WHITEPAPER

The Evolution of Smart Stream Processing Architecture

Enhancing the Kappa Architecture for Stateful Streams

Background

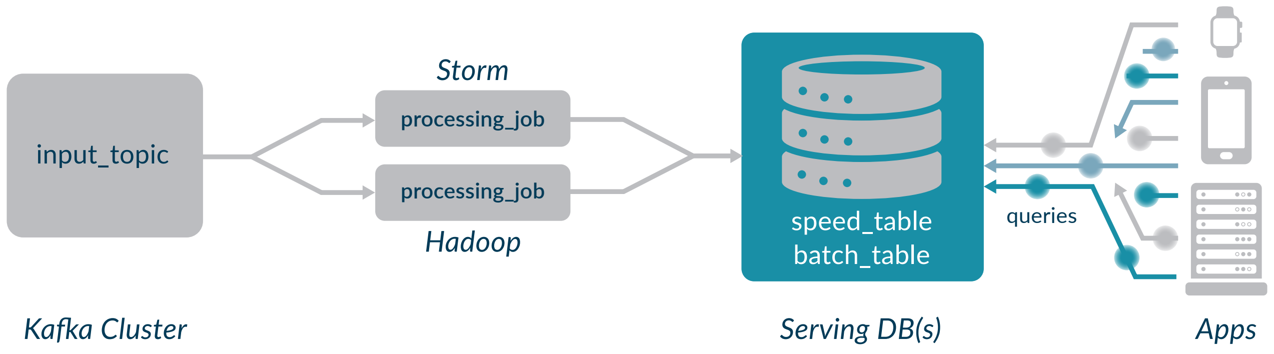
Real world streaming use cases are pushing the boundaries of traditional data analytics. Up until now, the shift from bounded to unbounded was occurring at a much slower pace. 5G, however, is promising to revolutionize the connectivity of not just people but also things such as industrial machinery, residential appliances, medical devices, utility meters and more. By the year 2021, **Gartner estimates** that a total of 25.1 billion edge devices will be installed worldwide. With sub-millisecond latency guarantees and over 1MM devices/km2 stipulated in the 5G standard, companies are being forced to operate in a hyper-connected world. The capabilities of 5G will enable a new set of real-time applications that will now leverage the low latency ubiquitous networking to create many new use cases around Industrial & Medical IoT, AR/VR, immersive gaming and more.

All this will put a massive load on traditional data management architectures. The primary reason for the break in existing architecture patterns stems from the fact that we are moving from managing and analyzing bounded data sets, to processing and making decisions on unbounded data streams. Typically, bound data has a known ending point and is relatively fixed. An example of bounded data would be subscriber account data at a Communications Service Provider (CSP). Analyzing and acting on bounded datasets is relatively easy. Traditional relational databases can solve most bounded data problems. On the other hand, unbound data is infinite and not always sequential. For example, monitoring temperature and pressure at various points in a manufacturing facility is unbounded data. This data is constantly being generated and must be analyzed in real-time for predictive maintenance.

As we move from bounded to unbounded data problems, Smart Stream Processing can be broadly considered as a continuous data processing system. This requires the ability to intelligently analyze the data streams on the fly, draw meaningful insights and take actions continuously, eliminating the need to store and subsequently query the data in a downstream serving database.

Lambda Proven Not to Work in Practice

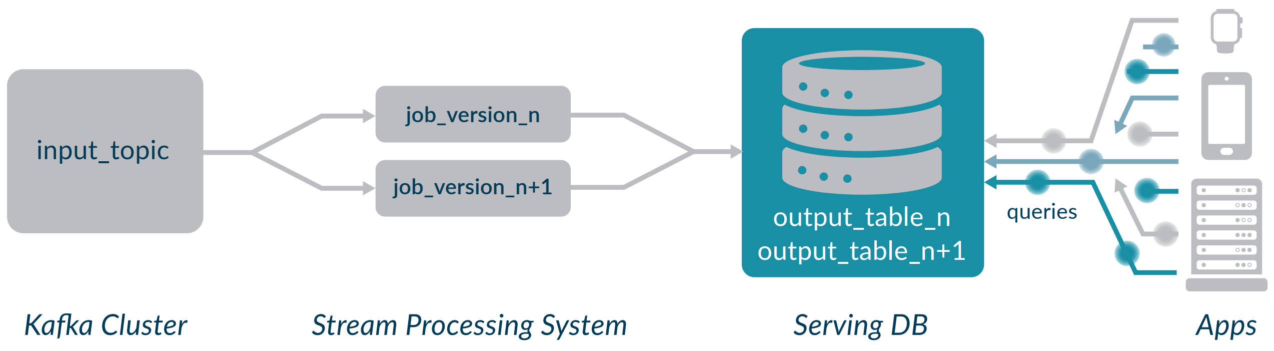
There have been numerous architectures floated over the last decade to handle large volumes of data. The Lambda Architecture was introduced to take advantage of both batch and stream processing methods. This approach attempted to balance latency, throughput and fault-tolerance by utilizing batch processing (e.g., Apache Hadoop) to provide accurate, comprehensive views, while simultaneously using real-time stream processing (e.g., Apache Storm, Apache Heron, Apache HBase) to provide visibility on in-the-moment data.



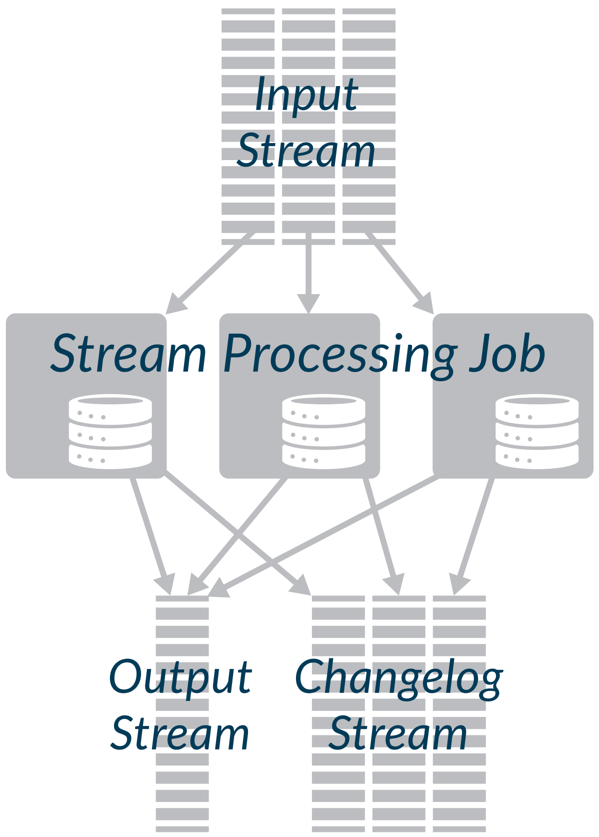
While good in theory, this approach was quite problematic as it involved having multiple code bases to produce the same result in two complex distributed systems. Multiple code bases in reality quickly became a maintenance nightmare. Programming in disparate, distributed frameworks like Apache Storm, Apache Heron and Apache Hadoop is complex. The code ends up being specifically engineered to run on the specific framework it's designed to run on, requiring reimplementation and maintenance using multiple technologies and languages. This results in massive operational complexity that is difficult to overcome.

Beyond Kappa for Next-Generation Streaming

Kappa Architecture uses a combination of stream ingestion from a messaging service like Apache Kafka, stream processing from Apache Spark or Apache Flink, and a datastore that's used as a serving database for output tables. The Kappa Architecture, however, falls short of addressing the demands of next-generation applications in the 5G era.



While traditional Kappa worked well for stream processing that required basic pattern recognition on incoming streams, it did not allow for querying with reference to state. Bringing in contextual state to stream processing allows for richer analysis. This was addressed contemporaneously by updates to the stream processing layer (The Apache Samza project) that uses a local embedded key-value database. It further leverages Apache Kafka's log compaction mechanism to make the state resilient against machine failures. This addresses the problem of contextual state for processing streams as it cuts down on round-trip network latency required to go to a remote database. This is called "turning the database inside-out" in Apache Samza literature.



Brilliant! Isn't it? Well, not so fast! While embedding a DB in your processing layer sounds like a good idea, in reality, it's like adding an egg to the cake after its baked.

While Apache Kappa may work fine for generic stream processing with some state, it is not designed for next-generation applications that require high volume, latency bound, complex workloads with ACID transactions. Next-generation applications can be classified by three essential elements:

1. Increased volume of data.
2. Short timeframe for response.
3. Complexity of decision making that requires streaming data analysis along with stateful ACID transactions.

Let's take the example of Telco Fraud: CSPs have had to combat hundreds of fraudulent schemes where scammers cheat the operators of billions of dollars by fake phone calls. The CSPs want to be able to prevent such fraudulent calls while still allowing for genuine user calls to get through. This problem has all three of the above elements. There is an increasing number of calls where someone is waiting for the call to be connected, and the CSP needs to run complex fraud prevention algorithms in-event to detect -and prevent — fraud from occurring. Based on the result of such algorithms being run in real-time, they subsequently ping the mobile device to make sure it's not a fraudster's computer. All of this must be achieved in under five milliseconds.

Even with an embedded key-value store, Kappa is not sufficient to address such use cases because of a variety of reasons:

- It does not offer native ACID transactions required for dealing with exactly once semantics. Workarounds to add exactly once semantics lead to a significant compromise on performance.
- While efficient on network latency, they still cannot meet the five-millisecond latency bound response.
- While simpler than a remote DB, there are still multiple operational issues with disparate tools glued together, such as failure/disaster tolerance, recovery, ACID state, and auto scaling.

As data volume goes up, decision complexity that combines stream and ACID state becomes higher and the time-window is mere milliseconds. In this time; you have to marry the incoming event stream with contextual information and apply complex supervised machine learning models to take optimal business decisions.

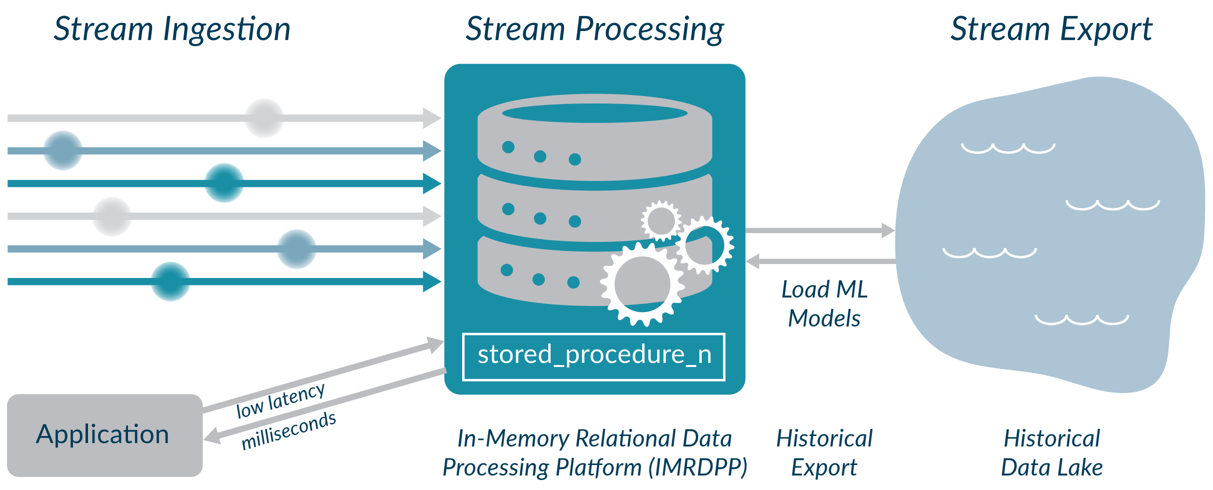
Further, as you look towards continuous data processing, it's not unthinkable to have self-learning Smart Stream Processing systems. Systems that can apply unsupervised machine learning training, scoring, validating and acting in a closed loop process on streaming data, eliminating the need for a downstream data warehouse.

5G Requirements

Are the "high velocity, latency bound complex workloads" common in the real world? The foundation of 5G is based upon these attributes. 5G will push the envelope on all three of the dimensions detailed above. Over the next few years, we will witness hundreds of billions of connected endpoints creating a vast deluge of data. But while network capacity is increasing with 5G, network latency won't be able to compete with the speed of processing locally near the edge, as network latency is ultimately subject to, at best, a fraction of the speed of light. This will put state closer to the edge by distributing what was monolithic and/or centralized before. Next-generation use cases in the 5G era will involve streams of transactional data for mission critical applications whereby someone or something critical is waiting for a response on the analysis to take a decision and act on it. Lastly, complexity of analysis and decisions has expanded beyond measure — from a handful of simple rules of thumb, to machine learning-driven dynamic rules involving hundreds to thousands of conditions/constraints that are calculated in real-time. Running these models in real-time is the only way we can move from reconciliatory analytics to an inline application of predictive & prescriptive analytics while adding millions of dollars to the bottom line.

Smart Stream Processing Architecture

Can Kappa Architecture address the needs of next-generation stream processing applications powered by 5G? The only way to address the massive potential of 5G-powered applications is to improve beyond the original Kappa Architecture. Let's call it Smart Stream Processing. Instead of bringing data closer to the processing layer, a much simpler approach is to perform the processing right where the data resides. For high velocity streams and latency bound applications, traditional DBMS systems are just not fast enough. This led to Kappa "turning the database inside-out" and embedding state within the processing layer. However, if you use an ACID compliant, In-Memory Relational Data Processing Platform (IMRDPP), ingesting, processing and storing can all be done in a single layer. By having accurate contextual state of your entire dataset, enriching it, analyzing it and taking real-time decisions makes for a Smart Stream Processing Architecture.



Using an IMRDPP eliminates the needs for Apache Kafka for ingest, Apache Flink or Apache Samza for data processing and the additional embedded key-value DB layer.

Using an IMRDPP eliminates the needs for Apache Kafka for ingest, Apache Flink or Apache Samza for data processing and the additional embedded key-value DB layer.

Often, Apache Kafka is used as a real-time glue for ingestion, simply because of fast ingestion and durability for speed mismatches downstream. An IMRDPP system, however, is fast enough to ingest streaming data and offers durability built-in. This eliminates the need for an Apache Kafka layer. The IMRDPP stores data in a relational format, therefore a separate serving database isn't required either. Finally, an IMRDPP system has Complex Event Processing (CEP) capabilities built-in, such as aggregations, filtering, sampling, and correlations. Additionally, SQL statements stored as stored procedures within the IMRDPP allow for embedding Java logic next to the stream, enabling real-time decision making. This also prevents an elongated Round-Trip Time (RTT) needed to query state in a remote serving database that might reside in a far-off central location or in the cloud. Embedding a key-value database in the stream processor falls short of an IMRDPP, as it does not offer millisecond response times, strict ACID, CEP, and high performance at linear scale.

You might ask, how much can an IMRDPP scale? An IMRDPP system can scale to terabytes of capacity while allowing for storage of all data necessary for real-time analysis at the edge. There could be multiple IMRDPP clusters distributed at various aggregation points in the edge that are connected to one another with active-active replication. There are some use cases where only a small subset of edge data is "hot", while other use cases may require a cap on the amount of memory allocated to an IMRDPP for cost reasons; in these scenarios, only the hot data remains in the IMRDPP. Everything else gets drained out via a historical export into the data warehouse or data lake of your choice at a central location. Note that such data warehouses / lakes are not a part of the Smart Stream Processing Architecture.

Data warehouses / lakes can also be used to train supervised Machine Learning (ML) models. These ML models can be re-imported as Java-based stored procedures into the IMRDPP multiple times per minute — or as necessary — to ensure that the ML model is always fresh, relevant, and operational.

Smart Stream Processing Requirements

The Smart Stream Processing Architecture utilizes an IMRDPP system to improve upon the traditional Kappa architecture. The IMRDPP system must be able to fulfill all of the requirements of next-generation applications:

- **Performance & Scale:** The ability to process terabytes of stateful streaming data within milliseconds.
- **Analysis & Decision Making:** This includes a wide variety of requirements from queuing, to data enrichment on the simple side, to being able to embed complex ML models on an incoming stream of information. The IMRDPP must have features for:
  - Data enrichment & transformation — Aggregates & windows.
  - ACID transactions — The ability to provide accurate information all the time.
  - ANSI SQL — Support SQL-compliant interactions with the data.
  - Automatic data migration to downstream systems for historical analytics and cold storage.
  - Embedded ML — Embed pretrained models in live streams to take smart decisions in real-time.

- **Stream Management & Monitoring:** To deploy Smart Stream Processing in mission-critical applications, you must have:
  - Functions like pause, resume and skip records in case of failure.
  - Ability to publish and subscribe to different data streams.
  - Durability features like Command Log/Snapshots to persist a power loss or failure.
  - High availability features that can tolerate node failures.
  - Replication features for disaster recovery and having a distributed mesh of clusters with active-active replication between them.
  - Detailed monitoring features that allow for the collection of statistics to make sure everything is up and running.

Benefits of Smart Stream Processing

Introducing real-time analysis and intelligence (i.e. Smart) to Stream Processing has many benefits:

- **Low predictable latency** — Removal of two or more disparate tools cuts down on response times.
- **Simpler development and testing** — An ACID-compliant SQL IMRDPP eliminates the need to write additional code to account for consistency in the application layer.
- **Much lower CAPEX / Hardware footprint** — Just to stand up an Apache Kafka, Apache Flink and a NoSQL DBMS requires 3x3 node clusters. Having a 3 node IMRDPP cluster instead, reduces the CAPEX down by at least 3x. The cost savings are even more magnified at scale.
- **Significantly reduced OPEX** — Just imagine the number of failure scenarios on a network partition between three distributed technologies like Apache Kafka, Apache Flink and a NoSQL DB? A unified IMRDPP is much more failure tolerant and easier to manage.
- **Lower TCO** — Ultimately, all of the the benefits listed above result in at least a 3x reduction in TCO.

Selected Use Cases

Historically, stateless streaming applications leveraged Kappa and did filtering at the edge while performing analytics in the cloud. Although this approach was bandwidth-efficient, it did nothing to cut down on the RTT latency, which is essential to make business decisions on time. Here are some of the next-generation Smart Stream Processing applications that 5G is enabling:

- **Smart Meter Billing** — Smart Grid Operators are leveraging Smart Stream Processing to do real-time billing for smart meter subscribers on two islands in Japan. The goal is to effectively determine the usage in real-time so that subscribers can effectively manage their consumption based on usage.
- **Predictive Maintenance in Industrial IOT (IIoT)** — In IIoT, Smart Stream Processing is used to record the time-based measurements of parameters and in-event decisions to keep the machinery operational.

- **Fraud Prevention** — Both finance and telco offer numerous fraud prevention examples like credit card fraud or international call fraud that get detected and prevented in real-time by leveraging Smart Stream Processing.
- **Real-Time Personalization and Offers** — Real-time personalization leverages Smart Stream Processing to serve up real-time offers for phone plan users just-in-time / before their prepaid plan expires.

Summary

Next-generation applications powered by 5G require real-time analysis and decisioning on streaming data at the edge. Existing streaming architectures rely on disparate embedded technologies to ingest, process, and store data. They try to bring the data closer to the processing layer. This results in much higher latency, lack of scalability, the inability to handle the complex workloads, and hardware sprawl. These architectures fall short of addressing the CEP, data accuracy, low latency and linear scale demands of today's market.

The Smart Stream Processing Architecture consists of one unified environment for ingestion, processing, and storage. This integrated approach with built-in intelligence performs the analysis right where the data resides. It utilizes a blazing fast IMRDPP to not just make streaming "smart", but also provides linear scale, predictable low latency, strict ACID, and a much lower hardware footprint that can easily be deployed at the edge.

With built-in analytical capabilities such as aggregations, filtering, sampling and correlation — along with stored procedures / embedded supervised and unsupervised Machine Learning — all the essentials of real-time decision-oriented stream processing are available in one integrated platform.

11 April 2019