Real-time Stream Processing

Kafka Streams

Prashant Pandey

Learning Journal

Kafka Streams

Real-time Stream Processing

Learning Journal

Prashant Pandey

Chapter 1 - Table of Contents

Preface i

About the Author ii

About the book iii

Who should read this book iii

What should you already know iii

Kafka and source code version iv

References and Resources v

Part - 1 Big Data and Stream Processing

Chapter 1 - Dawn of Bigdata 1

Inception 1

Volume 3

Variety 3

Velocity 4

Explication 4

Expansion 6

Summary 8

Chapter 2 - Real-time Streams 9

Conception of Events 9

Event Streams 10
Stream processing 11
Stream processing use cases 12
Incremental ETL 12
Real time reporting 14
Real-time alerting 14
Real-time decision making and ML 15
Online machine learning and AI 16
Real-time data preparation at Scale 16
Advantages of Stream processing 22
Summary 23
Chapter 3 - Streaming Concepts 24
Detaching Fallacies 24
Stream Processing Vs Analytics 24
Stream processing Vs ETL 25
Stream processing Vs Cluster Computing 25
Stream processing vs Batch processing 27
Attaching difficulties 28
Time domain 28
Event time vs Processing time 30

Time Window 32
Watermarks & triggers 35
Other window approaches 36
Stateful streams 37
Stream table duality 38
Exactly once processing 41
Summary 43

Preface

Welcome to the first edition of Kafka Streams: Real-time Stream Processing. I am excited to bring you a valuable resource on Apache Kafka. This book is focusing mainly on the new generation of the Kafka Streams library available in the Apache Kafka 2.0. The primary focus of this book is on Kafka Streams. However, I will also touch base on the other Kafka capabilities and concepts that are necessary to grasp the Kafka Streams programming.

Apache Kafka is currently one of the most popular and necessary components of any Big Data solution. Initially conceived as a messaging queue, Kafka has quickly evolved into a full-fledged streaming platform. As a streaming platform, Kafka provides a highly scalable, fault tolerant, publish and subscribe pipeline of streams of events. With the addition of Kafka Streams library, it is now able to process the streams of events in real-time with millisecond responses to support a variety of business use cases.

We hope this book gives you a solid foundation to write modern real-time stream processing applications using Apache Kafka and Kafka Streams library. In the next section, I will tell you a little bit about my background, who are the intended audiences of this book, and how I have organized the material.

About the Author

Prashant is passionate about helping people to learn and grow in their career by bridging the gap between their existing and required skills. In his quest to fulfil this mission, he is authoring books, publishing technical articles, and creating training videos to help IT professionals and students succeed in the industry.

With over 17 years of experience in IT as a developer, architect, consultant, trainer, and mentor, he has worked with international software services organizations on various data-centric and big data projects.

Prashant is a firm believer in lifelong continuous learning and skill development. To popularize the importance of lifelong continuous learning, he started publishing free training videos on his YouTube channel (www.youtube.com/learningjournalin) and conceptualized the Idea of creating a Journal of his learning under the banner of Learning Journal.

He is the founder, lead author, and chief editor of the portal (www.learningjournal.guru) that offers various skill development courses, training, and technical articles since the beginning of 2018.

About the book

I am writing Kafka Streams: Real-time Stream Processing! to help you understand the stream processing in general and apply that skill to Kafka streams programming. My approach to writing this book is a common-sense approach to teaching a complex subject. By using the common-sense approach, I will help you to apply your general ability to perceive, understand and reason the concepts that I am explaining in this book.

Who should read this book

Kafka Streams: Real-time Stream Processing! is written for software engineers willing to develop stream processing application using Kafka streams library. I am also writing this book for data architects and data engineers who are responsible for designing and building the organization's data-centric infrastructure. Another group of people is the managers and architects who don't directly work with Kafka implementation, but they work with the people who implement Kafka Streams at the ground level.

What should you already know

This book assumes that the reader is familiar with the basics of Java programming language. The source code and examples in this book are using Java 8, and I will be using Java 8 lambda syntax, so experience with lambda will be helpful.

Kafka Streams is a library that runs on Kafka. Having good fundamental knowledge of Kafka is essential to get the most out of Kafka Streams. I will touch base on the mandatory Kafka concepts for those who are new to Kafka, and you should be able to learn Kafka Streams that is the main subject of the book.

The book also assumes that you have some familiarity and experience in running and working on the Linux operating system.

Kafka and source code version

This book is based on Kafka Streams library available in Apache Kafka 2.0. All the source code and examples in this book are tested on Apache Kafka 2.0 open source distribution.

Some chapters of this book also make use of Confluent Open Source platform to teach and demonstrate functionalities that are only available in Confluent Platform such as prebuild connectors, KSQL and Schema Registry.

References and Resources

- Apache Kafka Documentation
  https://kafka.apache.org/
- Confluent Documentation
  https://docs.confluent.io/current/
- Confluent Blogs
  https://www.confluent.io/blog
- Wikipedia
  https://en.wikipedia.org/wiki/Web_search_engine
- Twitter timeline fanout video.
  https://www.infoq.com/presentations/Twitter-Timeline-Scalability
- Stream processing 101
  https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Part - 1

Big Data and Stream Processing

In part 1 of this book, I will talk about the beginning and rise of the big data era in general. We will then discuss how the big data problem is eventually progressing into real-time stream processing requirements. We will also try to understand the importance of real-time stream processing in the light of some common industry use cases. Finally, I will close the part 1 of the book talking about unique challenges of real-time stream processing and how it differs from batch processing of large volumes of data.

Chapter 1 - Dawn of Bigdata

The internet and the world wide web had a revolutionary impact on our life, culture, commerce, and technology. The search engine has become a necessity in our daily life. However, until the year 1993, the world wide web was indexed by hand. Sir Tim Berners-Lee used to edit that list by hand and host it on the CERN web server. Later, Yahoo emerged as the first popular method for people to find web pages on the internet. However, it was based on a web directory rather than a full-text index of the web pages. Around the year 2000, a new player Google rose to the prominence by an innovative idea of PageRank algorithm. The PageRank algorithm was based on the number and the domain authority of other websites that link to the given web page. One of the critical requirements of implementing the PageRank algorithm was to crawl and collect a massive amount of data from the world wide web. This is the time and the need that I would attribute to the birth of Big data.

Inception

Google was not only the first to realize the Big data problem, but also the first to develop a viable solution and establish a commercially successful business around the solution. At a high level, Google had four main problems to solve.

1. Discover and crawl the web pages over world wide web and collect the information about those pages.
2. Store the massive amount of data collected by the web crawler.
3. Apply the PageRank algorithm on the received data and create an index to be used by the search engine.
4. Organize the index in a random-access database that can support high-speed queries by the Google search engine application.

These four problems are at the core of any data-centric application, and they translate into four categories of the problem domain.

1. Data Ingestion/Integration
2. Data storage and management
3. Data processing and transformation
4. Data access and retrieval

These problems were not new, and in general, they existed even before the year 2000 when Google was trying to solve them using a new and innovative approach. The industry was already generating, collecting, processing and accessing data and the database management systems (DBMS) like Oracle and MS SQL server were in the centre of such applications. At the time, some organizations were already managing terabytes of data volume using systems like Teradata. However, the problem that Google was trying to solve

had a combination of three unique attributes that made it much more difficult to be addressed using DBMS of the time. Let's try to understand these attributes that uniquely characterize the Big data problem.

1. Volume
2. Variety
3. Velocity

Volume

Creating a web crawler was not a big deal for Google. It was a simple program that takes a page URL, retrieves the content of the page and stores it in the storage system. But in this process, the main problem was the amount of data that the crawler collects. The web crawler was supposed to read all the web pages over the internet and store a copy of those pages in the storage system. Google knew that they will be dealing with the gigantic volume of data and no DBMS at the time was scalable to manage that quantity.

Variety

The content of the web pages had no structure. It was not possible to convert them into a row/column format without storing and processing them. Google knew that they need a system to deal with the raw data files that may come in a variety of formats. The DBMS of the time had no meaningful support to deal with the raw data.

Velocity

Velocity is one of the most critical and essential characteristics of any usable system. Google needed to acquire data quickly, process it and use it at a faster rate. The speed of the Google search engine has been its USP since they came into the search engine business.

It is the speed that is driving the new age of big data applications that need to deal with the data in real-time. In some cases, the data points are highly time sensitive like a patient's vitals. Such data elements have a limited shelf-life where their value can be futile with time - in some cases, very quickly. I will come back to the velocity once again because the velocity is the driving force of a real-time system – the main subject of this book.

Explication

Google successfully solved all the above problems, and they were generous enough to reveal their solution to the rest of the world in a series of three white papers. These three whitepapers talked about Google's approach to solving data storage, data processing, and the data retrieval.

The first whitepaper was published by Google in the year 2003. (https://ai.google/research/pubs/pub51). The first paper elucidated the architecture of a scalable, distributed filesystem which they termed as Google File System (GFS). The GFS solved their problem

of storage by leveraging commodity hardware to store data on a distributed cluster.

The second whitepaper was published by Google in the year 2004 (https://ai.google/research/pubs/pub62). This paper revealed the MapReduce (MR) programming model. The MR model adopted a functional programming style, and Google was able to parallelize the MR functions on a large cluster of commodity machines. The MR model solved their problem of processing large quantities of data in a distributed cluster.

The last paper in this series was published by Google in the year 2006 (https://ai.google/research/pubs/pub27898). This paper described the design of a distributed database for managing structured data across thousands of commodity servers. They named it Google Bigtable and used it for storing petabytes of data and serve it at a lower latency.

All these three whitepapers were well appreciated by the open source community, and they formed the basis for the design and development of similar open source implementation – Hadoop. The GFS is implemented as a Hadoop distributed file system – HDFS. The Google MR is implemented as Hadoop MapReduce programming framework on top of HDFS. Finally, the Bigtable was implemented as Hadoop database – HBase.

Since then, there are many other solutions developed over the Hadoop platform and open sourced by various organizations. Some of the most widely adopted systems are Pig, Hive and finally Apache Spark. All these solutions were simplification and improvement over the Hadoop MapReduce framework, and they shared the common trait of processing large volumes of data on a distributed cluster of commodity hardware.

Expansion

Hadoop and all the other solutions developed over the HDFS tried to solve a common problem of processing large volumes of data that was previously stored in the distributed storage. This approach is popularly termed as batch processing. In the batch processing approach, the data is collected and stored in the distributed system. Then a batch job that is written in MapReduce is used to read the data and process the same by utilizing the distributed cluster. While you are processing the previously stored data, the new data keeps arriving at the storage. Then you must do the next batch and then take care of combining the results with the outcome of the previous batch. This process goes on in a series of batches.

In the batch processing approach, the outcome is available after a specific time that depends upon the frequency of your batches and the time taken by the batch to complete the processing. The insights derived from these data processing batches are valuable. However, all such insights are not equal. Some insights may have a

much higher value shortly after the data was initially generated and that value diminishes very fast with the time.

Many use cases need to collect the data about the events in real-time as they occur and extract an actionable insight as quickly as possible. For example, a fraud detection system is much more valuable if it can identify a fraudulent transaction even before the transaction completes. Similarly, in a healthcare ICU or in the surgical operation setup, the data from various monitors can be used in real time and generate alerts to nurses and doctors, so they know instantly about changes in a patient's condition.

In many cases, the data points are highly time-sensitive, and they need to be actioned in minutes, seconds or even in milliseconds. Enormous amounts of time-sensitive data are being generated from a rapidly growing set of disparate data sources. We will discuss some of the prevalent use cases and data sources for such requirements in the next chapter. However, it is essential to comprehend that the demand for the speed in processing the time-sensitive data is continuously pushing the limits of Bigdata processing solutions. These requirements are the source of many innovative and new frameworks such as Kafka Streams, Spark streaming, Apache Storm, Apache Flink, Apache Samza and cloud services like Google cloud dataflow and Amazon Kinesis. These new solutions are evolving to meet the real-time requirements and are called by many names. Some of the commonly used names are

Real-time stream processing, Event processing, and Complex event processing. However, in this book, we will be referring it as stream processing.

In the next chapter, I will talk about some popular industry use cases of stream processing. It will help you to develop a better understand of the real-time stream processing requirements and differentiate it from the batch processing needs. You should be able to clearly identify and answer when to use real-time stream processing vs. batch processing.

Summary

In this chapter, I talked about the big data problem and how it started. We also talked about the innovative solutions that Google created and then shared it with the world. We briefly touched on the opensource implementation of Google whitepapers under the banner of Hadoop. Hadoop grabbed immense attention and popularity among the organizations and professionals. We further discussed the growing expectation and the need to handle time-sensitive data at speed and why it is fostering new innovations in real-time stream processing. In the next chapter, we will progress our discussion about real-time stream processing to develop a better understanding of the subject area.

Chapter 2 - Real-time Streams

In the earlier chapter, I talked about the genesis of Bigdata, and we learned that how it is expanding into the real-time requirements. In this chapter, we will try to develop a broader sense of the following.

1. What is a stream?
2. What do we mean by stream processing?

We will start by developing a general sense of stream processing and elaborate the subject in the light of some simple use cases.

Conception of Events

Big data emerged with a focus on storing the data first and then processing large volumes of data afterward. The advent of the Data Lake is the result of the same idea. The data lake is the place where data goes to sit, and later you process it in smaller batches.

While many are building upon the data lake, at the same time, a whole new fleet of technology companies like Uber, Netflix, Twitter, Spotify, and many more start-ups are emerging and differentiating themselves. They are architecting their systems in a non-traditional method that takes everything happening in the business and make it available as the sequence of events.

The notion of looking at the day to day business activities as a sequence of events is in the foundation of the real-time stream processing systems. A retailer has a series of events for orders, returns, shipments and so on. A bank has events of customer requests, stock ticks, market indicators, trades, and financial time series transactions. Websites have an event of clicks and impressions. Google AdSense is all about matching a bid event to the ad request event in real-time and then tracking the ad click events.

The first step of heading towards the stream processing is the conception of business activities as events.

Event Streams

An individual event happening in business is discrete, but these events are happening as a continuous stream. The event stream is nothing but a constant flow of discrete business events that are happening in real-time, and in this book, we refer to them as a real-time stream of events.

The real big of the Bigdata is created by the act of capturing each event of the business and put them to use for decision making. However, we are talking about the change in the perspective to look at those events as a real-time stream rather than a dataset stored in the data lake. This new perspective of the data as a stream of events may seem a little foreign to people who are

accustomed to thinking of data as a table in a database or a file in the data lake. The data lake is stationary in nature, but the stream refers to the “Data in Motion.”

Once you are comfortable with the notion of looking at the data as a real-time stream of events rather than a file or a table, you are ready to dive into the new era of stream processing.

Stream processing

The notion of processing event streams appears in many different areas, and hence different people in the various field may have a different vocabulary to refer to the same thing. However, in general, the stream processing is the act of continuously incorporating new data to compute a result. Processing a stream is all about dealing with a series of event that arrives at the stream processing system.

It is essential to understand that the stream has no predetermined beginning or end. It just forms a series of unbounded data packets such as credit card transactions, clicks on a website, or sensor readings from an IoT device. The stream processing applications continuously incorporate this data into the system and compute various queries over this stream or perhaps join them with some other data sets and channel the outcome to external storage.

Stream processing use cases

Everything that I talked so far should be good enough to help you develop a general perception of what the stream processing means, and what we do in a stream processing system. However, to cement the idea and concrete your understanding, I am going to give five example use-cases of stream processing. While it is not an exhaustive list, but I have arranged them in the increasing order of complexity. A typical business may want to start with the first category of the use case and progress towards the more complex use case.

Finally, I will talk about the sixth use case in little more detail to develop a holistic understanding of the streaming processing implementation for an internet scale application.

Incremental ETL

Incremental ETL is one of the most common requirements for the enterprises. They turn their operational data stores into a stream of transactions by applying some change data capture (CDC) tool and bring those changes to the data warehouse or data lake at low latency. A figure below shows a typical implementation of CDC in a data warehouse or data lake setup. The change producer process continuously monitors the redo logs and produces a stream of events for the consumer on the other side. The consumer materializes the stream back to the target database. Most of the

CDC tools would need another system to configure, coordinate and monitor the whole process.

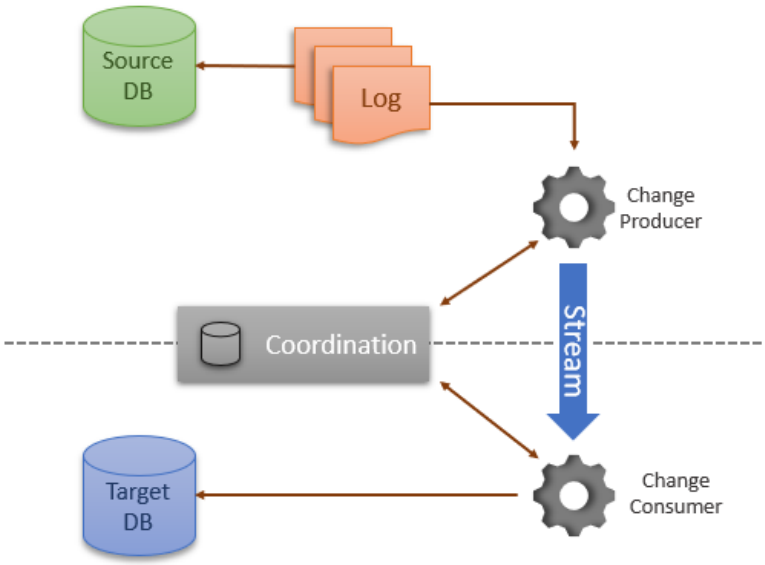

Figure 2.1 - CDC turns your database into a stream

The change consumer process may be as simple as continuously bringing data to the data lake, but in many cases, we can also implement a sequence of transformations on the fly. Such data transformations may include data quality checks, data enrichment, and other data enhancement. Incremental ETL is the most basic streaming application, and it is quite popular because it allows you to incorporate new data within seconds, enabling users to query it faster downstream. We will come back to this use case in part two of the book and discuss it further in greater detail.

Real time reporting

Many organizations use stream processing applications to run real-time dashboards. The real-time dashboarding is a popular category, and some specific use cases are listed as following.

1. Infrastructure monitoring – Every platform team would use a dashboard to monitor the platform KPIs such as storage, CPU, network usage, overall system load, uptime, etc. Another excellent example of monitoring is to watch the adoption of new features as they are rolled out among applications.
2. Campaign management – Internet applications continuously monitor the effects that new campaigns and site changes have on their traffic such as to see whether a one-day promotion is driving traffic to your site?
3. Business dashboards – Businesses use real-time dashboards to monitor KPIs such as a delivery backlog, on time shipment, loading time indicators, etc. These KPIs when generated in real-time are critical for planning, organizing and taking in-time decisions for day to day operations.

Real-time alerting

Notification and alerting are probably the most obvious streaming use case. In a typical example, these applications identify a pattern of events or continuously compute a threshold to trigger an alert. Some specific use cases are listed below.

1. Supply chain alerts – Real-time notifications are pushed to the mobile devices of the warehouse employees for the fulfilment of critical orders.
2. Healthcare monitoring – Patient's vital readings are continuously monitored remotely against a threshold by using sensors and alerts are generated to provide efficient medical services in time.
3. Traffic monitoring – These systems use traffic data streams to immediately detect incidents and reduce the impact by initiating timely actions.

Real-time decision making and ML

Real-time decision making involves analysing new inputs and responding to them automatically using business logic or applying some machine learning models. This business logic may include a set of simple business decisions or an application of medium to sophisticated machine learning model, but these systems make clear, concrete and automated decisions. Some specific examples are listed below.

1. Fraud detection - A bank that wants to automatically verify whether a new transaction on a customer's credit card represents fraud by applying a decision tree and deny the operation if the charge is determined fraudulent.
2. Real-time clinical measurement - Visensia safety Index is real-time data-driven clinical measurement. It can make accurate

predictions about the patient's deterioration up to 24 hours earlier. Such predictions should permit medical interventions to save both lives and money.

3. Real-time bidding – When a user visits a webpage, a request for ad goes to the exchange where multiple advertisers bid for the ad space in real-time. However, the decision goes in favour of the highest bidder.
4. Personalized customer experience – Customer experience is critical for every industry but for our purpose, let's take an example from an online gaming system. Gaming platforms implement adaptive gameplay to fine tune the difficulty level ensuring that the session is sufficiently challenging without being boring or frustrating.

Online machine learning and AI

Online machine learning is the real artificial intelligence and the most challenging area of stream processing and machine learning. In this technique, the real-time data is used to update the best prediction at each step. In many cases, the online learning dynamically adapts the new patterns in the data and optimize the learning model in real-time. There is not much work done in this area and still undergoing various research.

Real-time data preparation at Scale

So far, we have seen a bunch of use cases where generating and processing a stream of real-time events make sense. By now, you

must have clarity on where the real-time stream processing is applied in the industry. In this section, I want to talk about an example of real-time stream processing in little greater detail. The example should help you to develop a bigger picture of critical requirements for implementing a real-time streaming use case at internet scale.

Let's take a simple problem from social media applications such as LinkedIn, Facebook and Twitter. They have the challenge to support millions of active users at any point in time. These users are continuously performing two types of operations.

1. Create content
2. Consume content

The create content is a write operation, and the consume content is a read operation. The big question is, how do you support these operations for such a huge population?

Let's look at a simple high-level approach represented by the following figure.

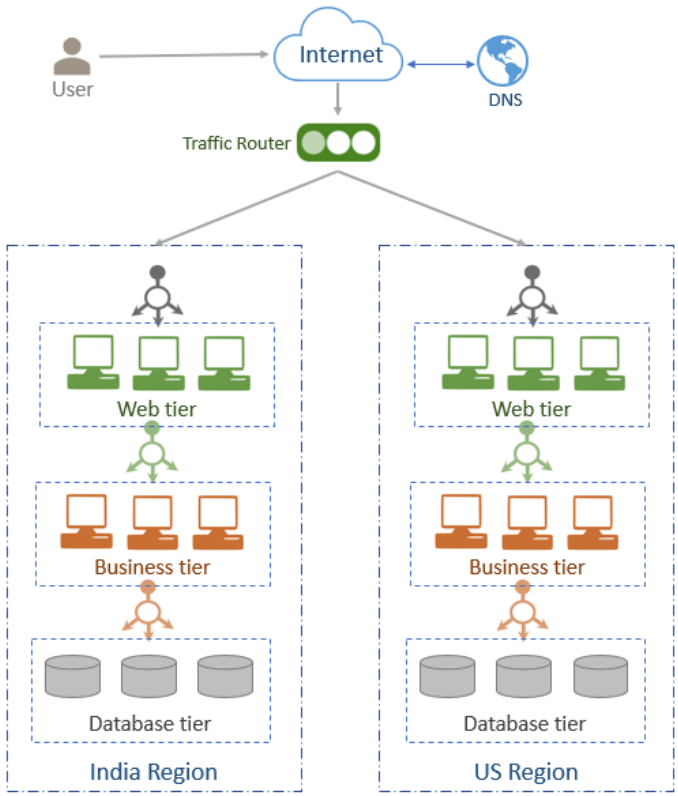

Figure 2.2 - Scalable web application

The above figure is a pictorial representation of some key ideas that are listed below.

1. Implement a distributed architecture and serve a finite set of users from a partition of your distributed system.
2. Implement an intelligent load balancing to redirect user request to their respective services and then to their separate database partitions.

3. Design your system to ensure that the users are served from their nearest datacentre at the lowest network delay.
4. Maintain an in-memory cache of the active users to boost up the performance.

The above approach can offer a predictable performance to your users for the write operations. However, the read operations may still suffer from poor response time. Let's try to understand it further with a concrete example. I will be quoting LinkedIn to help you make sense of the example. However, LinkedIn may or may not have implemented things in the way I explain in this example. The discussion is only illustrative.

Let's assume I am using LinkedIn and they have a dedicated database partition based on my user-id where all my contents are stored. I can create a post, I can write some comment, I can like a comment and many other things. All these activities are generating some content that goes and sits into my database partition. The write operation is efficient because the load balancer redirects my request to the service that stores and serves my content in my database partition.

Now let us look at the read operation. The read operation also appears to be quick. I open the LinkedIn App or the website, based on my user ID, the page load is redirected to the service that serves me.

The service reads the page content from my database partition and servers it very quickly. All looks good. However, there is a catch.

Some of the key ingredients of my LinkedIn page are my feed, my messages, and my notifications. Where are they coming from? These are not the content that I created, and hence they were not supposed to be in my storage partition. My LinkedIn feed is the content that is produced by the people I follow. My notifications are based on the activities of other people on my content such as they commented or liked one of my posts.

What does all that mean? Well, these are the events that are being generated by some other sources such as the people whom I follow, the people who comment on my post, the ad that is identified relevant for me. Those events are flowing to my database partition in real-time so that they can be served to me quickly.

The figure below explains the process at high level. A working solution to the above problem starts by creating an event for every user action. The event can be represented by immutable data records or a message object that goes to a message bus. A stream processing application listens to the message bus and picks up the messages of interest. Then it is the responsibility of the stream processor to channel those messages to the appropriate services that persist the message in the concerned user's storage. That's how your activities may be flowing to the feed of all those who are following you in your favourite social media platform.

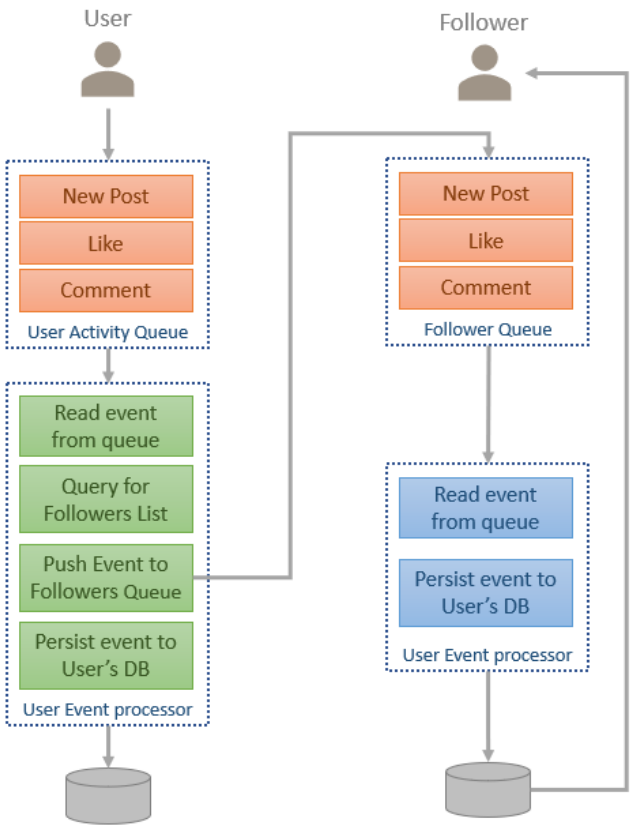

Figure 2.3 - Real-time data preparation in social media

This is what we call the real-time data preparation and routing to the appropriate data consumers. The solution to this problem could be an excellent implementation of real-time stream processing.

At first instance, this process might appear like the combination of real-time incremental ETL and real-time reporting use cases. However, it not the exact same thing. Twitter has such a similar

process for Twitter timelines, and they call it the Fanout Service (https://www.infoq.com/presentations/Twitter-Timeline-Scalability).

Advantages of Stream processing

As we have seen some use cases of stream processing, let's try to blot some primary advantages of stream processing systems. Stream processing is essential to respond in real-time at low latency. When your application needs to react quickly in milliseconds or seconds, you need a streaming system that can keep the required data ready to get the acceptable performance.

Stream processing is great for scalability because they allow you to decompose your application into event producers and event processors. You can even disintegrate your event processors based on different events and various independent data flows. All of them can operate independently and can take advantage of more parallelism across multiple machines.

In the next chapter, I will talk about some critical requirements of a typical stream processing platform and also talk about the main challenges and concepts that are necessary to understand before you start operating on real-time streams.

Summary

In this chapter, I helped you start looking at your data as events and how you can envisage your data as a continuous stream of events. The notion of event stream is the foundation to start the development of stream processing applications. We talked about a bunch of use cases to help you understand what the stream processing means and how it is being used in the industry. I also covered an example to give you a sense of how a stream processing pipeline can be used to solve a stringent response time expectation for an internet scale application such as LinkedIn and Twitter. Finally, I talked about the main advantages of stream processing. In the next chapter, we will take our stream processing discussion further and try to understand the key concepts associated with the domain of stream processing.

Chapter 3 - Streaming Concepts

In the earlier chapter, we learned that most of the business activities can be modelled as a stream of events. Likewise, there is a lot of hype around “machine-generated data” and “Internet of things.” These buzzwords may have a different meaning for different people, but a considerable part of these areas is about the collection and processing of big data streams. Like any other domain, stream processing has its own set of challenges and concepts. In this chapter, I will help you to familiarise yourself with the challenges and concepts that you will be dealing with while developing your streaming solutions.

Detaching Fallacies

The popularity and increasing demand for stream processing is a natural progression of Bigdata evolution. Most of the stream processing platforms are built by taking capabilities from the big-data batch processing world and making them available for a low-latency domain. This approach created some misconception around the notion of stream processing. In this section, I will try to highlight the central fallacies of stream processing.

Stream Processing Vs Analytics

People often take stream processing synonymous with real-time analytics. Real-time analytics is one important use case of stream

processing. However, it doesn't always mean analytics. There are many more applications of real-time stream processing. More often, your stream processing application would implement core functions in the business rather than computing analytics about the business. Analytics may be an individual service along with many other services on your streaming platform. Stream processing is a backbone infrastructure and a set of services that coordinate with analytics function. It is more like the glue that binds your platform to deliver the business processes that may also include analytics.

Stream processing Vs ETL

Some people take the stream processing as an ETL pipeline that works in real-time. That might be a good starting point on stream processing. However, stream processing applications require addressing needs that are very different from the ETL domain of conventional big-data processing. Stream processing application is more like microservices rather than a scheduled job. Like a typical microservice, stream processing applications might not be processing requests over HTTP, but they operate on asynchronous event streams over a pull mechanism. You can take the stream processing applications as an application programming model for asynchronous services rather than an ETL framework.

Stream processing Vs Cluster Computing

Cluster computing or the distributed computing is all about sharing the workload over a pool of resources in the cluster of computers.

This is often implemented using a framework that is responsible for providing the fault tolerance, scalability, and many other orchestration capabilities. These cluster management frameworks manage the resources in the form of containers which is nothing but a logical bundle of cluster resources such as memory, CPU, networking and in many cases operating system image as well. This area is going under a separate evolution, and we can see a further decoupling of responsibilities for greater flexibility. Docker containers have decoupled the container configuration and packaging from the cluster management. The Kubernetes is trying to solve the problem of resource allocation and placement of containers in a fault tolerant and scalable cluster.

On the other side, a stream processing application has a different domain of problems and challenges while dealing with the stream of events in the real-time. They have nothing to do with the cluster management and resource allocation, and in fact, you would want to have the flexibility to take advantage of the separate developments that are happening in the cluster computing space.

Many of the big-data processing frameworks such as Apache Spark assumes the responsibility of packaging the dependencies, serialize your code and send it to the workers over the network. This process has an inherent delay to start the execution, and it also imposes a restriction on packaging and deployment flexibility that in turn takes away a bunch of advantages of CI/CD and few other things.

To summarize this discussion, when you are designing a stream processing solution, you may not want to get trapped into the cluster management requirements. You stream processing application should not have a dependency on the cluster management frameworks. In fact, the cluster management platform should be entirely optional for your streaming application as well as you should have the flexibility to take advantage of any of such mature platforms.

Stream processing vs Batch processing

The first and most obvious thing that you would notice about the stream processing is that you are going to deal with an unbounded, ever growing, infinite dataset that is continuously flowing to your system.

In contrast, the batch processing is dealing with a bounded, fixed and finite data set that has already landed at your system in the past. If you carefully think about the bounded dataset, it is nothing but a small subset of the unbounded dataset that is chopped into a smaller set from the stream.

The approach taken up by a micro-batch processing system is a bottom-up approach where they first solved a subset of the problem by processing smaller batches and tried to enhance and extend the batch method to solve a much bigger problem of dealing with infinite streams.

However, the systems that are initially designed for the stream processing in mind, they would have the flexibility to take a radical approach to solve a bigger problem, and then use the same method to deal with the smaller batches of bounded data, that are in fact a subset of the problem that the stream processing system already solved. The point is straight. A well designed, efficient stream processing system will eventually eliminate the need for the batch processing systems.

Attaching difficulties

Like any other technical domain, stream processing also poses a unique set of challenges. Before we start working with streams, it is critical to have some good sense of the difficulties that you are going to handle while dealing with the event streams. In this section, I will introduce you to some new problems that you are likely to manage and help you start building some new concepts. These new concepts are instrumental for the stream processing solution design.

Time domain

One of the capabilities of the stream processing system is the power to extract value out of the time-sensitive events before the value vanishes. For example, healthcare systems want to send an alert to nurses as soon as possible. With time, the value of such alarm disappears. This time sensitivity is not only associated with the life-critical systems. Time makes sense in many other use cases.

For example, your application is processing a stream of ticket booking events.

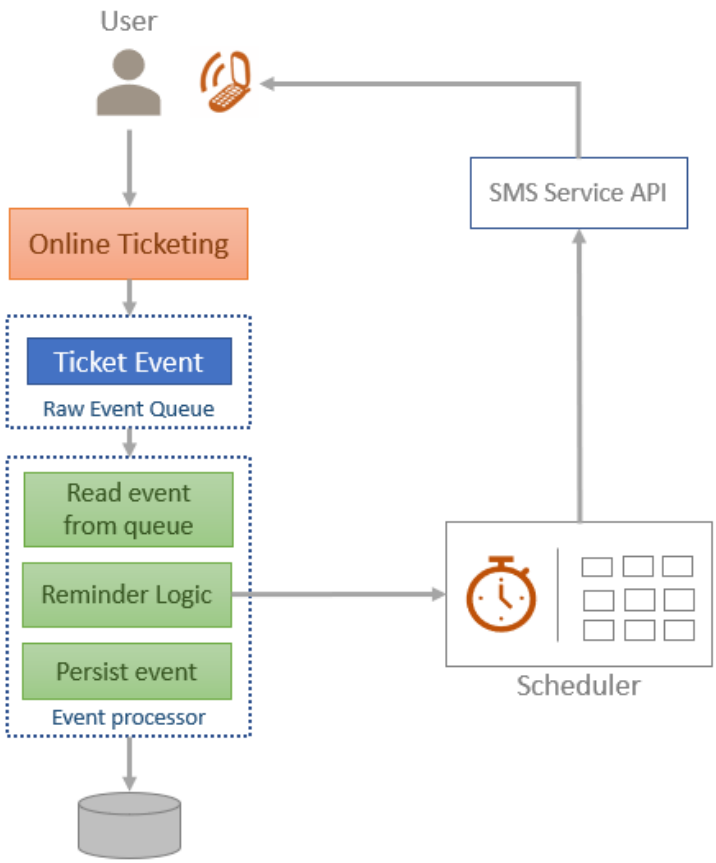

Figure 3.1 - Event stream of online reservation

As the event is received to your application, you want to schedule a reminder that should be sent a couple of hours before the show. However, such a reminder doesn't make sense If the ticket is booked fifteen minutes before the show, and hence you want to

filter out those tickets from flowing to the scheduler. The point is straight. The timestamping of the event is critical for many stream processing use cases, and hence you may have to attach a timestamp with your events.

Event time vs Processing time

For a stream processing system, we often care about two types of timestamps.

1. Event time
2. Processing time

Event time is the time that an event happened in the real world. For example, the time when a ticket was booked by an online ticket booking platform. Another example would be the time when the search button is clicked by the user to search some product on your eCommerce website.

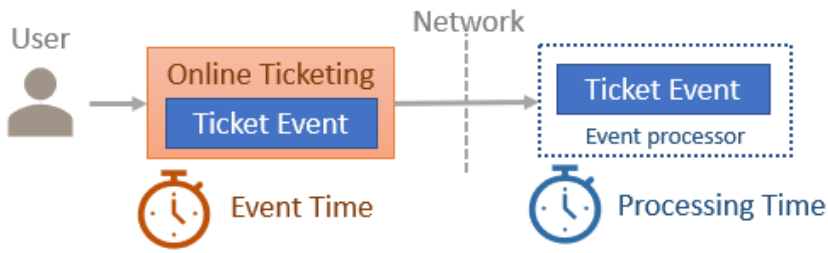

Figure 3.2 - Event time vs Processing time

Processing time is the time that the event is observed by the machine that is processing it. For example, the time when the ticket booking event is received at your stream processing system that takes a decision to schedule an alert.

Both notions of time are useful depending on the application. For example, a ticket is booked at 10.30 AM for the show that starts at 11.00 AM. For some strange reasons, the ticket booking event reaches your stream processing application at 11.05 AM. You may not want to send the reminder notification now. The show should have already started. In this case, the processing time makes more sense over the event time.

On the other hand, if you are computing a search pattern over time to learn what products are being searched on your eCommerce website in the morning, versus afternoon, versus evening. For such computations, the event time makes more sense.

In an ideal world, the event time and the processing time should be same assuming it takes no time to reach the machine that processes the event. However, in a real world, the processing time would be later than the event time due to the network latency, resource sharing and a variety of other reasons.

These differences in event time and processing time may be as small as few milliseconds to many hours or even days. For example, you are collecting usage patterns for some mobile application that

works online as well as offline. This data can be transmitted to your servers only when the mobile device has connectivity. If the user goes to an area where he loses the connectivity for a few hours, you may get those events after hours. If the user switches off the mobile data for a couple of days, this delay may be as significant as a few days.

Time Window

Many of the standard data processing operation appears to be time agnostic such as filter, join and grouping. Let's assume that you are processing web traffic log entries. You want to group the records by the country of origin and count them. For this simple requirement, you may not have to consider event time or the processing time. However, the number of users by country may not be a useful metric because it keeps increasing forever. Breaking the ever-growing count in temporal boundaries should make more sense. For example, the number of users per hour or per minute for a given country would be more meaningful. The point is straightforward. Some of the requirements would need you to slice the time domain in precise time windows. You would want to collect the events generated or received within the window and perform some analysis on the whole set.

Your time window may be of two types: tumbling (fixed) or sliding.

Tumbling window

Tumbling window is fixed and non-overlapping time window. To visualize the notion of tumbling window, let's consider an input stream as shown below.

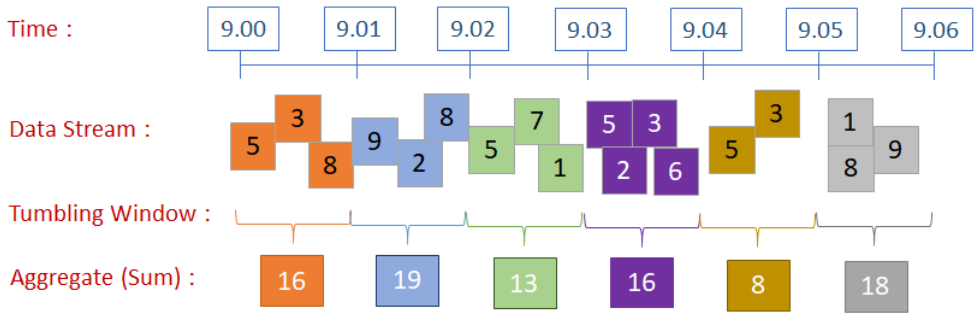

Figure 3.3 - Tumbling Window

The first row represents the time. Each number in the second row represents a data point. The next line groups the data by a tumbling window of one minute. Finally, we take a sum of the data points for each tumbling window.

Sliding window

Sliding windows are fixed length, but they slide over the period. Let's consider the same input stream that we used in the tumbling window and see how a sliding window gives you a different perspective of the time.

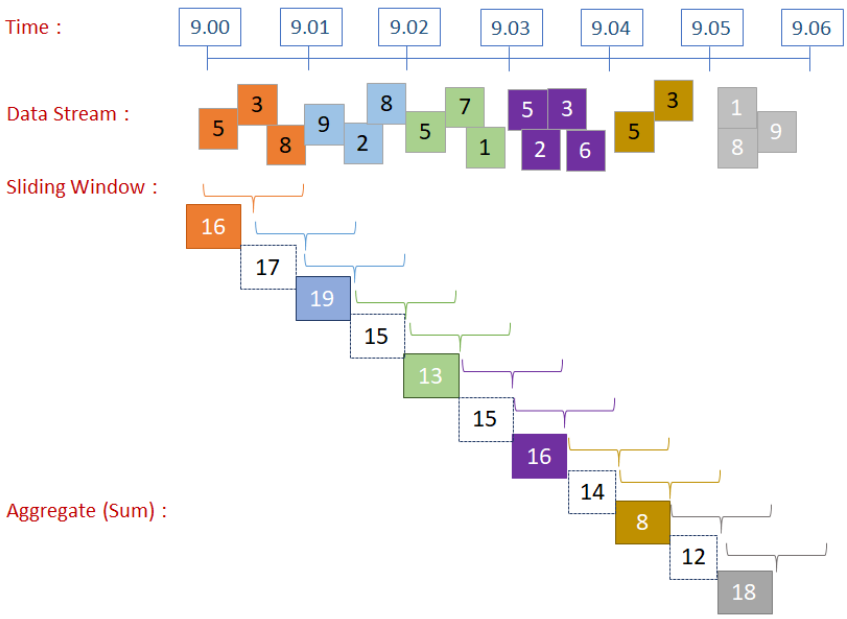

Figure 3.4 - Sliding Window

The first row represents the time. Each number in the second row represents a data point. The next line groups the data by a window of one minute that slides every thirty seconds. Finally, we take a sum of the data points for each sliding window.

Now let's talk about how these two windows are different. While using a tumbling window, you would buffer the data for a minute and then take a sum of all the data points. In this case, every computation has a delay of one minute. However, the sliding window updates the sum every thirty seconds for the most recent one minute. You would eventually get the same result as the fixed

window, but you also get an early sense by an intermediate result after every thirty seconds.

Watermarks & triggers

We learned that you may want to slice up your event stream into time-based windows and aggregate your results over time such as per second or per minute. However, the windowing introduces a new problem of latecomers. The stragglers can jeopardize the correctness of your results.

Let's assume that you are you are monitoring traffic by computing the number of vehicles pass by every minute.

You started at 9.00.00 AM and wanted to sum up the number of cars that passed by between 9.00.00 and 9.01.00. However, the event takes five seconds to reach your application. So, the event that generated at 9.00.00 will arrive at your system at 9.00.05, and similarly, the event created at 9.00.59 will reach to your system by 9.01.04.

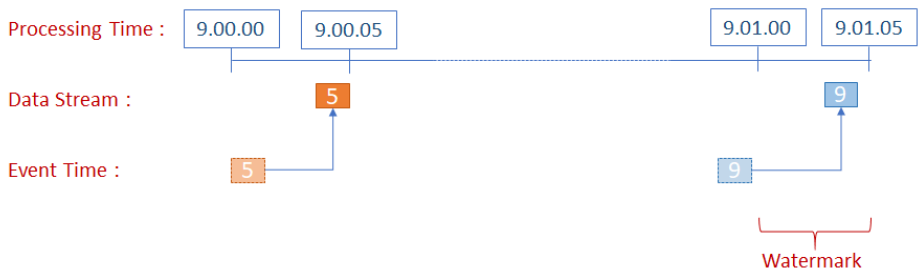

Figure 3.5 - Notion of Watermark in streaming

So, to compute the correct results, you should wait until 9.01.05 to ensure that all events that are generated between 9.00 and 9.01 successfully reached your application. If you calculate your sum before that time, your output is incorrect.

The notion of waiting for the stragglers is known as watermarks or the triggers. Different stream processing system handles it differently. However, the idea is termed as the watermark. Watermarks are a feature of streaming systems that allow you to specify how late they expect to see data in event time. The triggers in this respect are the mechanism to end the waiting of the new events and trigger the computation. Watermarks and triggers are essentially an approach to handle stragglers.

Handling late record is one of the most complex problems of the stream processing system for the fact that you can't accurately predict the delay in most of the cases. It is tough to guarantee that your application has received all the data that are generated before specific event time, and you do not have anything hanging somewhere due to some unknown reasons. This uncertainty of stragglers is the greatest streaming challenge.

Other window approaches

Time is one of the most common approaches to implement the notion of a window. One of the main reasons to use the time for windowing is that the time always goes on, and the time window will eventually close for sure. Another approach is to implement a

window on the count of events. In this approach, we wait for some x number of records to trigger the computation. For example, you may want to wait for 100 records to arrive and then trigger the calculations. Such notions should be implemented with extra care because in certain conditions you may be waiting forever as you never got 100 records.

Another common approach is to base your window on the notion of session. In general sense, a session is a period of activity that is preceded and followed by a period of inactivity. For example, a series of interactions of a user on a website, followed by no further response for a certain period may be termed as an end of the session. Sessions are often implement using timeouts because they typically do not have a set duration or a set number of interactions. The timeout basically specifies how long we want to wait until we believe that a session has ended.

Stateful streams

A state is nothing but a persistent and durable store where the application can buffer or store some data for later reference.

Many of the stream processing use cases are stateless, and they do not need to maintain any state. For example, when you are processing one event at a time, and all you need to do is to transform the event and pass it to some other processor to take necessary action. Simple transformations and filers on individual events are the most common examples that may not need to

maintain a state. However, most of the stream processing applications need to keep some state, and hence they should be categorized as stateful applications.

Your application is stateful whenever it needs to aggregate, implement a window, perform a join operation, or ensure that they can be stopped and resumed.

One of the common reasons for streaming applications to maintain a state is to avoid reprocessing of huge volumes in case of failures. You may need the ability to resume from wherever you were paused or stopped. Since a streaming application works on a continuous stream of data, and if they need to go back in time and reprocess all the data once again for some reason, they may not be able to catch up with the new data in due time and hence defy the whole purpose of stream processing.

Stream table duality

Stream table duality is one of the most talked topics in the stream processing domain. The idea is quite simple. However, it can take you a long way to conceptualize your stream processing solutions. The notion of stream table duality talks about the relationship between data streams and database tables, and how you can create one from the other.

When implementing stream processing solutions in practice, you need streams as well as the database tables. For example, when

you are processing an eCommerce transaction as an event stream, your stream processing application may want to join the transaction with the customer information from a database table. Streams are everywhere. However, the database tables are also everywhere. The point is straight. Tables are the reality, and you can't avoid them. Tables make perfect sense in a stream processing solution as well. So, you may want to combine the ideas from both worlds and design a practical solution.

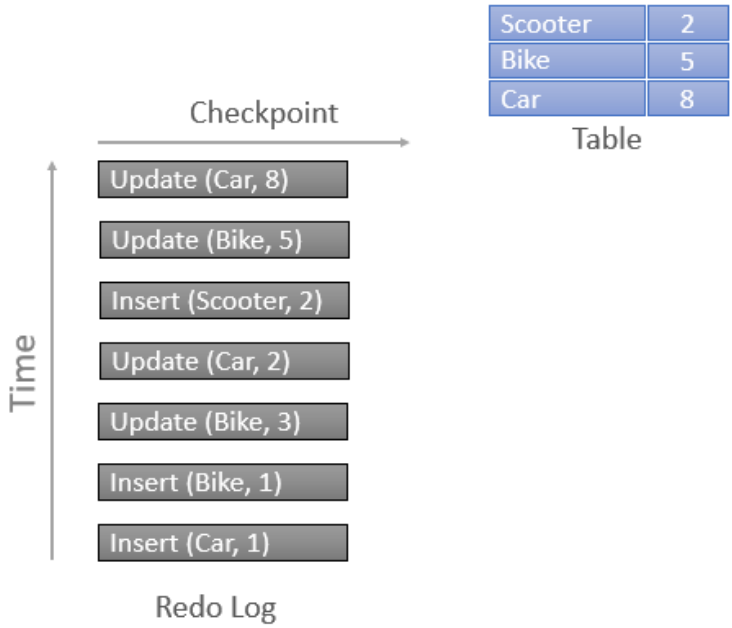

Figure 3.6 - Convert Stream to table

The relationship between stream and tables can be understood more clearly by realizing the fact that a table is nothing but an

aggregated view of a stream at a given point in time. This idea is applied by the databases to create tables.

In most of the databases, transactions are recorded in a log file in chronological order. The log file is a representation of a stream as it records immutable and continuous events. The databases read this log and apply them in a sequence to materialize the table. The table represents the latest state of each unique record however the log represents a sequence of transactions as they arrive. In fact, the database is converting a stream into a table.

On the other side, A table can also be converted into a stream. The idea is implemented by most of the change data capture (CDC) tools such as Oracle golden gate and HVR. These tools monitor all the changes to the table at one end of the pipeline and stream the changes to the other end of the pipe.

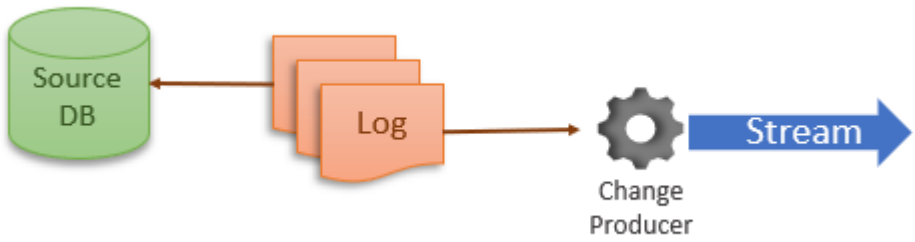

Figure 3.7 - Convert table to stream using CDC

The point is straightforward. The data exists in two related forms: streams and tables. Both are essential constituents of a practical solution.

Exactly once processing

The correctness of your results is of prime importance in most of the cases. However, some use cases are based on approximation, and a nearly accurate result is well accepted. The argument of accuracy and approximation are intensely discussed in the stream processing domain with three notions.

1. At least once processing
2. At most one processing
3. Exactly once processing

At-least-once processing talks about ensuring that we do not lose any event and each record is processed at least once. However, at the same time, at-least-once will leave a possibility for duplicate events. It accepts that an event is computed more than once but guarantees that none of the events are lost.

On the other hand, at-most-once processing focuses on avoiding duplicates. It guarantees that an event should not be processed more than once. However, it is acceptable to lose some of the events. In more casual terms, it means that you can miss some events, but you will never process it twice.

At-least-once and at-most-once are the two sides of an approximation. At-least-once approximates the value to the actual or more. For example, if you are counting clicks and the accurate result is 100, then the at-least-one approach will give you an answer as 100, or something more than 100. Since you chose to implement at-least-once and eliminated the possibility of losing any event, there is no chance of getting less than 100. However, the duplicate events may increase the count to something more than 100.

On the other side, at-most-once may give you 100 or something less than that as you accepted losing some events but ensured that you do not count any event twice.

At-least-once and at-most-once are about accepting tread-off depending on what makes more sense in your use case.

Exactly-once processing is ensuring that every event is processed exactly once without losing any event, and without duplicating anything as well. Exactly-once appears to be the best and the most desirable choice. However, implementing exactly-once is hard in a practical sense. One of the main problems is the delay in arriving records at the processor. You can’t keep waiting for the event forever, and it is almost impossible to ensure that the event reaches at your processor within an acceptable timeline or before the practically feasible watermark.

Some systems allow you to achieve exactly-once processing. However, that involves implementing database like transactional features to ensure that the transaction is successfully committed exactly-once. Implementing such transactions come at the cost of additional complexity, and that is why we often resolve to at-least-once, or at-most-once schematics whenever approximation is an acceptable thing.

Summary

In this concluding chapter of the first part of the book, we tried to clarify some common delusions that are often instigated by experience in batch processing systems. Then we moved into discussing some of the terminologies and core concepts that would be helpful for stream processing learning journey. Some of the central ideas of stream processing are the notion of time, windowing, and persistent state. The correctness vs. approximation is the other trade-off that we would have to deal in stream processing applications. This book will be talking about all these and many more concepts throughout the chapters. However, we wanted to set the stage to start learning stream programming.