CSCI-550 Presentation Paper

David Kelly
Peter Ottsen

1. Event Stream Processing Overview

Event stream processing is the practice of continually analyzing a feed of data points that come from a source that is constantly creating data. The data points are referred to as ”events” and the ”stream” is the continuous creation and passing of those events. The stream is then processed as frequently as necessary or as frequently as processing power will allow.

Data or events are generated by a publisher or source then delivered to a stream processing application where the data is analyzed, then the result is sent to the subscriber or sink. This can be seen in Figure 1. Example of producers are internet of things (IoT) sensors, payment processing systems, and server or applicaiton logs.

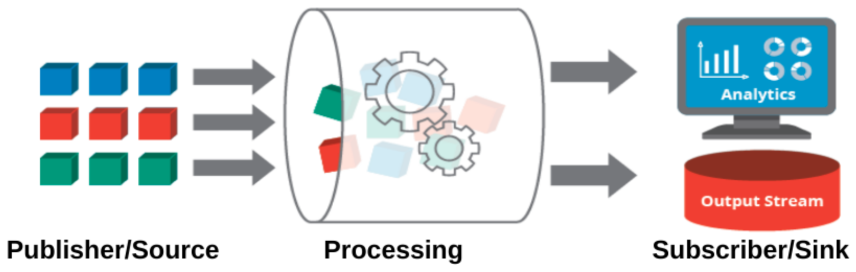

Figure 1: Event stream processing flow diagram

1.1 Evolution of ’Real-time’ Stream Processing

Stream processing is usually done in ’real-time’, but what is considered ’real-time’ has changed over the years. ’Real-time’ processing is usually done in one of three ways: batch, micro-batch or stream. Batch processing is the historical method of analysis and is still used in some applications today. Batch processing is done by analysing data at set intervals. For example, process the previous hour’s data every hour. Micro-batch processing was developed due to the increasing amount of data contained within in each event and increases in processing power. Micro-batching means more frequent batch processing, so instead of every hour, the data is batched through every second or minute. What is considered real-time stream processing today is handling a dataset a single event at a time. This allows applications to respond to new data events immediately and is possible due to increasing processing power. Figure 2 gives an illustration of the three different ways.

1.2 Real-Time Streaming Use Cases

Two major examples of real-time streaming use cases are for fraud detection and the IoT. For fraud detection, the ability to analyze each transaction as it occurs has allowed one credit card provider to block charges and save \$800M per year. While with the IoT, manufactures can recognize upsets and anomalies within the processes immediately which reduces waste and equipment costs.

2. Kafka

As real-time streaming has become more prevalent, the tools enabling it have become more advanced and optimized. Kafka is one such platform, used by 80+ percent of Fortune 500 companies. The relatively simple, but highly flexible nature of the APIs as well as the scalability of the platform due to the ability of it to run on bare metal, VM's, or in the cloud distributed across data centers has made it extremely popular with companies. Kafka supports three key capabilities of event streaming: allowing clients to publish and subscribe to events, storing streams of events durably and reliably for as long as desired, and processing streams of events as they occur or retrospectively. These capabilities can't be achieved by message queues or traditional databases. Unlike message queues, events can be processed by multiple consumers simultaneously due to the partitioned (duplicated) topics and multiple times due to the persistence of the topics. Unlike traditional databases, Kafka does not support random access or queries - instead the consumer clients subscribing to a topic are only able to lookup events by using an offset in the log. The log, synonomous for a topic, is a log data structure as opposed to an application log. It is simply a time ordered sequence of events, and as far as Kafka is concerned, the message data stored in the log can be anything because it is simply a byte array. Part of what makes Kafka so lightweight is that each topic maintains a log from which subscribers can read and derive their own representations of the data and that each consumer is responsible for tracking which messages it has consumed, rather than Kafka. Kafka has three primary APIs that enable streaming. The producer (publisher) API allows client applications to send streams of data to topics in the Kafka cluster. The consumer (subscriber) API allows client applications to read streams of data from the Kafka cluster and process them. The streams API combines the functionality of both the producer and consumer by allowing an application to read a data stream from one topic, process the stream, and write the processed stream back to another topic in the cluster. Part of what makes Kafka such a popular choice for a stream processing platform is the diversity of the clients it supports. The clients come in languages such as Java, Scala, Python, Go, and C/C++, and an organization is not limited to using only one single language for different applications due to Kafka storing data as byte arrays rather than interpreting data types itself. In conclusion, the simple functionality, high performance, persistent storage, versatile APIs, and decoupled clients/microservices make Kafka a very appealing platform for organizations looking to process streams of data with multiple applications.