High Performance Kafka Powered Scalable Real Time Rule Engine Model for Event Stream Processing

P Murali Krishna, Pallav Kumar Baruah

Abstract— Rule based systems execute actions based on the predefined rules. These systems are widely used in business, government and organizations. Traditional rule based systems perform poorly in big data applications such as IoT and others. In order to meet this challenge, we exploit the inherent nature of the problem to come up with a distributed processing system. We propose a scalable real time rule based engine model to logically distribute the events/data streams generated and apply inference on each logically separated event stream.

Index Terms— Apache Kafka, Big data, Drools, IOT, Kafka Streams, Rule Engine, Stream Processing.

Æ

1 Introduction

Rule engines (or Production system) provides the good way of representing knowledge and reasoning. Rule engine consists of three main components Knowledge base, Working memory and Inference engine. Inference engine is the brain of the Rule engine, it uses some pattern matcher and repeatedly tries to find the rule base that could match with the facts and execute the desired actions. Rule engines are computationally expensive and slow. When the size of the problem continues to grow it will perform poorly.

In the context of IOT, embedded systems like patient health monitoring devices, traffic control devices, temperature control devices etc, produce a large amount of data. These data should be processed in real time and the processing system should be able to handle the large data generated. Traditional Rule based systems cannot process the large data on the single system because of memory and processing power limitations. In order to address this issue, we have proposed a Scalable Real Time Rule Engine Model.

In our model we exploit the inherent data parallel nature of the problem. In our approach we scale up rule engine performance by distributing incoming data streams using Apache Kafka and applying inference algorithms independently on data stream. But this leaves the constraint in cases like parcel tracking, patient health monitoring, traffic flow monitoring etc,. where, related data needs to be processed by the same rule engine to maintain its state and avoid inconsistency. In the case of patient health monitoring system, data about a specific person should reach a fixed rule engine in order to infer knowledge about the person. To tackle this problem, we have chosen key based approach. Our approach requires that the data generated should always have the uniqueness associated with it in the form of key. But, this requirement is inherently met in use cases like those mentioned above. For example, in the context of patient health monitoring system we have the Patient-ID as the key.

Our idea is to build a general model using the big data tool Apache Kafka, in integration with Drools rule engine [1] to handle the above mentioned requirements. We have chosen drools to be the underlying rule engine because of its wide community support and usage in many areas like network fault management [2], intrusion detection system [3] and other.

2 Related Work

Rete by Forgy [4] is a pattern matching algorithm and is one of the main contributions. Inspired by this there are other improvements and modifications like Treat [5], Rete/UL [6], Rete [7],[8]. But all these improvements have not addressed the problem of processing a large number of rules and facts on

P Murali Krishna is currently pursuing masters degree in Computer Science in Department of Mathematics and Computer Science in Sri Sathya Sai Institute of Higher Learning, Puttaparthi, India.
PH: +91 9490844978, Email: muralikrishna.mdh@gmail.com.

Pallav Kumar Baruah, Head of Department, Department of Mathematics and Computer Science in Sri Sathya Sai Institute of Higher Learning, Puttaparthi, India.

PH: +91 9440699887, Email: pkbaruah@sssihl.edu.in.

a single system because of memory and processing power limitations. These methods keep rules and facts in memory to process them. With the increase in data size and the number of rules, the performance of the system deteriorates. So these methods are not directly applicable for big data environments. To address this issue, author has proposed[9] a message passing model to deal with big data.

In this paper, we propose a scalable real time rule engine model to logically distribute the events/data streams generated and to apply inference on each logically separated event stream. We have used apache kafka for logically distributing the data and apache drools for inference on each processing node. Finally, we implement our model with a bank transaction case study and show its scalability and performance through experimentation.

This paper is organized as follows. Section 3 provides the background information of Apache Kafka, Rule Engine and Kafka Streams used in our model. Section 4 describes the architecture and the components used in our model. Section 5 describes the details of implementation. Section 6 describes the detailed implementation of our model with a case-study. Section 7 is about the various experiments conducted to show the scalability of our model. We conclude by mentioning possible future works in Section 8.

3 BACKGROUND

3.1 Apache Kafka

Apache Kafka[10][11] is a publish-subscribe messaging system. It is horizontally scalable and fault-tolerant. Kafka cluster consists of multiple brokers(servers). Each broker is identified with an id. Each broker contains certain topic partitions. Fig. 1, shows the kafka with 3 brokers having topic-1 with 3 partitions, topic-2 with 2 partitions and topic-3 with 1 partition.

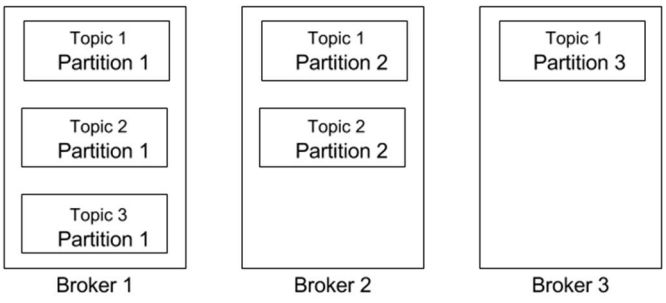

Fig. 1. Brokers configured with different Topics and Partitions.

Producers are the processes which publish data to the kafka topics. A consumer of topic pulls the data from kafka topic. Producer decides in which topic partition data should be stored, it can be based on round robin fashion simply to balance the load among partitions or based on semantic partition function (based on some key in the record). A Topic is a category/feed name to which messages are published. All the messages in the kafka are organized as a topics. If any message published to the kafka, it should be published to specific topic and if any messages to be read, it should be read from the specific topic. Kafka topics are divided into multiple partitions. This provides multiple consumers to read data from a topic partitions parallelly. Each message in a partition is assigned and identified by a unique offset. Consumers read messages from a specific offset or they read from any valid offset point they chose. Consumer group contains set of consumer instances and consumer group subscribes to a specific topic to read data from. Consumers read data in a consumer group. Each consumer within a group read exclusive partitions.

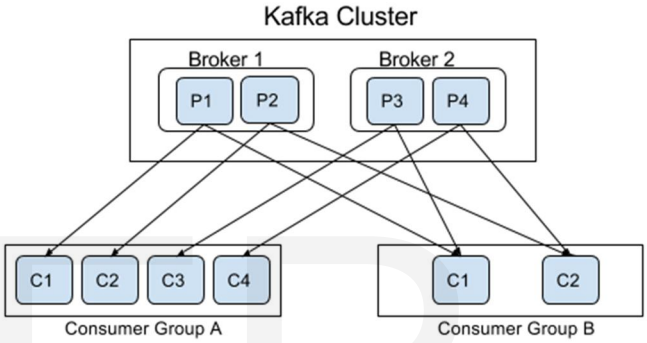

Fig. 2 shows the mapping of partitions to consumer group instances.

3.2 Kafka Streams

Kafka Streams [12] client library is built on top of kafka producer and consumer clients. It uses stream partitions and stream tasks as a logical unit of parallelism. Stream partitions are mapped to kafka topic partitions. Data record in the stream is a message in kafka and key of the record determines the partitioning of data in both kafka and kafka streams. Kafka task is build using kafka consumer API. Same partitions to consumers mapping applies to kafka task instances in the kafka streams too. Each kafka stream runs with an application ID that will act as the consumer group name in kafka.

3.3 Rule Engine

A rule based engine uses the predefined rules to infer some knowledge from the facts. Rule based engines consists of three main components:

1. Knowledge Base: contains set of rules and facts.
2. Working Memory: used to store the facts of current system state.
3. Inference Engine: applies logical rules on the facts to infer new facts.

3.3.1 Knowledge Base

Knowledge Base may be considered as a representation of

logical thinking of an expert in the form of rules. The rules are expressed as: *IF (Condition) THEN (action)* format. It can also be expressed as *LHS(rule) and RHS(action)*, where left hand side contains the condition or rule and the right hand side contains the action to be taken based on the condition/rules satisfied.

3.3.2 Working Memory

The working memory stores the facts of current system state, and is referred to as facts base. The facts which are inputs to the rule engine and the facts derived from the rule engine are kept in the working memory for the further inference and to maintain the state of the rule engine. Initially working memory is initialized with the facts of the rule base.

3.3.3 Inference Engine

A typical inference engine contains the pattern matcher, conflict resolver and execution engine. Pattern matcher matches the rule base with the working memory(facts). When the rule is fully matched, a rule match is created with rule and matched facts, and placed onto the Agenda. Agenda controls the order of execution of the matches using conflict resolution strategy.

4 Architecture

In our Scalable Real Time Rule Engine (SRTRE) Model, we propose an architecture to scale up rule engine performance by distributing incoming data streams and applying inference algorithms independently on each data stream. This goal demands a requirement that the data streams should be logically separated so that all data required for inference is available at a single node. To solve this problem we require a system which distributes the data logically (based on a record key) to the processing nodes. Apache kafka provides the above mentioned functionality. So we have chosen apache kafka to act as the messaging platform to distribute the data/records to the processing nodes in our model. Fig. 3 shows the overall architecture of our SRTRE Model.

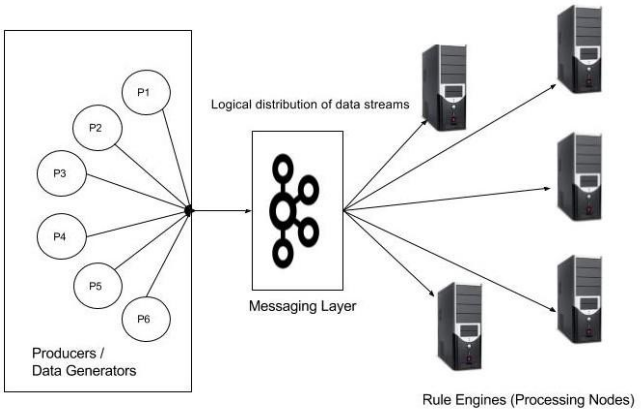

Fig. 3. High Level Architecture of SRTRE Model

SRTRE Model consists of three modules.

1. Data Generators generate events/facts to kafka cluster.
2. Messaging layer logically distributes the data to the processing nodes for inference. Each processing node has a rule engine running in it.
3. Rule Engine (Processing Node) receives its share of stream of facts and matches them against rules and takes the desired action.

5 Implementation

SRTRE Model uses Apache Kafka as a messaging platform to distribute the data/facts among the processing nodes and Apache Drools as a rule engine to process the facts and rules generated by the clients on each processing node. In this model, rules are written by the clients and facts are events generated by the producers.

5.1 Data Generator

Data generator module generate a stream of events which are referred to as facts in the rule engine. These events can be anything like sensors sending a continuous stream of current temperature in the area, patient health status, transactions made by the bank customers etc. We have used kafka producer to publish the generated events to the kafka topic. This module contains one or more producers, each generating and writing a stream of event data generated to a topic. Each event record generated is in the form:

Key, Value1, Value2, Value3, ...

Each producer connects to any of the brokers in the kafka cluster and will gain access to all the brokers in the kafka cluster. Producer is responsible for choosing which topic partition the fact record is to be assigned to. It is done according to some semantic partition function (based on the record key). This makes sure that each record with a particular key reaches the same topic partition. So all the records with a particular key will reach a fixed partition.

5.2 Message Broker

We have used Kafka Cluster as messaging system. It consists of one or more brokers. Any fact record published to the kafka is written to the topic partition and replicated for fault tolerance in the kafka cluster. Kafka cluster uses Zookeeper[13] for managing and coordinating kafka brokers. Producers and Consumer are notified by the Zookeeper about the presence of new Kafka broker or failure of kafka broker in the kafka cluster. It maps topic partitions to the consumer instances in the consumer group. Partitions are exclusively mapped to the consumer instances in the consumer group, so that records with particular key always reach the fixed consumer instance in the consumer group.

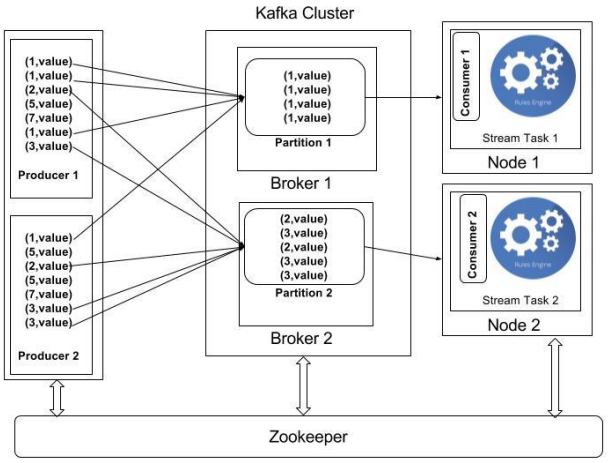

Fig. 4. Detailed Architecture of SRTRE Model

5.3 Rule Engine (Processing Node)

We have configured processing nodes with kafka stream instance for receiving stream of fact records along with drools rule engine. Drools is an object oriented rule engine written in java. It is a forward and backward chaining based inferencing rule engine. Every fact in drools is an object. Fact record received by the consumer is parsed, a fact object is instantiated with the parsed record and inserted into the drools. Inference engine is brain of any rule engine. It matches the facts against the rules to infer conclusion which result in action. Fig. 4 shows the detailed implementation of SRTRE Model.

6 Illustrating Example

This section illustrates an implementation of SRTRE Model with a variation of banking benchmark. The Banking benchmark [14] contains three main classes AccountPeriod, CashFlow and Account. Each AccountPeriod object contains the starting and ending dates of accounting period, Account object contains the information of one customer, CashFlow object contains type of transaction CREDIT or DEBIT and amount of transaction. This benchmark contains a data generator and set of rules to calculate each customer's balance. Data generator produces large amount of data/facts that are published to the messaging layer (Kafka cluster) in SRTRE Model. The two rules used in our example are as follows:

```
rule "Test 01 - Credit CashFlow"
    salience 100
    when
        AccountingPeriod( $start : start, $end : end )
        not AccountingPeriod( start < $start)
        $cashFlow : CashFlow( $account : account, date <= $end && >= $start,
        $amount : amount, type==CashFlow.CREDIT )
    then
        $account.setBalance($account.getBalance() + $amount);
        retract($cashFlow);
end

rule "Test 01 - Debit CashFlow"
    salience 100
    when
        AccountingPeriod( $start : start, $end : end )
        not AccountingPeriod( start < $start)
        $cashFlow : CashFlow( $account : account, date <= $end && >= $start,
         $amount : amount, type==CashFlow.DEBIT )
    then
        $account.setBalance($account.getBalance() - $amount);
        retract($cashFlow);
end
```

KStream object is instantiated with key and value as string data type. mapValue() function calls parse_insertData() on record by record basis to parse and insert data into the kieSession created in drools and to match the facts against the rules and fires the action.

7 Results

We have evaluated our model with banking benchmark with two rules. In our first experiment, we have setup Kafka cluster with two nodes; four partitions for the topic with replication factor of two for each partition; two Kafka streaming instances running on these nodes with two partitions being mapped to one Kafka streaming instance. Each node is configured with an Intel Core i5-4670 CPU @ 3.40GHz, 4 core and 16GB RAM. Fig. 5 shows the performance result with the above configuration.

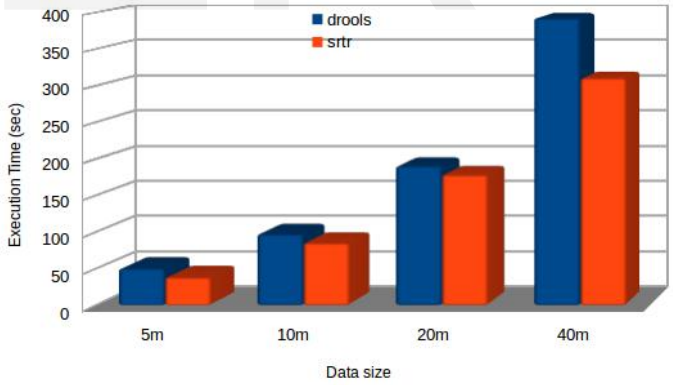

Fig. 5. Performance evaluation of drools and SRTRE Model using two nodes

We have evaluated our model with various configurations for scalability with the data range from 5m - 40m records for different cluster configurations of 2 nodes, 4 nodes and 6 nodes. For a cluster setup with 2 nodes, we configured 2 brokers, 2 partitions; 4 nodes, we configured 2 brokers, 4 partitions and 3 brokers, 6 partitions for 6 processing nodes. In the above configurations, all the nodes are functional and were used in processing. In Fig. 6 we have evaluated our

model across varying number (afore mentioned) of processing nodes and partitions. As seen from the graph in Fig. 6, our model takes the least amount of time despite increasing data sizes thus performing better than Drools.

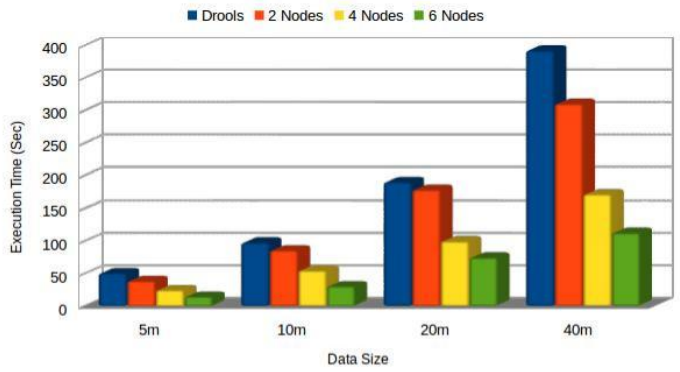

Fig. 6. Performance evaluation of drools and SRTRE Model with varying number of nodes (with console output).

The performance of system measured in Fig. 6 included console outputs. On running the system without console outputs, the performance improvement achieved is summarised by Fig. 7. In the later case we have also measured the system performance with 60M inputs.

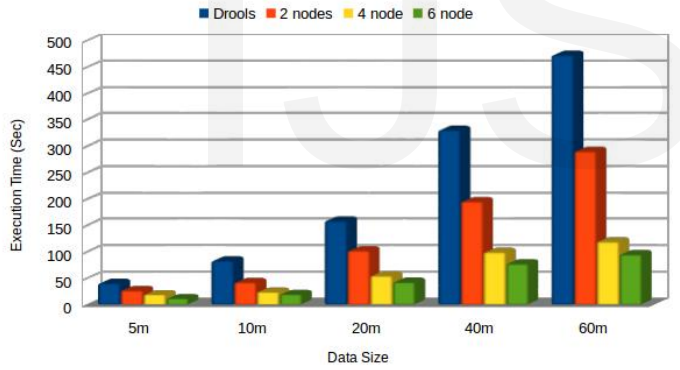

Fig. 7. Performance evaluation of drools and SRTRE Model with varying number of nodes (without console output).

Furthermore, we increased our data set ranging from 40m, 60m and 80m to emphasize very good scalability of our model. For this, 2 different cluster configurations were used to depict the scalability First, 6 brokers, 10 partitions for 5 nodes configuration where each node consists of two processing instances. Second, 6 brokers, 12 partitions for 6 nodes configuration where each node of two processing node instances. Fig. 8 shows that for large data sets (>40mn) our system scales well. In this experiment, console output was enabled.

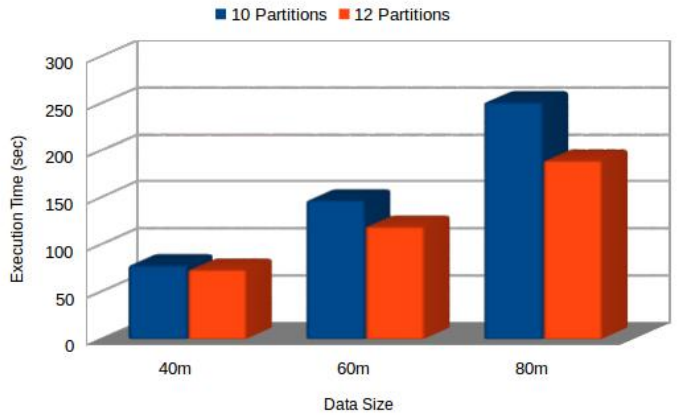

Fig. 8. Performance evaluation of SRTRE Model with larger data set.

In order to identify the main factor responsible for scalability we evaluated our model by fixing the number of brokers and varying no. of partitions and processing nodes. Each partition is mapped to a single node. The partitions were varied from 2 partitions - 12 partitions with 2 fixed brokers cluster configuration with data size of 60M. Fig. 9 shows that as we increase the number of partitions, the time taken by our model decreases rapidly. This behaviour is observed due to increased scope of parallelism as we increase the partitions and the processing nodes. This experiment was carried out without enabling console output.

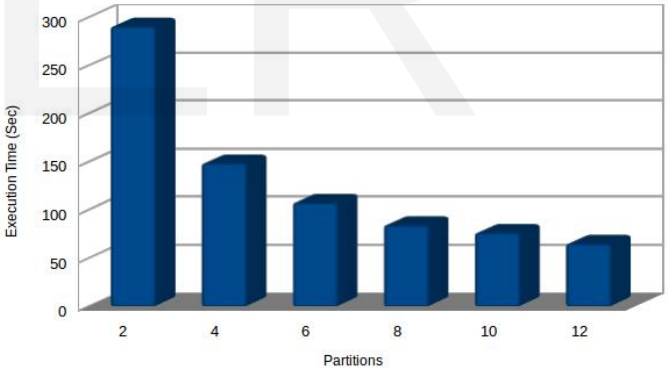

Fig. 9. Performance evaluation of SRTRE Model with fixed 2 brokers and varying partitions.

8 Conclusion and Future Work

We have proposed a scalable and real-time model for Rule-engine in which we exploit the distributed architecture of *Apache Kafka* to distribute the records and *Kafka Streams* to process the data using *drools engine*. In this model, an engine can infer only with data available on the node where it is running. This implies that inference on data with different keys may not be possible until both the keys are mapped to the same partition or partitions are mapped to the same Kafka Stream processing node. In the future work we plan to focus on the following improvements:

1. Infer conclusions from all the processing nodes.
2. Add a persistent layer to store the facts and inference made for the future analysis using one of the distributed storage systems like Apache HBase, Apache Kudu, etc.

ACKNOWLEDGMENT

We acknowledge the support of our institute for this work. We would also like to extend our thanks to Basanta Sharma for helping us out with proofreading. Finally, we would like to dedicate this work to our Founder Chancellor, Bhagawan Sri Sathya Sai Baba.

REFERENCES

[1] "Drools," http://docs.jboss.org/drools/release/7.0.0.Final/drools-docs/html_single.
[2] Xiu-li Ma, Hong-xia Wang, and Ling-yun Zhang, "Application of drools in network fault management system [j]," Computer Engineering and Design, vol. 8, pp. 015, 2009.
[3] Eduardo Mosqueira-Rey, Amparo Alonso-Betanzos, Bertha Guijarro-Berdinas, David Alonso-Rios, and J Lago-Pineiro, "A snort-based agent for a jade multi-agent intrusion detection system," International Journal of Intelligent Information and Database Systems, vol. 3, no. 1, pp. 107–121, 2009.
[4] Charles L Forgy, "Rete: A fast algorithm for the many pattern-many object pattern match problem," Artificial intelligence, vol.19, no. 1, pp. 17–37, 1982.
[5] Daniel P Miranker, TREAT: A new and efficient match algorithm for AI production system, Morgan Kaufmann, 2014.
[6] Robert B Doorenbos, "Production matching for large learning systems.," Tech. Rep., Carnegie-Mellon University Pittsburgh, PA, Department of Computer Science, 1995.
[7] Ian Wright and James A. R. Marshall, "The execution kernel of rc++: Rete*, a faster rete with treat as a special case," Int.J. Intell. Games & Simulation, vol. 2, no. 1, pp. 36–48, 2003.
[8] Jeong A Kang and Albert Mo Kim Cheng, "Shortening matching time in ops5 production systems," IEEE Transactions on Software Engineering, vol. 30, no. 7, pp. 448–457, 2004.
[9] Jinghan Wang, Rui Zhou, Jing Li, and Guowei Wang, "A distributed rule engine based on message-passing model to deal with big data," Lecture Notes on Software Engineering, vol. 2, no. 3, pp. 275, 2014.
[10] "Apache Kafka,"https://kafka.apache.org/.
[11] Kreps J, Narkhede N, Rao J. Kafka: A distributed messaging system for log processing. InProceedings of the NetDB 2011 Jun 12 (pp. 1-7).
[12] "Kafkastreams,"https://kafka.apache.org/0110/documentation/streams.
[13] "Zookeeper,"https://zookeeper.apache.org/doc/trunk/zookeeperOver.html.
[14] "Banking benchmark," https://github.com/codehaus/rulessandpit.