Analytics of Reliability for Real-Time Big Data Pipeline Architecture

Thandar Aung, Aung Htein Maw
University of Information Technology, Yangon, Myanmar
thandaraung@uit.edu.mm,ahmaw@uit.edu.mm

Abstract

Nowadays, many applications need high reliability pipeline architecture to get faster process and reliable data within short time. Kafka has emerged as one of the important components of real-time processing pipelines in combination with Storm. This paper focuses to develop the real-time big data analytics pipeline architecture for reliability. Real-time data pipelines can be implemented in many ways and it will look different for every business. To develop the pipeline architecture, we create real time big data pipeline by using Apache Kafka and Apache Storm. Kafka and Storm naturally complement each other and their powerful cooperation enables real-time streaming analytics for fast-moving big data. Then, the experiment will be conducted how the processing time decreases with the same messages on the different partitions.

Keywords- Messaging, Real-time processing, Apache Kafka, Apache Storm

1. Introduction

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 User behavior data, Application performance tracing, Activity data in the form of logs and Event messages. Processing or analyzing the huge amount of data is a challenging task. It requires a new infrastructure and a new way of thinking about the way business and IT industry works. Today, organizations have a huge amount of data and at the same time, they have the need to derive value from it.Considering the huge volume and the incredible rate at which data is being collected, the need arises for an efficient analytic system which processes this data and provides value in real time.

Real-time processing is a fast and prompt data processing technology that combines data capturing, data processing and data exportation together. Real-time analytics is an iterative process involving multiple tools and systems. It consists of dynamic analysis and reporting, based on data entered into a system less than one minute before the actual time of use [1].In contrast to traditional data analytical systems that collect and periodically process huge –static –volumes of data, streaming analytics systems avoid putting data at rest and process it as it becomes available, thus minimizing the time a single data item spends in the processing pipeline[2].The main purpose of Big Data real-time processing is to realize an entire system that can process such mesh data in a short time[4]. 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 each end.

Real-time usage of these multiple sets of data collected from production systems has become a challenge because of the volume of data collected and processed. Kafka has high throughput, built-in partitioning, replication, and fault-tolerance, which makes it a good solution for large scale message processing applications [8]. In this paper, we propose to develop real time big data analytics pipeline architecture by using Apache Kafka and Apache Storm.

The remainder of this paper is organized as follows: section 2 reviews the related work of this paper. Section 3 presents the proposed system architecture. In Section 4, we describe the architecture of Kafka, the zookeeper which needs to run Kafka. The process of Apache Storm shows in Section 5. Section 6 describes the framework of our system and testing results for this proposed system. Then, Ring Election Algorithm is intended to enhance the pipeline architecture in the future. Section 7 describes conclusion and future work.

2. Related Work

Khin Me Me Thein [1] has proposed to provide the secure big data pipeline architecture for the scalability and security.The author used Sticky policies and AES Algorithm for secure big data pipeline for real time streaming applications.

Steffen FriedWolfram Wingerath, FelixGessert,rich, and Norbert Ritter [2] have also proposed qualitative comparison of the most popular distributed stream

processing systems.The author gives an overview over the state of stream processors for low-latency Big Data analytics and conduct a qualitative comparison of the most popular contenders, namely Storm and its abstraction layer Trident, Samza and Spark Streaming. In their paper, Streaming processing system is high availability, fault-tolerance and horizontal scalability.

Mohit Maske, Dr. Prakash Prasad, International Journal of Advanced [3] intends to ensure the practical and high effiency in simulation system that is established and shown acceptable performance in various expressions using data sheet. It proved that data analysis system for stream and real time processing based on storm can be used in various computing environment.

Wenjie Yang, Xingang Liu and Lan Zhang [4] have also proposed to ensure the practical applicability and high efficiency, to establish and shows acceptable performance in simulation. In their paper, an entire system RabbitMQ, NoSQL and JSP are proposed based on Storm, which is a novel distribution real-time computing system. The paper organized a big data real-time processing system based on Strom and other tools, and according to the simulation experiment, the system can be easily applied in practical situation.

Martin Kleppmann [5] explains the reasoning behind the design of Kafka and Samza, which allow complex applications to be built by composing a small number of simple primitives – replicated logs and stream operators. We draw parallels between the design of Kafka and Samza, batch processing pipelines, database architecture and design philosophy of UNIX.

P Beaulah Soundarabai, ThriveniJ, K R Venugopal, L M Patnaik [6] describes the process of ring Election Algorithm and presents a modified version of ring algorithm. Their paper involves substantial modifications of the existing ring election algorithm and the comparison of message complexity with the original algorithm. Simulation results show that our algorithm minimizes the number of messages being exchanged in electing the coordinator. Each of Election Algorithms gives better performance in terms of time and messages.

Seema Balhara, Kavita Khanna[7] has proposed to maintain coordination between the nodes and leader node have to be selected. Their paper contains the information about the various existing leader election mechanisms which is used for selecting the leader in different problem. The author discusses about several election algorithm in Distributed system.

Jiangyong Cai, Zhengping Jin[12] has proposed a real-time processing scheme for the self-health data from a variety of wearable devices by using storm. Their designs a framework using Apache Storm, distributed framework for handling stream data, and making decisions without any delay. Their framework has improved more efficient than the old method of using regular task with DB cluster.

3. Proposed System Architecture

In this section, we focus on the design and architecture of big data real-time pipeline as our proposed system architecture in Figure 1.

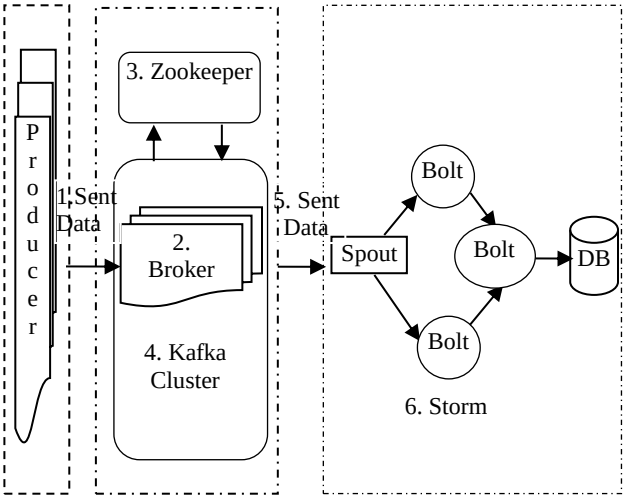

Figure 1. Proposed System Architecture

The processes of proposed system architecture are as follows:

1. In Apache Kafka, Producer send messages to consumers. Brokers can divide messages in many partitions.
2. Each partition is optionally replicated across a configurable number of servers for fault tolerance. Each partition available on either of the servers acts as the leader and has zero or more servers acting as followers.
3. If one of the followers fails, the system can choose follower in-sync replicas (ISR) list. If the leader fails, the system can elect leader randomly in processing.
4. Kafka is a high-performance publisher-subscriber-based messaging system .Kafka spout is available for integrating Storm with Kafka clusters.
5. The Kafka spout is a regular spout implementation that reads the data from a Kafka cluster. Kafka has emerged as one of the important components of real-time processing pipelines in combination with Storm.
6. Kafka can act as a buffer or feeder for messages that need to be processed by Storm. Kafka can also be used as the output sink for results emitted from the Storm topologies. By constructing real time pipeline architecture, two processes can run concurrently. When a process is running in storm, another process can run in Kafka. So, real time message processes faster and faster. It can process high performance in message parsing system.

4. Apache Kafka architecture

Kafka[8] is 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 which provides a real-time publish-subscribe solution for overcoming the challenges of consuming the real-time and batch data volumes that may grow in order of magnitude to be larger than the real data.

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

Apache Kafka is a real time, fault tolerant, scalable messaging system for moving data in real time. Kafka 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. Producers send messages over the network to the Kafka cluster which in turn serves them up to consumers. A producer publishes messages to a Kafka topic. Kafka topic is also considered as a message category or feed name to which messages are published. Kafka topics are created on a Kafka broker acting as a Kafka server. Processes that subscribe to topics and process the feed of published messages are called consumers. Brokers and consumers use Zookeeper to get the state information and to track message offsets, respectively. In figure 2, single node-multiple broker architecture is shown with a topic having four partitions. There are five components of the Kafka cluster: Zookeeper, Broker, Topic, Producer, and Consumer.

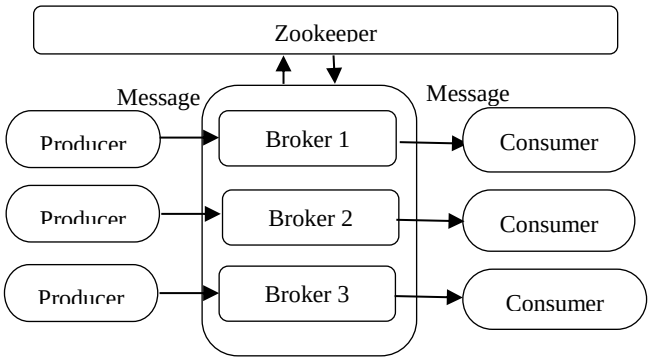

Figure 2. A single node-multiple broker architecture

All the message partitions are assigned a unique sequential number called the offset, which is used to identify each message within the partition. Each partition is optionally replicated across a configurable number of servers for fault tolerance. Each partition available on either of the servers acts as the leader and has zero or more servers acting as followers. Here the leader is responsible for handling all read and write requests for the partition while the followers asynchronously replicate data from the leader. Kafka dynamically maintains a set of in-sync replicas (ISR) that is caught-up to the leader and always persist the latest ISR set to Zookeeper. In a Kafka cluster, each server plays a dual role; it acts as a leader for some of its partitions and also a follower for other partitions. If any of the follower in-sync replicas fail, the leader drops the failed follower from its ISR list. After the configured timeout period and writes will continue on the remaining replicas in ISRs. Whenever the failed follower comes back, it truncates its log to the last checkpoint and then starts to catch up with all messages from the leader, starting from the checkpoint. As soon as the follower becomes fully synced with the leader,the leader adds it back to the current ISR list.

If the leader fails, the process of choosing the new lead replica involves all the followers' ISRs registering themselves with Zookeeper. The very first registered replica becomes the new lead replica and its log end offset (LEO) becomes the offset of the last committed. The rest of the registered replicas become the followers of the newly elected leader. The system occurs the problem in

leader Election because Kafka dynamically maintains a set of in-sync replicas. So the replica is not reliable in processing. Figure 3 explain replication in Kafka:

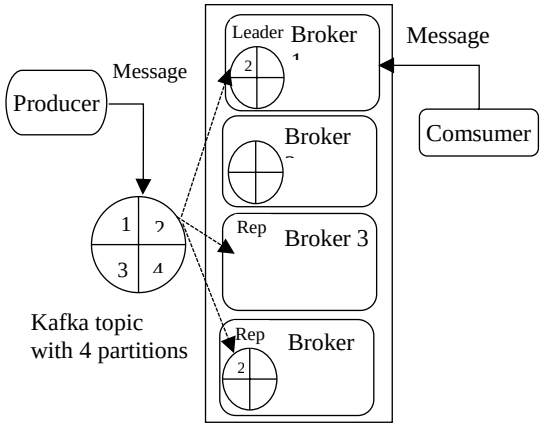

Figure 3.Replication in Kafka

4.1. Zookeeper

Zookeeper [10] is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Zookeeper is also a high-performance coordination service for distributed applications. 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. When it works correctly, different implementations of these services lead to management complexity when the applications are deployed. The service itself is distributed and highly reliable.

Kafka uses Zookeeper for the following tasks: Detecting the addition and the removal of brokers and consumers. Triggering a rebalance process in each consumer when the above events happen, and Maintaining the consumption relationship and keeping track of the consumed offset of each partition. Specifically, when each broker or consumer starts up, it stores its information in a broker or consumer registry in Zookeeper. The broker registry contains the broker's host name and port, and the set of topics and the partitions stored on it.

5. Apache Storm

Storm [9] is also an open source, distributed, reliable, and fault-tolerant system for processing streams of large volumes of data in real-time. It supports many use cases, such as real-time analytics, online machine learning, continuous computation, and the Extract Transformation Load (ETL) paradigm. Storm can be used for the following use cases:

Stream processing: Storm is used to process a stream of data and update a variety of databases in real time. This processing occurs in real time and the processing speed needs to match the input data speed.

Continuous computation: Storm can do continuous computation on data streams and stream the results into clients in real time. This might require processing each message as it comes or creates small batches over a little time. An example of continuous computation is streaming trending topics on Twitter into browsers.

Distributed RPC: Storm can parallelize an intense query so that you can compute it in real time.

Real-time analytics: Storm can analyze and respond to data that comes from different data sources as they happen in real time. A Storm cluster follows a master-slave model where the master and slave processes are coordinated through Zookeeper. The Storm Cluster is made up of a main node and several working nodes [4].

5.1. Nimbus

The Nimbus node is the master in a Storm cluster. A daemon process called "Nimbus" is running on main node, in order to allocate codes, arrange tasks and detect errors.

5.2. Supervisor

Supervisor nodes are the worker nodes in a Storm cluster. Each working node has a daemon process called "Supervisor" to monitor, start and stop working process. The coordination work between Nimbus and Supervisor is handled by "Zookeeper" as shown in Fig 4. Zookeeper is the subproject of Hadoop, and it aims at coordinate works in large-scale distribution system. The Storm Cluster is similar with Hadoop, where Nimbus corresponds to Job Tracker, and Supervisors correspond to Task Trackers.

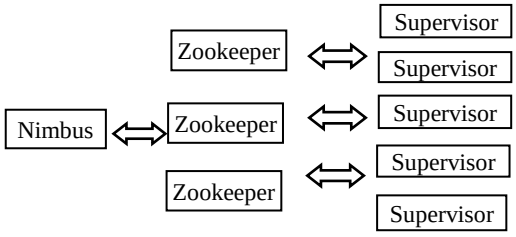

Figure 4. Storm Cluster's Architecture

In Storm terminology, [9] a topology is an abstraction that defines the graph of the computation. A topology can be represented by a direct acyclic graph, where each node does some kind of processing and forwards it to the next node(s) in the flow. The followings are the components of a Storm topology:

Stream: A stream is an unbounded sequence of tuples that can be processed in parallel by Storm. Each stream can be processed by a single or multiple types of bolts.
Spout: A spout is the source of tuples in a Storm topology. Spout is the input stream source which can read from external data source. [12] For example, by reading from a log file or listening for new messages in a queue and publishing them-emitting, in Storm terminology-into streams.
Bolt: The spout passes the data to a component called bolt. Bolts [12] are processor units which can process any number of streams and produce output streams. A bolt is responsible for transforming a stream. Each bolt in the topology should be doing a simple transformation of the tuples, and many such bolts can coordinate with each other to exhibit a complex transformation. There is an example of one topology in Figure 5.

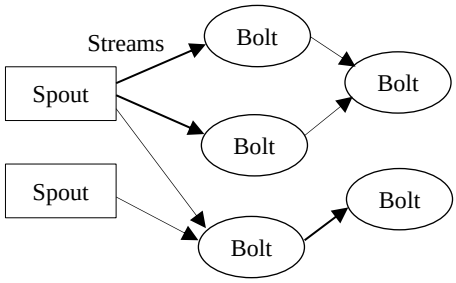

Figure 5. Storm Topology

6. Experimental Setup and Results

The experimental set up is performed by using two open source frameworks Apache Kafka 0.8.1.1 and Apache Storm 0.9.3 as the main pipeline architecture. JAVA/jre 1.4 is running on underlying pipeline architecture. The Apache Marven 3.11 is used as in Kafka-Storm integration.

The processes of overall pipeline architecture are as follows:

1. Start zookeeper server for processing.
2. Start Kafka local server to define *broker_id, por*t and *log dir*.
3. Create topic to show a successful creation message.
4. Producer publishes them as a message to the Kafka cluster.
5. The consumer consumes messages.
6. Get some message in our Apache Kafka cluster.
7. Execute by using command to verify whether topic created.
8. Recall producer and write messages again.
9. Get some messages and run Kafka-Storm integration pipeline.

Table 2 shows testing data in pipeline architecture which is one broker and five partitions using text messages. The purpose of our experiment is to show the comparison of the processing time on the same messages with different partitions. We tested four different numbers of messages; they were 18, 22, 24 and 29 messages with five partitions. It shows the faster processing time on various messages and partitions.

Table 2. Testing Data in pipeline

	Number of messages with processing time			
No. of partitions	**18 Msg**	**22 Msg**	**24 Msg**	**29 Msg**
1	**1.7 sec**	**1.9 sec**	**2 sec**	**2.4 sec**
2	**0.9 sec**	**1.1 sec**	**1.4 sec**	**1.5 sec**
3	**0.6 sec**	**0.7 sec**	**0.8 sec**	**0.9 sec**
4	**0.4 sec**	**0.5 sec**	**0.6 sec**	**0.7 sec**
5	**0.2 sec**	**0.3 sec**	**0.4 sec**	**0.5 sec**

Msg=Messages
Sec=seconds

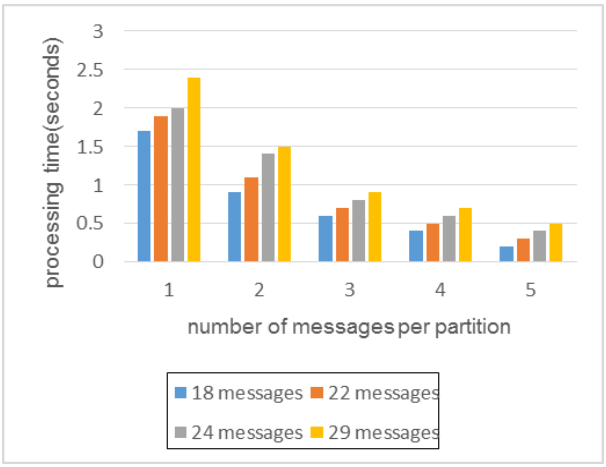

Figure 6.Testing Result of different numbers of messages per partition in pipeline Architecture

Figure 6 shows the process of pipeline by testing on various partitions. According to the experiment, processing time decreases with more partitions are used and the comparison of number of partitions based on various amount of messages. So, this pipeline architecture effects on parallel processing in real time.

6.1 Ring Election Algorithm

In the future, we intend to propose ring election Algorithm in replication process. In Kafka framework, we face any problem in leader Election in processing. In processing, if one of the followers fails, the system can choose follower in-sync replicas (ISR) list. If the leader

fails, the system can elect leader by replacing ring election Algorithm in processing. By using Ring Election Algorithm, we can get more reliable data in Kafka-storm pipeline architecture.

The goal of Ring Election Algorithm is to choose and declare one and only process as the leader even if all processes participate in the election. And at the end of the election, all the processes should agree upon the new leader process with the largest process identifier without any confusion. Ring Election Algorithm can elect new leader without wasting of time and number of messages which are exchanged. Depending on a network topology, many algorithms have been presented for electing leader in distributed systems. The Ring Election Algorithm is based on the ring topology with the processes ordered logically and each process knows its successor in a unidirectional way, either clockwise or anticlockwise. The process of Ring Election Algorithm [10] describes in Figures 7.

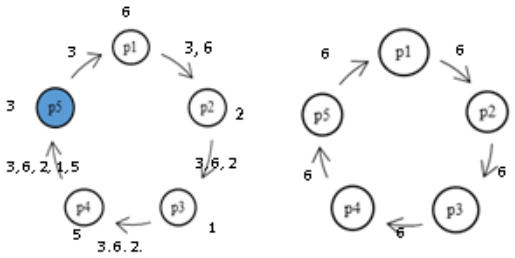

Figure 7. Processes of Ring Election

Step 1.A process is a leader as it has highest id number. When a leader process fails, it starts leader election. It sends message with its id to next node in the ring. The next process passes the message on adding its own id to the message again and again.

Step 2.When starting process receives the message back, it knows the message has gone around the ring, as its own id is in the list. Picking the highest id in the list, it starts the coordinator message as the leader around the ring.

7. Conclusion

In this paper we have implemented a real time framework using Apache Kafka and Apache strom.This pipeline has the reliability to deliver the streaming data. We use Apache Kafka and Apache Storm to develop high performance big data pipeline architecture for real time streaming applications. Using the proposed pipeline architecture, the completion time decrease although a number of messages increase. So, this factor is reliable for the proposed pipeline architecture. We emphasize to be reliable message in real time big data pipeline architecture.

As future direction, we intend to propose Ring election Algorithm in replication process. By using Ring Election Algorithm, we can get more reliable data in Kafka-storm pipeline architecture.

8. References

[1] Khin Me Me Thein, "Security of Real-time Big Data Analytics Pipeline", International Journal of Advances in Electronics and Computer Sciences, Feb, 2017.

[2] Wolfram Wingerath*, FelixGessert, Steffen Friedrich, and Norbert Ritter,"Real-time stream processing for big data", May 2016.

[3] Mohit Maske, Dr. Prakash Prasad, "A Real Time Processing and Streaming of Wireless Network Data using Storm ", International Journal of Advanced Research in Computer Science and Software Engineering, January 2015.

[4] Wenjie Yang, Xingang Liu and Lan Zhang, "Big Data Real-time Processing Based on Storm", 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013.

[5] Martin Kleppmann, "Kafka, Semza and the Unix Philosophy of Distributed Data" Bulletin of the IEEE computer Society Technical Committee on Data Engineering.

[6] P Beulah Soundarabai, ThriveniJ, KR Venugopal, L M Patnaik, "An Improved Leader Election Algorithm for Distributed System", International Journal of Next-Generation Networks (IJNGN), March2013.

[7]Seema Balhara, Kavita Khanna,"Leader Election Algorithms in Distributed System", International journal of Computer Science and Mobile Computing, June 2014.

[8] Nishant Garg, "Apache Kafka", PACKT Publishing UK, 2015.

[9] http://zookeeper.apache.org/.

[10]Tanenbaum Andrew, Tanenbaum-Distributed operating system, Wikipedia, p-100, 1994

[11] Shay Kutten, Shlomo Moran, Leader election, Wikipedia, 24 August 2017

[12] Jiangyong Cai, Zhengping Jin, "Real-time Calculating Over Self-Health Data Using Storm", 4th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering (ICMMCCE 2015).

[13] Ankit Jain, Anand Nalya,"Learning Storm" PACKT Publishing UK, 2015.