EPISODE 520

[INTRODUCTION]

[0:00:00.3] JM: At a big enough scale, every software product produces lots of data. Whether you're building an advertising technology company, a social network or a system for IoT devices, you have thousands of events coming in at a fast pace and you want to aggregate them, you want to study them, you want to act upon those events.

For the last decade, engineers have been building systems to store and process these vast quantities of data. The first common technique was to store all of your data in HDFS, the Hadoop Distributed File System, and run nightly Hadoop map reduce jobs across that data. HDFS is cheap, because you're storing data on disk. It's effective, because Hadoop had this revolutionary effect on business analysis. It's easy to understand what was happening.

Every night you take all the data from the previous day, you analyze it with Hadoop and you send an e-mail report to all the analysts. Of course, the problem was that this was a batch system and you did not have the ability to react to events as they were coming in. You were just running nightly Hadoop jobs and your application or your business would get updated the next day.

The second common technique was the Lambda architecture. The Lambda architecture used a stream processing system like Apache Storm to process all incoming events as soon as they were created, so that software products could react quickly to the changes occurring in a large-scale system. Events would sometimes be processed out of order, whether we get lost due to node failures. To fix those errors, the nightly Hadoop map reduce jobs would still run and they would reconcile all the problems that might have occurred when the events were processed in the streaming system.

The Lambda architecture worked pretty well. Systems were becoming real-time and products like Twitter were starting to feel alive as they were able to rapidly process the massive volume of events on the fly. Managing a system with a Lambda architecture was painful. You had to manage both a Hadoop cluster and a Storm cluster. You had to make sure that your Hadoop processing did not interfere with your Storm processing.

Today, a newer technique for ingesting and reacting to data has become more common. It's referred to as streaming analytics. Streaming analytics is a strategy for performing fast analysis of data coming into a system. You only have to do it once, instead of the Lambda architecture model of having to do things twice.

In streaming analytic systems, events are set to a scalable, durable pub sub-systems, such as Apache Kafka. You can think of Kafka as this huge array of events that have occurred, such users liking tweets, or clicking on ads. Stream processing systems like Apache Flink or Apache Spark will read the data from Kafka as if they reading an array that is being continually appended to. This array, the sequence of events that get written to Kafka are called streams. This could be confusing, because with a stream you imagine this constantly moving transient sequence of data. It's just streaming by and you have to capture it.

That's partially true, but data will stay in Kafka as long as you want to. You can set a retention policy for two weeks, or two months, or two years, and as long as that data is still retained in Kafka, your stream processing system can start reading from any place in that stream. The stream processing systems like Flink or Spark that read from Kafka, they're still grabbing batches of data. They're processing them in batches. They're reading from the event stream buffer in Kafka, which you can think of as an array.

This is something that confused me for a very long time. I'm still wrapping my head around what exactly streaming means, versus batch processing. If you're still confused, don't worry, we explain it more in this episode.

Tugdual Grall is today's guest. He's an engineer with MapR. In today's episode, we explore use cases and architectural patterns for streaming analytics. Full disclosure, MapR is a sponsor of Software Engineering Daily. In past shows, we've covered data engineering in detail, and I'm looking forward to seeing more data engineering topics and presentations. I'm attending the Strata Data Conference in San Jose next month. If you're going to be there, please send me an e-mail. I'd love to see you there.

I love that Strata has been kind enough to give me a ticket. O'Reilly has given me tickets over the years, since before I had a big audience. They've always been very supportive of Software Engineering Daily, so in turn am very supportive of O'Reilly Media, which is just a fantastic company.

If you want to know more about other data engineering topics, you can check out our app where we have all 650 episodes of Software Engineering Daily in a searchable format. We've got episodes about streaming architecture, from companies like Uber. We talked to Matei Zaharia about the basics of Apache Spark. We've explored the history of Hadoop. You could find all these episodes, by downloading the app for iOS or Android, the Software Engineering Daily app.

We've got recommendations and categories, related links and discussions around the episodes. It's all free and it's also open source. If you're interested in getting involved in our open source community, we have lots of people working on the project and we do our best to be friendly and inviting to new people who are coming in and looking for their first open source project. You could find it at github.com/softwareengineeringdaily. We very much appreciate anybody who checks out the community, or joins our Slack. It would be great to see you there.

Thanks for listening and I hope you enjoy this episode.

[SPONSOR MESSAGE]

[0:06:03.4] JM: Apps today are built on a wide range of back ends, from traditional databases like PostgreSQL to MongoDB and Elasticsearch, to file systems like S3. When it comes to analytics, the diversity and scale of these formats makes delivering data science and BI workloads very challenging. Building data pipelines seems like a never-ending job, as each new analytical tool requires designing from scratch.

There's a new open source project called Dremio that is designed to simplify analytics on all these sources. It's also designed to handle some of the hard work, like scaling performance of analytical jobs. Dremio is the team behind Apache Arrow, a new standard for end-memory columnar data analytics.

Arrow has been adapted across dozens of projects, like Pandas, to improve the improve the performance of analytical workloads on CPUs and GPUs. It's free and open source. It's designed for everyone from your laptop, to clusters of over 1,000 nodes.

Check out Dremio today at dremio.com/sedaily. Dremio solved hard engineering problems to build their platform, and you can hear about how it works under the hood by checking out our interviews with Dremio's CTO Jacques Nadeau, as well as the CEO Tomer Shiran. At dremio.com/sedaily you can find all the necessary resources to get started with Dremio for free.

I'm really excited about Dremio. The shows we did about it were really technical and really interesting. If you like those episodes, or you like Dremio itself, be sure to tweet @dremiohq and let them know you heard about it from Software Engineering Daily.

Thanks again to Dremio and check it out at dremio.com/sedaily to learn more.

[INTERVIEW]

[0:08:03.9] JM: Tug Grall is a engineer at MapR. Tug, welcome to Software Engineering Daily.

[0:08:08.8] TG: Thank you.

[0:08:10.5] JM: I want to talk today about how to build streaming applications. I want to start from the top about what a streaming application is and why these are starting to become more and more popular in all kinds of industries. You've got all these different industries that create large volumes of streaming data. You've got factories, sensors that are strewn throughout agricultural field, you've got oil refineries and all these big industrial systems that gather all these types of data.

Of course, you've got adtech. You've got all kinds of real-world systems that are generating tons and tons of data. We have not had the tools to process those data streams in what some people might call real time. We'll get into what real time may or may not actually mean. We haven't really had the great tools to do that processing until recently.

Maybe you could give a little bit of history on what were the tools that were being used in the past, prior to the days when we had things like Kafka and Flink, what were people doing to process these vast quantities of data?

[0:09:18.9] TG: Yes. One of the very interesting point about streaming and streaming the data, because companies have data forever and they have been moving the data around batch mode before. The first part is before even talking about streaming is people are moving data, processing the data at scale or not, and you add a very good set of tool who is Hadoop on the big ecosystem to do that on a very large amount.

More people were asking that I want to be able to get the data and process the data as they are created, as they are edited by user, by systems. You are talking about factories on some sort. First thing that we have seen at scale, when I say at scale it's on a distributed systems that have to deal with million of events. It was part also of the Hadoop ecosystem with Storm. Storm has been very successful to get the data in real-time, but we had some challenges regarding the messaging part moving the data around.

This is where new systems came into the picture, because all system and messaging layer like GMS on the M2 series on all the message queueing platform were great to walk in a streaming part, where you are sending messages that's had a lot of issues to scale and to really provide a flexible architecture of all new applications.

[0:10:45.4] JM: I think one of the big architectural shifts that happened was people started to converge on patterns around Kafka. Before people started to do that, everybody knew, okay we got all these data that's being generated by people clicking on ads, or users doing various types of activity in systems where we just want to log their data, or factories, or oil refineries or whatever, you've got all these data.

The ingress process, there was not convergence around that ingress process. Maybe your data comes in and you're buffering it to a file and then you save the file to HDFS, the Hadoop

Distributed File System and then later on, you run some batch Hadoop job on it. That's a messy process. You're just reinventing the wheel every single time. It seems like recently, people are starting to really figure out, "Okay, how can we use Kafka to be the buffer between the generation of data and the recording of data into our data lake?"

[0:11:49.7] TG: Yeah. What you said about HDFS is correct. People were tweaking the system to transform something that has been built for batch to try to do a near real time. With MapR file system, we had a different approach, because you can save on just flush data as it goes. You were close to what you can do with Kafka in terms of accessing the data. As you mentioned, it's not only moving the data. It's also providing some interesting architecture choice on the way you want to build your application.

First of all, you want to do that with many different people at a different system will push the data into it. You are talking about ad industry with all the clicks and reviews. It will be dozens or hundreds of application server walking in parallel generating all the data. You push that into a Kafka product, a Kafka architecture, so you're walk in parallel with fail over, with distributed system to ingest and send totally [inaudible 0:12:51.7] way.

This is a very important part that you disconnect completely the source of the data on the potential targets, who will consume the data, as to be totally independent because you want to keep flexibility to add in new services, to add business processing or data processing.

You add this nice architecture that can ingest a lot of messages. It is saved into the platform, distributed or variable and saying you have consumer coming in to the game, where they have some interesting characteristic, like for example the scale is important. You can read in parallel the many processes. Also, you can have multiple type of application reading the same messages, reading the same event.

This is where Kafka has an interesting part also by providing an even plug, where you can replay if you want you can either just say, "I want to just subscribe on everything that is new when I start, or I want to wait, do and we wait and consume all the message from the beginning." This was not really possible before, not in an event manner. You had to wait a large amount of file with no context. Where in this case, you just have a very simple way of doing that.

[0:14:11.1] JM: You touched on something there, the segregation of reads and writes being pretty important. Is this the term CQRS, Command Query Responsibility Segregation, is that the pattern you're referring to?

[0:14:23.7] TG: Yeah. It's one of the pattern. Myself, I don't necessarily go in all these large architectures, because the reason why I don't and you know it's really your choice, because you will always have people – it's intellect when we talk about west full architecture. You will have different way of achieving that and people we never agree what we should do, why we should not do.

You are correct. You will have some part of the systems that is very responsible of pushing on writing the data. Another part of the system will come from query. Having something based purely on streams is you do that on the flow of the data coming in. You just process, you just come from them the way you want with different tools, because one of the key part is when you are asking about what is the big change, or what with previous architecture?

It's a fact that also there were industries, all the tools now will be compatible and using the Kafka infrastructure and even MapR we have built a tool called Map Stream that use the Kafka API to be able to leverage this ecosystem, in the way you published, on either way you consume the data.

[0:15:33.8] JM: I'd like to dive in a little deeper into what a stream is and how people then think about streaming, because this is a confusing abstraction for some people, because we think about – if you're just the application developer, then you think about request response. You're building a service-oriented architecture, or microservices architecture and you build a back-end that has some put request, get request, those are one-by-one requests.

The data engineering conversation is often about the streams, so you've got streams of data being created. How do we reconcile the idea of the fact that request response, like if I make a request to some endpoint somewhere, that is inherently a batch request. It's just one off request and I get a response, and that's one atomic request.

We talk about data in terms of streams, so that you have this mental image of the continues stream of data, going from one point to another. How do we reconcile those two ideas of data being passed from one place to another?

[0:16:44.2] TG: Yes. Usually when I talk about streaming and when I start a discussion about streaming architecture, or streaming processing and all this stuff, exactly the way you mentioned is ad developer, application developer will be very familiar with a request response. You put some data in, or you create some data, you have some processing, you have [inaudible 0:17:02.9], you have another where dealing with the data that would be the pure batch.

You have gigabyte of data into a database. You have gigabyte of data into the file system, or into a set of files and you have a job that's going on a regular basis to do some stuff; to calculate aggregations, or to do machine learning modeling and all of this. You have something in the middle that is in fact the most important in my opinion and the most exciting part is the streaming.

One way of doing, of looking at it is as you say it's a continues generation of data, a continues generation of event. In the simple way, I like to start is if you look at the technical log you have inside your application, so you have your web application, your web application is writing every time you have a click – every time you have somebody looking at some seeing a new application, you have a [inaudible 0:17:53.7] the log file.

If you do that the old way, old fashion of using this data will be you take the log file, you read the data and be aware of and this is a batch. In the same time, if instead of reading the file and instead of writing each click into a log file, you send it into a messaging layer. You send it into Kafka, each of the clicks become an event. The big benefits of that is you can say that I want to do something immediately when I receive the event.

Instead of waiting every hour, or having a request response as we have explicitly to say, I want to do this now, or every hour we'll have something running is I have an event that is generated by an application. It's somebody that click as logging to my application in terms of web, this generate an event. This event is now inside your system. If you didn't build anything, it's there and you will stay here and you can still process it in a batch mode. You don't do anything by doing streaming, you can still look at a batch being reading a subset, or a [inaudible 0:19:06.5] of events.

The interesting part is if you send my application, I want to react immediately when an event comes into the system. You don't run anything to your architecture, you say now every click is sent to Kafka, I will build a consumer, so I will be in a consuming application and this application every time if you subscribe to the specific topic, the specific set of event, every time the stuff is generated immediately a few millisecond after, I will receive the data.

What I do with it is different things. The streaming base it's moving the data, being able to publish the data in real time and consumes the data in real time. Then when I add the data, we will talk about what we do and stay in processing, it's when you get the data, you have to react.

It's a big difference compared with a great response on batch, because batch you control the amount of data you want to deal with, same with request response, where you walk with a streaming, you have your applications receiving one event, two event, three event, or maybe signs up an even one by one, but in a few seconds, or few minutes.

You have to cut a little bit differently do the stream processing to say, I want to detect if somebody, this specifically at rest is trying to login to my system multiple time in one second. Too many time in 30 seconds, this is probability of Had. The nice part about that, and this is a big, big difference in the one way of looking between batch and traditional architecture to streaming is instead of having a batch equivalent every two minutes, you will have immediately one login, two login, three login, three logins events, five of the six login event trying to connect to the system in less than 30 seconds, you can write a program that will say, "This is a [inaudible 0:20:58.1]. I have to react on this."

[SPONSOR MESSAGE]

[0:21:08.6] JM: Today's podcast is sponsored by Datadog, a cloud-scale monitoring platform for infrastructure and applications. Datadog's new container orchestration report, Kubernetes holds a 41% share of Docker environments, a number that's rising fast. As more companies adopt containers and turn to Kubernetes to manage those containers, they need a comprehensive monitoring platform that's built for dynamic modern infrastructure.

Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker, so that you can monitor your entire container infrastructure in one place. With Datadog's new live container view, you can see every container's health, resource consumption and running processes in real time.

See for yourself by starting a free trial and get a free Datadog T-shirt at softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog. Thank you Datadog.

[INTERVIEW CONTINUED]

[0:22:17.2] JM: That was a great explanation, great example as well and so much to dive into there. Tell me if this is a correct way of thinking about it. Instead of thinking of it as streaming, which you could use that term, thinking of the stream as a data structure; it's an abstraction. People are making requests to some backend microservices and those microservices are logging data to a stream, and that's a Kafka stream. You've just got this long stream of log events and it's like those logged events are a byproduct of the request response.

Then once you've got that long stream of events that is a byproduct, some of them are maybe going to be sitting in-memory on Kafka, some of them maybe have been written to disk on Kafka. Any case, you basically have this abstraction of a stream, but it is a stream that you can read from at any point. You can take a subset of it from any point, as long as those data points you're still on Kafka and typically your Kafka retention policy is pretty long, or it's reasonably long where you can say, "I want to read all the events that happened from two weeks ago to today and aggregate data on them."

Or you can say, "I want to just aggregate a sliding window of every three minutes." The whole idea is that you have all those data accessible to you and your performance characteristics are going to go up and down based off of how much of that stream you want to process at any given time, or how responsive you want to be to that processing of inbound events that are being written to that stream.

It's just an easier abstraction to work with to have this stream that is continuously being appended to, or you can start from any portion of it. It's just like a long array, where you've got essentially constant time access to any place in that array, rather than the previous model where you've got this huge data lake, everything is on disk. In order to do anything interesting with that data, you've got to spend a bunch of time pulling in from HDFS and putting it into memory and then starting to do a bunch of operations on it.

With the Kafka abstraction, we're writing everything to Kafka. Then it's going to be a more performant, more flexible, and it's already segregated by topics so you've already got essentially a schema across all of the actionable data in your organization.

[0:24:41.9] TG: Yeah, exactly. I think the most important part that you mentioned really is when you have people – emitting the event, so you have event one, two, three, four, five in this order. When you will read them by default, you will read by one, two, three, four, five, also doing any type of logic. Or you can just take the new messages and then what you said about segregation, to talk aggregation of the data, you will have any number of topics depending on the type of event, depending on the type of processing, or how you want to organize your data. It could be based on security and so on.

One of the matter on the architecture that you see, you will have even coming in home for example, factory generating many events on specific part. This event is quickly processed generating another set of different type of event. Let's take very simple example. You will lot of [inaudible 0:25:39.5] in the factory generating events one, two, three, four, five, for each – with a timestamp and submit the data information.

One of them is about for example, the temperature of one sensor. If the sensor is becoming to hurt, you will immediately generate a new event that will be now left on a different topic to be able to provide information to another type of services. You do that in a flow, in a real-time and with lot of, lot of flexibility in term of development.

You mentioned microservices before. Typically, this is where you will be able to quickly add new business logic, new processing, new services in the small amount of core, honing on its own. This is why it's very nice well doing microservices.

[0:26:29.8] JM: Yeah, you could call this enrichment, or adding a new materialized view. You gave example of a temperature data. You can also have – you've got a bunch of maybe I'm walking around and my Fitbit is periodically pinging the server and logging my location in some compressed GPS format. Then there's a data analyst sitting over at Fitbit, and they want to be able to analyze the data.

They want to deal with just GPS endpoints, or sorry, GPS data points. The would rather deal with like a location. You might have a small processing unit that takes those events and translates the GPS coordinates into something more rich, like Jeff is in Austin, Texas. He's walking at a pace of 3.2 miles per hour, and a bunch of other data that you could gather from just a GPS coordinate, because maybe it takes the GPS coordinate, takes the three previous GPS coordinates and calculates the velocity that I'm moving at.

That data can be much easier to act upon. What's interesting about Kafka is you could have two separate topics. You could have a topic for GPS events and you could have a topic for enriched location data. Basically, those are two materialized views of the same data, where you've got data coming in as GPS and then it gets enriched and then written back on to Kafka, so that Kafka becomes this really flexible accessibility layer for different types of data.

In that example, what would be the best type of application system – like if I want to write this thing that is just a translation layer that every time a GPS point is written onto that Kafka queue, I want to trigger an event that enriches it with location data and user velocity through space. Where would I write that? What kind of system would I write that in?

[0:28:29.0] TG: You have different options. The first will be the event layer that will be Apache Kafka on map streams, where we will be publish on consume information. This will give you access to many things. When you are a very simple case, we'll say this one is simple, you can maybe just use a simple job application, or scale application, or Python application that will read the message, transform the location into a different format with the velocity and emit using again the Kafka API to another topic.

This is something that you can still do writing very small and simple application in Java, or in Python, doing basic transformation of basic enrichment. It's one option in many, because sometimes you have some complex things to do like time windowing, because what you want to do, you want to do some complex calculation about the velocity, where you want to be sure to calculate your average speed on the sliding window every 5 seconds or something like that.

You start to think and look at that and you say, "It's becoming a little more complex to manage." Which type of tool I can use, because all the sample we are saying that working on very easy to do if you have let's say one event every second and you only have one or two users or to [inaudible 0:29:48.7] Fitbit in your applications.

Supposed you are really Fitbit and you have million of devices and you have distributed all over the planet and you have million of messages coming inside the system, it's a different picture. Kafka can handle that no issue. Your small level application, which will be a little complex because you have to add more applications. This is where you will bring specific stream processing layer. The two big things, or the three things we see on the market is Spark, with Spark streaming, Apache Flink and also Kafka streams.

All these will give you a set of libraries on API to simplify the development for doing a stream processing, really consuming one or multiple events. Take an action on inner where use case the action is two times from that using let's say a back streaming, or you choose to develop for example in scalar, you do some time windowing manipulations and you can emit another event into a new topic, and this topic will be the information that you send to as application.

[0:31:00.8] JM: That makes complete sense. If we've got this example of writing – let's take this example even further. We've got all these Fitbits of lots of different users and it's recording their GPS coordinates every 8 seconds or 5 seconds, and you want a streaming service that reads in those GPS coordinates and maybe it reads in a sliding window of those – the most recent 10 GPS coordinates and then calculates the user's current location in terms of where they – like the city they're in, maybe the country they're in and their velocity, because you've got the last 10 events.

What would be – if you've got tons and tons of users who are doing this at the same time and you've got a choice, you can use Kafka streams, you can use Spark streaming, you can use Apache Flink, I think there's a bunch of other ones. I mean, I know you mentioned just three, but there's also like Apache Apex.

[0:31:56.1] TG: Exactly. You have many of them.

[0:31:59.3] JM: Right. Then you've got Apache Beam, maybe we could talk about that a little bit later, which is like that API for unifying them or something. How do you choose between those different streaming systems?

[0:32:10.3] TG: This is probably the hardest question you asked in the beginning of this podcast. The reason why I'm saying it's hard, because they all have very good characteristics and so on. For example, we'll say my positioning today and the positioning we have at MapR is because we have a complete support of Spark – when we said the complete support of Spark, it's not only the APIs, we have expert internally in the company, a customer that is buying the product can connect and ask specific questions.

Because Spark today is the biggest community, I will say if you have to do some stream processing today and you are okay with developing in Java, or even better in the case of Spark with scalar, you can work with Spark. This is because I am walking on MapR and this is what I see the most, this will be my first choice based on supportability and community adoption.

Then if you look at this and you realize that, "Yes, but I need some more advanced capabilities in terms of time windowing and the type of window – the type of time that I can manage an event." One of the big benefits of Flink is it has many ways of dealing with time on states of the events where you want to save it between the phases of processing.

If you are very advanced, walk to do with the timestamp of the event depending on what you call the timeframe, the event when it has been created by the sensor, when it's come into the system, or when it's processed by the system. You have these options. In this case, Flink could be a better use case.

We can continue the discussion with for example Apex. When I will say that also you have the community itself. For example, I am based in Europe and in Europe Flink has bigger adoption than Apex. I don't know for the US, but for example here, I will say that except if you are a big contributor and a big actor on the Apex community in Europe we'll look first at Spark, then Flink and you'll have Kafka streams.

Kafka streams will be for a lot simpler use cases, but the benefits is that it's lightweight library compared with the other. In our story with Fitbits, I will go probably with Spark streaming first, because o the size.

[0:34:36.1] JM: Because of the size of the community.
[0:34:37.9] TG: Yes. Size of the community, but also the size of the data, the number of topics and the partitions and the way – the key that you will be done is just a size of the cluster that you will have to build will probably have some very lot of interest. Because what we discuss today in what we are discussing is really focusing on the streaming part light. But you have all the part of your system.

You may need to do some machine learning in parallel with the same data on streaming data or not. You may need to do some analytics on the data in streaming or not, because if we look at the pure streaming part, it simplifies a choice. You can say, "Oh, I just want to use Kafka/map streams and some simple processing in Java on Kafka stream."

Maybe I have all the part of the applications, where I have some data scientist that will be models and they are familiar with specific tools. This is why I was saying it's at equations, because you have to look at the full ecosystems not as a tool, but ecosystem as a number of application on developer, you having the company you are working with. Depending of the skills, do they have what we see a lot, a lot, a lot is a part of some of data engineers, data scientist will prefer to walk, with spy turn a lot.

This is where for example, the mix of Kafka and the Spark will be interesting, because me as a co-engineer, as a co-data engineer I will focus on moving the data, processing the data as fast as possible using Java and scalar. My colleagues that will be enrichment of the data with some specific model may use by turn, and in this case will use by turn on Spark.

We are using the same cluster. We are using the same on-time, on the same data set, new transformation of the data in the found processing layer. This is why I will say Spark and Spark streaming for this use case.

[0:36:34.0] JM: Okay. It sounds like, if we're talking about just that simple enrichment process, it's not that – you think Spark streaming is necessarily the best tool for that case. It's more that you think that maybe you want to have multiple different uses for your streaming system and like maybe Kafka streams, or MapR streams would be better for that simple enrichment process.

If you're going to be also doing some large scale machine learning across that materialized view that you're making with the user velocity for example, you don't want to have to context switch between different streaming frameworks. You would rather just write it all in Spark streaming. Do I understand it correctly?

[0:37:19.8] TG: Yes. It will depend on the maturity of the teams, the maturity of the enterprise regarding big data/streaming, because what we realize all the time is my statement saying I will push back for this reason is we'll say if it's a first big project people are doing with this type of architecture and data, because going forward, people would be more familiar with distributed system as a storage, or as a processing.

They may in turn use a new technology, or different technology without being afraid of mixing, without being afraid of okay, let's use back streaming and Spark and back streaming for the machine learning part. Let's use Flink to the other system, so within this case have to runtime.

I like to always try when I discuss with a developer and architects, or doing [inaudible 0:38:13.7] to say please keep it simple at the beginning and add complexity always when it's needed, because sometimes we just get to a little bit too excited on we build very complex system that are already complex based on the volume of information, based on the distribution of the data. In term of framework, if we can start simple and add complexity only when it's needed will be great.

[0:38:40.9] JM: Stream processing systems have this windowing process. That's a time period over which aggregations are made. What's the difference between a window and batch processing?

[0:38:56.3] TG: The batch processing will be I want to take all these data, or a subset of this data on this date to this date. Or let's talk about database that will store all the events the same way instead of being stored in Kafka map streams, it will be stored into a database. It takes the data from the 1st of January to today and do some processing. This is a batch part of it.

When you will work with streaming on windowing, it will be a little bit different, because the way you will do it, you will receive event and you have different options. You can say, every 10 event do this. This one will be event by event. Keep in mind that when in the first example I use a database, I will read all the record in one big batch.

Where here, when I say wait for 10 events, or wait for 10 seconds, or wait for the first event on the second event means that in the time window of 10, of 20 seconds between the two event, this is something that could happen in different times. It's a time window, we will wait for the 10 seconds, but if I said 10 events, so 10 events could come in 1 second. Or the 10 events could come in 2 minutes, or 10 days.

The fact that you work with a time window, account window of event, we'll give you a different way of processing the data. You will process the data exactly the amount of data you want based on your business need, instead of doing a big, big batch. You will first not only be more reactive, you need to react faster to the event, but you will also globally consume only the processing power and the time needed for the business you are trying to solve.

[SPONSOR MESSAGE]

[0:40:48.2] JM: Amazon Redshift powers the analytics of your business. Intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are choking on data. The problem could be with how you are managing your Redshift cluster.

Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the tool chain of everyone downstream from your data warehouse. The team at Intermix has seen so many redshift clusters, they are confident that they can solve whatever performance issues you are having.

Go to Intermix.io/sedaily to get a 30-day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your redshift logs and makes it easy to figure out what's wrong, so that you can take action, all in a nice intuitive dashboard.

The alternative is doing that yourself; running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time. Intermix is used by Postmates, Typeform, Udemy and other data teams who need insights to their redshift cluster.

Go to Intermix.io/sedaily to try out your free 30-day trial of Intermix and get your redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:42:34.0] JM: These stream processing systems are often doing multiple stages of data transforms. You might have a Spark streaming operation where you've got multiple stages of your pipeline, where you've got directly acyclic graph essentially of different operations that you're going to run over your data. It's going to go through the different stages in Spark streaming. These transformations take time and the stream processing system could fail during that period of time. You could have machine failures. That you have snapshotting. The systems may snapshot data to disk

How do the different streaming systems that we've been talking about, like the Spark streaming versus Flink, versus I don't know, Apex – how do the different streaming systems handle snapshot checkpointing? Is that one of the areas that they trade off against?

[0:43:27.7] TG: Yeah. You have to find ways, because for examples when we talk about as a streaming part, your stream processing layer of back streaming, Flink or Apex will subscribe to the Kafka topic and you will receive event, and you can choose also as you say start to processing with some events, start the processing and maybe as a failure.

First of all, you will have a first legal limit for your processing that will be when do I say it was a Kafka that I have read another message. When do I commit self that when I say, I have read this message and I have processed this message, because you can say until my full process, my food processing has been done, I don't want to say to Kafkas that I have read this message.

Exactly if I fail, I will reread the message – I will read all the message again and do the processing again. Then you have the snapshots of the state management into the different frameworks where each of them here will have their own way of dealing with – in which state of my processing on how I can – continue to run on different nodes. I personally have not looked in all the detail for the different frameworks here.

[0:44:40.7] JM: Okay. Understood.

[0:44:42.4] TG: A big part for me that when we talk about stream processing, is how do you interact based on the Kafka model where you read the message saying, "When do I say to Kafka that I have read and process this message?" Because a principle is just we read it fast, we want to say, "Yes, I have committed this upset. I have read on process. I know this message."

[0:45:04.2] JM: Talk more about that. Why is that an issue that you focus on?

[0:45:07.5] TG: It's not an issue. It's just a choice of design. To make it very efficient to process on distribute the read of the log of all the event, you can choose to not commit every time you read the message, because it will be a little bit more expensive to say, "I read this message. I am processing it. I read the second message, I'm processing it. I have to process the three message, [inaudible 0:45:28.4] two together," you can choose as they have to sign to commit for each message, or just sign when you have finished the job.

You should do that at each messages. The risk is if you're processing layer of fail and you don't have all the good snapshots and you don't know exactly what the state is, when the job will restart, you may not read the first message because you already said I have read this message.

If you choose to put a commit on the offset at the end of the processing layer, so if you have a failure when you write one message, two message, or start the processing and you fail without sending the commit to the Kafka layer, what will happen when it will restart, it will read again message one, mission two and continue the processing until such three arrive and do the job.

This is a type of thing that not only you do that when you do stream processing with Spark, or Flink or Apex, this is also the way you work when you build your own consumer writing in Java, or when you want to work with tools like, level toolings like talent stream set, that gives from a nice tooling for the developer they will use the same concept.

[0:46:35.6] JM: Understood. You work at MapR, which has a data platform that combines a lot of these technologies that we've been talking about. I think one of the goals is to make them a little bit more accessible and usable to organizations where maybe you don't have a giant fleet of distributed systems experts. What are the kinds of customers that you encounter and what kinds of systems do you want to build for them to make their lives easier?

[0:47:06.0] TG: I will say any application common to additional batch-oriented big data to a pure streaming, like we talk about. Even interactive applications, because we have a NoSQL database into a system. Main point of MapR, the wait has been build in addition to provide a very rich file system giving wait wide access to any file. You can start a small file like big file. It's not like in a HDFS where you are limited – not limited technically, but you don't try to save small files. In MapR, you can use any type of files. Most importantly, if you look at it on the different steps we have been talking, is you need a messaging or an event layer that is a Kafka cluster.

You need to save the Kafka somewhere, so you may need a file system based on Kafka, or this with the file system to make it more variable. You need a NoSQL engine sometimes to do some work on a distributed file system for very large file. What we have done with MapR, we put everything, all this into a single platform. We have MapR file system, MapR DB and MapR streams. MapR streams is using the Kafka API with a different way of storing the event. MapR DB is using the Hbase API with a different way of storing the data, and same with MapR DBG and we have a G on API.

This simplify a lot architecture of a system administrator, because you have a single cluster to put in place a single security model. Then at the top of the that, you use any open source project to build your application, like we talk about Spark. You still have all – lot of hive and map reduce job learning. You can run a fling job, you can many things at the top of this application. The type of enterprise is any type of companies, we have companies working on oil and gas industries, we have a bunk building their data lake in addition to transactions – listing for transactions, not part of pure financial stuff, but for regulations, to be able to check compliance in real-time. We have many, many type of application you can build with a simplified infrastructure.

[0:49:13.2] JM: Do they come to you with just data strewing throughout their system and unorganized fashion and they ask for help in getting it together and put – I think there's actually a lot of organizations out there who are – they're going through what might be termed a digital transformation. Maybe the first step is probably even getting their version control system up and running. They're not even on the stage of digital transformation where they're building a data lake, or doing sophisticated analytics. They're just moving off of the FTP server for example.

For us that are deeply involved in the tech industry, it might seem like, "Oh, everybody's got an HDFS cluster with all their data. It's just like a big backlog waiting for a data scientist to come in." There's actually a lot of organizations that do not even have that. How do you see these larger organizations getting started?

[0:50:08.6] TG: It really depends. You have people that come to MapR just because they want to verify a full storage. Some people will just want a better HDFS, better file system for big data. In this case, there are some maturity on a specific use case in view. A part of our job and the makings of community participating to events, try to get some interest, interaction, some discussing to find a first use case that will get people exciting by building new services.

It could be as simple as all these data sources and I want to be build a 360 degree view of something, 360 degree over my customer. In this case, we have different approach who can just every time our professional services are the partner ecosystem to build a step-by-step an application that will be this nice UI at the top of multiple data sources.

Then you start to talk about streaming and then you show that you can build processing and analytics in real-time at the top of that. You have many different stages. The easiest way is where people who have been walking with big data for years now they start to think, “I want to build real-time application.” I will come back to real-time in a minute. I think it’s important to mention.

They will say, “We need to move to streaming instead of batch,” that we have talked about. We need a NoSQL database to build an application that will react very quickly. This is easiest for us, because they have started the journey, but we have different hooks depending of the customer.

[0:51:39.7] JM: Okay. Real-time, what did you want to discuss with that?

[0:51:43.8] TG: Yeah. When we talk about real-time in big data and we are [inaudible 0:51:49.0] is we are not building a market that we have to send to another planet. We don’t have to react in – not yet, yes. What we call real-time in the project we are building and helping customer build this real-time business point of view. It could be a few milliseconds, a few seconds, or few minutes.

An example will be I am in a mall looking in different shops. What I want in real-time as an application is I want when my mobile phone is recognized on the Wi-Fi network of the mall. I need to send some specific promotion, some specific frontend with my user in less than 1 minute. In this case, this will be real-time and it’s good enough.

Another use case is I am using the platform to monitor all my applications, so I have Kafka map streams to list in all the logs we are talking about. When I start to see somebody trying to do it for in term of logging, or an application that's becoming too slow, I have to react immediately. In this case, it's a few milliseconds to send an alert to another system. This is real-time.

[0:52:59.6] JM: Right. It's a variable amount of time. It is real-time depending on how low you need the latency to be.

[0:53:08.5] TG: Yes. With all the framework we have today, we can adjust to that.

[0:53:12.7] JM: Yeah. Well, let's begin to wrap-up by talking a little bit about the future and maybe some of your opinions. What are the unsolved problems in the streaming systems? What are the canonical problems that you're seeing, where something just doesn't seem right? Like something is a little bit too hard and maybe we're going to see some kind of solution or startup, or open source project in the near future?

[0:53:38.6] TG: I think what we see quickly is all the security, how do you protect the data from the stream, and how do you make them available to specific set of users? MapR started to walk on this. This is for me, the first part is big step things that you have to solve. For me the biggest challenge that we see today is company start with very complex deployment in term of the deployed in their own data, sometimes deployed in multiple platform, either a multiple geos all over the world depending of the application.

We work for even pull with the ad management platform that have multiple data sometimes in the US, multiple data sometimes in Europe and Asia. They have to synchronize all the streaming and real-time between all this. This is from your big place where we have to continue to improve the platform to make that easy for the developers and the architect.

[0:54:35.7] JM: Last question. I don't know if you'll have a good answer to this, but we did a bunch of shows recently about these serverless tools, basically on demand compute where you don't have servers running continuously. You have on-demand uptime that scales up or down based off of certain workloads.

I've had conversations with people about how this can be used with Kafka. Have you thought much about the opportunities for serverless, the on-demand computing model for how that could impact streaming?

[0:55:08.6] TG: Yeah. The thing is what we are working on it's not necessarily purely the serverless. What I see and I see we should walk on that we started is not only the pure serverless, like the big provider providing, but more at least to use a content analyzation, docker and Kubernetes to provide processing on the fly.

The pure servless part will be adaptable of that, because what will happen is every time you will sum the messages all about the place, you will have the application reacting to that. Yes, I think it's one of the most very exciting part of the new application we see building. That all the stage for many enterprise. The first investment we do on the enterprise are doing that we see, at least that I see is really first about content analyzation of the applications and elasticity around processing of consuming the different events.

[0:56:03.7] JM: Okay, Tug. Well, it's been great talking to you. We covered a lot of stuff here. Thanks for coming on Software Engineering Daily.

[0:56:09.8] TG: Thank you.

[END OF INTERVIEW]

[0:56:13.2] JM: If you are building a product for software engineers, or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an e-mail jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers, because I talk to them all the time. I hear from CTOs, CEOs, Directors of engineering who listen to the show regularly. I also hear about many newer, hungry software engineers who are looking to level up quickly and prove themselves.

To find out more about sponsoring the show, you can send me an e-mail or tell your marketing director to send me an e-mail jeff@softwareengineeringdaily.com. If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company.

Send me an e-mail at jeff@softwareengineeringdaily.com. Thank you.

[END]