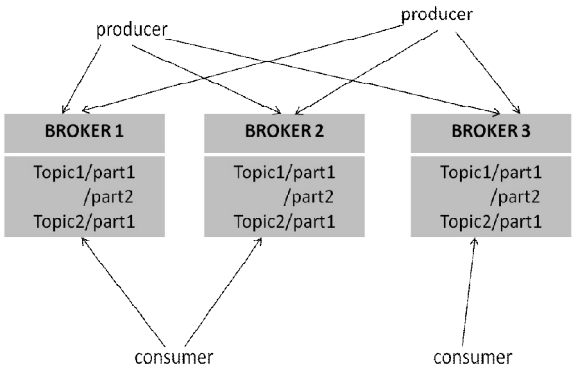

Figure 3: Kafka Architecture

4.1. Producers

Producers publish data to the topics of their choice. The producer is responsible for choosing which message to assign to which partition within the topic. This can be done in a round-robin fashion simply to balance load or it can be done according to some semantic partition function (say based on some key in the message) [3].

4.2. Consumers

Messaging traditionally has two models: queuing and publish-subscribe. In a queue, a pool of consumers may read from a server and each message goes to one of them; in publish-subscribe the message is broadcast to all consumers. Kafka offers a single consumer abstraction that generalizes both of these – the consumer group [3].

4.3. Zookeeper

Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed [6].

Zookeeper is also a high-performance coordination service for distributed applications. It exposes common services - such as naming, configuration management, synchronization, and group services - in a simple interface so you don't have to write them from scratch. You can use it off-the-shelf to implement consensus, group management, leader election, and presence protocols. And you can build on it for your own, specific needs [7].

V. APACHE STORM

Apache Storm is a free and open source distributed real-time computation system. Storm makes it easy to reliably process unbounded streams of data. Storm does for real-time processing what Hadoop did for batch processing. Simple, can be used with any programming language.

Five characteristics make Storm ideal for real-time data processing workloads. Storm is [11]:

- Fast – benchmarked as processing one million 100 byte messages per second per node
- Scalable – with parallel calculations that run across a cluster of machines
- Fault-tolerant – when workers die, Storm will automatically restart them. If a node dies, the worker will be restarted on another node.
- Reliable – Storm guarantees that each unit of data (tuple) will be processed at least once or exactly

once. Messages are only replayed when there are failures.

- Easy to operate – standard configurations are suitable for production on day one. Once deployed, Storm is easy to operate. A storm cluster has three sets of nodes [11]:

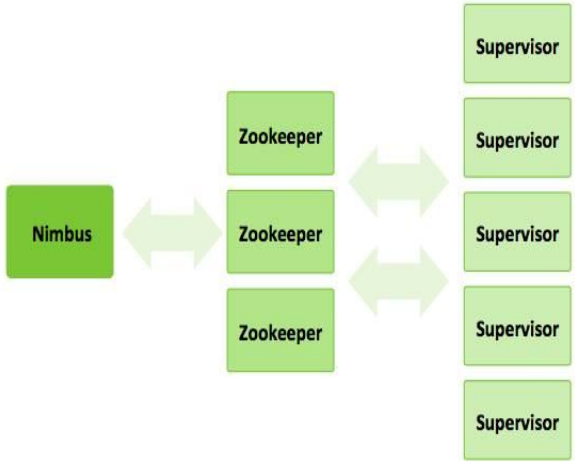

Figure 5: Storm Architecture

VI. STICKY POLICIES

In general, policy refers to guidelines or regulations that encourage user engagement and protect participants' data reports. Policies that attach conditions and constraints to data to define allowed usage and obligations are called sticky policies. Policy is defined according to the National Institute of Standards and Technology as the "aggregate of directives, regulations, rules, and practices that prescribes how an organization manages, protects, and distributes information". In practice, policies are needed to protect sensitive personal data. The concept of sticky policy is to attach privacy policies to data owners' data and drive access control decisions and policy enforcement [12].

6.1. Characteristics of sticky policies

Depending on the degree of a policy's stickiness, the data might be encrypted, with access to the content allowed only upon the satisfaction of these policies. Specifically, the policies govern the use of associated data, and could specify the following [13]:

- proposed use of the data—for example, for research, transaction processing, and so on;
- use of the data only within a given set of platforms with certain security characteristics, a given network, or a subset of the enterprise;
- specific obligations and prohibitions such as allowed third parties, people, or processes;
- blacklists; notification of disclosure; and deletion or minimization of data after a certain time; and
- a list of trusted authorities (TAs) that will provide assurance and accountability in the process of granting access to the protected data, potentially the result of a negotiation process.

VII. PROPOSED SYSTEM ARCHITECTURE

We have designed real-time pipeline architecture for big data security.

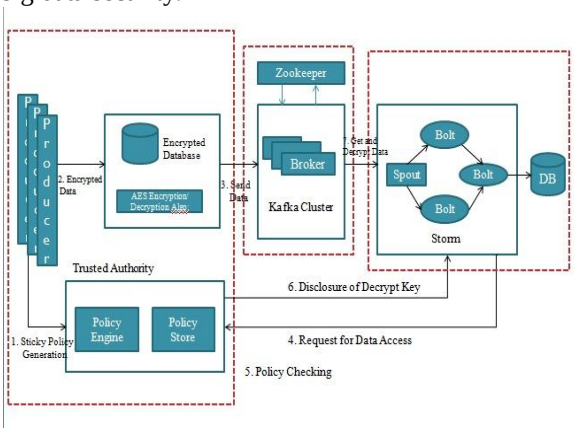

Figure 7: Proposed System Architecture

Figure 7 shows a description of this process, in which the labeled stages are as follows:

1. The data owner describes which data user will be granted access to certain data under specified constraints, and generates a policy rule, then sends the rule to the trusted authority.
2. The data owner encrypts the data with the encryption key, and then stores encrypted data in the database.
3. The encrypted data is sent into the Kafka cluster which is comprised of one or more servers each of which is called a broker.
4. The data consumer sends a request to the trusted authority for data access, which involves passing on the sticky policy.
5. The trusted authority checks policies , potentially including challenges to the data user.
6. If all the policy checks are fulfilled and validated, the trusted authority releases the private decryption key to the data consumer.
7. The data consumer can get the encrypted data from the kafka and decrypt it by using the decryption key.

- The trusted authority includes two components:
  - o The policy engine is the core component of the trusted authority domain. It provides security by keeping track of promises the involved parties make to access data, along with controlling access to such data. The data is encrypted and is only accessible upon the acceptance and satisfaction of specified constraints and duties imposed by the policies.
  - o Policy store deals with policy storage. Policy rules can be safely stored and retrieved.
- The trusted authority's role is expanded to check the integrity and trustworthiness of the data user's

credentials and its environment before releasing the decryption key.

7.1. An Application Use Case in Bank

This section uses a scenario in bank to illustrate an application of the proposed framework. The scenario is described as follows:

Suppose there is a customer who wants to do process in bank and bank staffs for this scenario.
In this scenario, several users are involved and each user has a different role and access control requirements for accessing the process of bank. These users include: (1) customers; (2) receptionists; (3) teller; (4) second in charge and (5) IT team.

Table 1 and 2 show the security policies that regulate the access of different users to the customer's data and some bank routines. Let's walk through the events that would occur in this scenario. As a customer's role, he/she can transfer, withdrawal and deposit the money. As a bank's receptionist, he/she can read the personal data of customer. But he/she can't write the personal data of customer and the bank amount of the customer. And he/she can create the new account for the customer. And teller can accept the money but he/she can't update the customer data. Second in charge of bank has the highest priority and he/she can update the customer data. IT team can maintain the customer data. But they can't update the customer data.

Table 1: Policies for a Bank

Rule ID	Rule	Description
R1	Subject. Role="Customer" and Subject. Credential= * and Object. Type="Transfer Money" and Action={read,*} and goal="confidentiality" → Permit	Customer can transfer money.
R2	Subject. Role="Customer" and Subject. Credential= * and Object. Type="Withdrawal Money" and Action={read,*} and goal="confidentiality" → Permit	Customer can withdrawal money.
R3	Subject. Role="Customer" and Subject. Credential= * and Object. Type="Deposit Money" and Action={read,*} and goal="confidentiality" → Permit	Customer can deposit money.
R4	Subject. Role="Receptionist" and Subject. Credential= * and Object. Type="Customer Personal Data" and Action={read,*} and goal="confidentiality" → Permit	Receptionist can read the personal data of customer.
R5	Subject. Role="Receptionist" and Subject. Credential= * and Object. Type="Customer Personal Data" and Action={read,*} and goal="privacy" → Deny	Receptionist can't write the personal data of customer.
R6	Subject. Role="Receptionist" and Subject. Credential= * and Object. Type="Customer Bank Amount" and Action={read,*} and goal="privacy" → Deny	Receptionist can't write the bank amount of the customer.

Table 2: Policies for a Bank

Rule ID	Rule	Description
R7	Subject. Role="Receptionist" and Subject. Credential= * and Object. Type="Create Account" and Action={write,*} and goal="confidentiality" → Permit	Receptionist can create the new account for the customer.
R8	Subject. Role="Teller" and Subject. Credential= * and Object. Type= "Accept Money" and Action={write,*} and goal="confidentiality" → Permit	Teller can accept the money .
R9	Subject. Role="Teller" and Subject. Credential= * and Object. Type= "Update Customer Data" and Action={write,*} and goal="privacy" → Deny	Teller can't update the customer data.
R10	Subject. Role="Second In Charge" and Subject. Credential= * and Object. Type="Update Customer Data" and Action={write,*} and goal="confidentiality" → Permit	Second in charge can update the customer data.
R11	Subject. Role="IT Team" and Subject. Credential= * and Object. Type="Maintain Customer Data" and Action={write,*} and goal="authorization" → Permit	IT team can maintain the customer data.
R12	Subject. Role="IT Team" and Subject. Credential= * and Object. Type="Update Customer Data" and Action={write,*} and goal="privacy" → Deny	IT team can't update the customer data.

CONCLUSION AND FUTURE WORK

Real-time big data isn't just a process for storing petabytes or exabytes of data in a data warehouse, it's about the ability to make better decisions and take meaningful actions at the right time. Big Data challenges and issues are needed to be handling effectively and in an efficient manner. Stream processing is also required when data has to be processed fast and / or continuously. We use Apache Kafka and Apache Storm to develop secure big data pipeline architecture for real time streaming applications. We propose the sticky policy and encryption and decryption algorithm in addition to the security in the big data pipeline. We will use visualization tool to analyze the output in user-understandable format.

REFERENCES

[1] http://hortonworks.com/hadoop/kafka
[2]Jay Kreps, Neha Narkhede and Jun Rao, "Kafka: a Distributed Messaging System for Log Processing", LinkedIn Corp.
[3] http://kafka.apache.org
[4] Nishant Garg, "Apache Kafka" , PACKT Publishing
[5]http://www.infoq.com/news/2013/12/apache-afka-messaging-system
[6] http://zookeeper.apache.org/
[7] http://zookeeper.apache.org/doc/trunk/
[8] http://hortonworks.com/hadoop/kafka/
[9] https://cwiki.apache.org/
[10] http://www.michael-noll.com
[11] http://hortonworks.com/
[12] Shuyu Li, Tao Zhang, Jerry Gao, Younghee Park . "A Sticky Policy Framework for Big Data Security"
[13] Siani Pearson and Marco Casassa Mont , "Sticky Policies: An Approach for Managing Privacy across Multiple Parties"

★★★