Spark Structured Streaming Helps Smart Manufacturing

Xiaochang Wu
xiaochang.wu@intel.com

Hao Cheng
hao.cheng@intel.com

Intel Big Data Engineering Team

#EUstr1

Who am I?

- Currently working at Intel Big Data Engineering team based in Shanghai, China
- Joined Intel from 2006, focus on performance optimization for CPU / GPU / System / App for ~10 years
- Delivering the best Spark performance on Intel platforms
- Building reference big data platform for Intel customers

Agenda

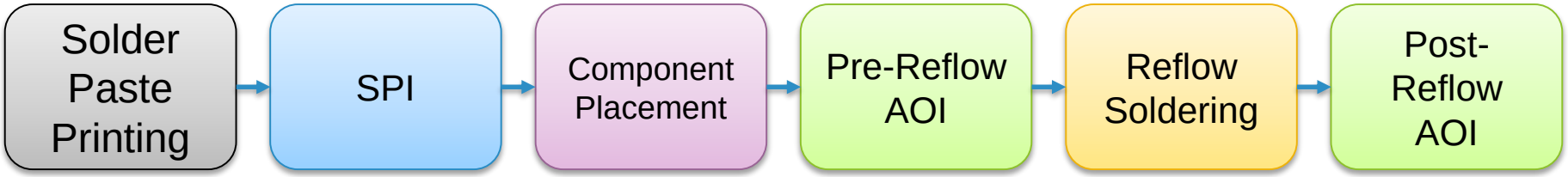
- Motivation
- Architecture and Features Overview
- Adopting Spark Structured Streaming
- Improvement Proposals

Motivation

- Building Spark-centric IOT Data Platform for PCB Manufacturing
  - Data collection, processing, analytics, visualization and alerting
  - Short-term goal: Enhance predictive fault repair and material tracking efficiency
- Take advantage of latest Structured Streaming
  - Streaming ETL
  - Stateless and Stateful processing
- Learn from real-world problems

Overview

- Transform the PCB surface mount lines

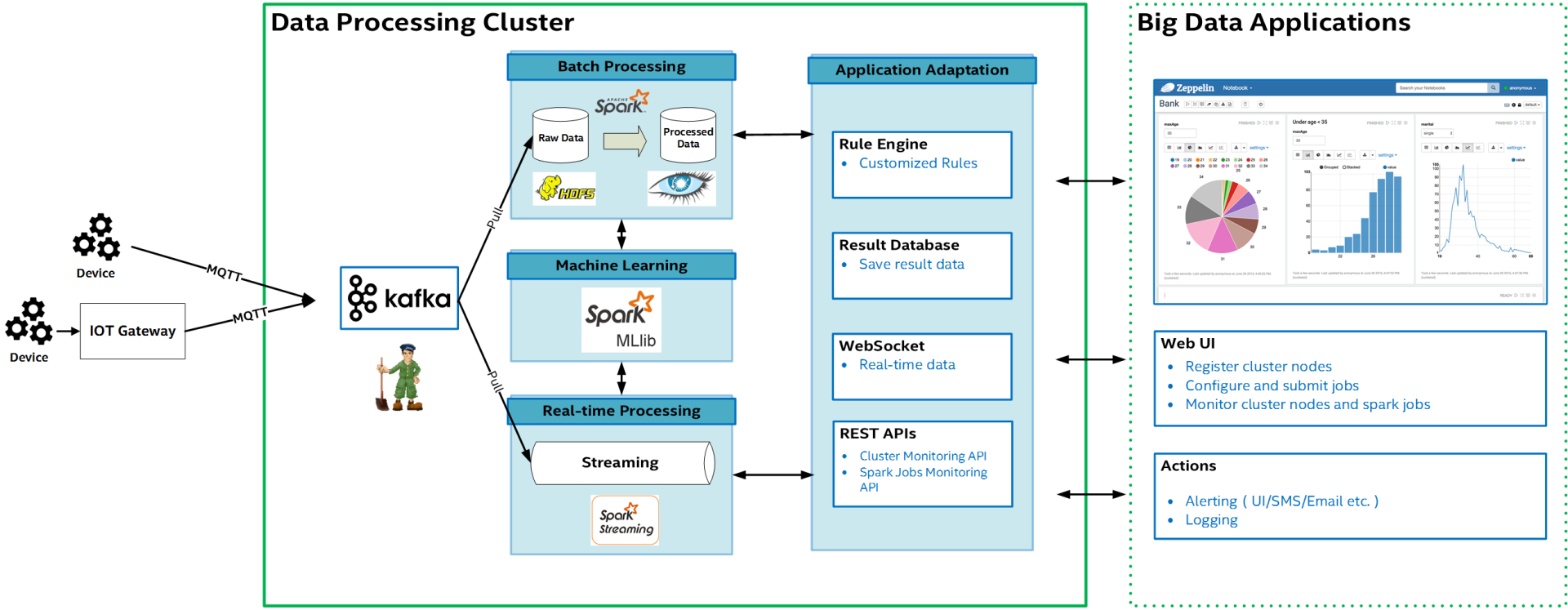


- SPI: Solder Paste Inspection
- AOI: Automated Optical Inspection

Data Platform Features

- Device Management
  - Register and manage devices
  - RPC commands to access device attributes
- Data Collection
  - Collect and store device data in reliable way
- Data Processing & Analytics
  - Domain-specific scripts for both batch and real-time processing
  - Support using Machine Learning algorithms for advanced analytics
- Rule Engine & Actions
  - Process incoming data with flexible user-defined rules.
  - Trigger alarms or send commands to devices using rules.
- Data Visualization
  - Customizable user-friendly UI & Dashboards
  - Generate interactive data reports to fulfil business needs
- Multi-tenancy, Scale-out, Fault-tolerance & Security

Data Platform Architecture



*Other names and brands may be claimed as the property of others.

Data Characteristics

- Data Sources
  - Manufacturing Execution System (MES)
  - Sensor, Camera, Log
  - OCR for machine screen output
- Structured, semi-structured and unstructured data
  - Low frequency but large amount
    - E.g. Picture and video
  - High frequency continuous data
    - Not large amount per unit. Total amount is very large
    - E.g. vibration sensor data used to detect the quality of the equipment.

Spark Structured Streaming

- Built on the Spark SQL engine
  - Express streaming the same way as batch
  - Dataset/DataFrame API in Scala, Java, Python or R
- Incrementally and continuously updating the final result
  - Handling Event-time and Late Data
  - Delivering end-to-end exactly-once fault tolerance semantics
  - Output modes: Append, Complete, Update mode
- Support Stateless and Stateful operations

Fast, scalable, fault-tolerant, end-to-end exactly-once

Streaming ETL

- Data are produced from sources as JSON objects, transferred with MQTT and saved to Kafka
- JSON objects are complex and have nested structures
- Use Spark SQL functions to extract
  - *selectExpr, get_json_object, cast, alias*

```
val ds = dataFrame.selectExpr("CAST(value AS STRING)").as[String]
val parsedDF = ds.select(
  get_json_object($"value", "$.key").alias("key"),
  get_json_object($"value", "$.idno").cast("int").alias("idno"),
  get_json_object($"value", "$.ts").cast("timestamp").alias("ts"),
  get_json_object($"value", "$.OffsetX").cast("double").alias("OffsetX"),
  get_json_object($"value", "$.OffsetY").cast("double").alias("OffsetY")
)
```

Stateless and Stateful Operations

- Stateless
  - Projection, Filter
- Stateful (Implicit)
  - Aggregation
  - Time window
  - Join
- Advanced arbitrary stateful operations
  - mapGroupsWithState / flatMapGroupsWithState

Structured Streaming Considerations

- Output mode: **append, update, complete**
- Handling late data
  - Method 1: use time window and automatic watermark
  - Method 2: use manual watermark
- Post-processing
  - Debug: console sink / memory sink
  - File sink / Foreach sink
  - Collect to driver side and process

Case 1: Find out offset AVG and STDDEV in Solder Paste Inspection (SPI)

```
val resultDS = dataFrame
  .groupBy("idno")
  .agg(mean("OffsetX").as("meanX"),
       mean("OffsetY").as("meanY"),
       stddev("OffsetX").as("stddevX"),
       stddev("OffsetY").as("stddevY"),
       count("idno").as("recordsCount"))
  .filter($"idno".isNotNull)
  .orderBy("idno")
  .as[ResultRecord]
```

Case 2: Find out machine off time from event logs (1)

- Track sessions from streams of events and find out machine off time
  - mapGroupsWithState / flatMapGroupsWithState
  - Off time = Duration from status “off” to next “on”

```
{"device_id":1, "timestamp":"2017-01-07 15:05:00", "status":"off"}
{"device_id":2, "timestamp":"2017-01-07 15:02:00", "status":"on" }
{"device_id":3, "timestamp":"2017-01-07 15:04:00", "status":"off"}
{"device_id":1, "timestamp":"2017-01-07 15:02:00", "status":"on" }
{"device_id":1, "timestamp":"2017-01-07 15:01:00", "status":"on" }
{"device_id":3, "timestamp":"2017-01-07 15:03:00", "status":"off"}
{"device_id":2, "timestamp":"2017-01-07 15:03:00", "status":"on" }
{"device_id":2, "timestamp":"2017-01-07 15:04:00", "status":"off"}
{"device_id":2, "timestamp":"2017-01-07 15:08:00", "status":"on" }
{"device_id":3, "timestamp":"2017-01-07 15:10:00", "status":"off"}
```

Case 2: Find out machine off time from event logs (2)

- Event: define data types for events
- SessionInfo: Session information as state
- SessionUpdate: Update information returned as result

```
case class Event(device_id: Long, timestamp: Timestamp, machine_status: String)

case class SessionInfo(offTime: Long,            // Total machine off time so far
                       currentTime: Timestamp,   // Latest event timestamp from last update
                       currentStatus: String,    // Latest machine status from last update: "on" or "off"
                       eventList: List[Event])   // Current event list

case class SessionUpdate(deviceId: Long,    // Device ID
                         offTime: Long,     // Total machine off time
                         expired:Boolean)   // If session is expired
```

```scala
val sessionUpdate = iotEvents
  .groupByKey(event => event.device_id)
  .mapGroupsWithState[SessionInfo, SessionUpdate](GroupStateTimeout.ProcessingTimeTimeout) {
    case (deviceId: Long, events: Iterator[Event], state: GroupState[SessionInfo]) =>
      // If timed out, then remove session and send final update
      if (state.hasTimedOut) {
        val finalUpdate =
          SessionUpdate(deviceId, state.get.offTime, expired = true)
        state.remove()
        finalUpdate
      } else {
        val eventList = events.toList

        val updatedSessionInfo = if (state.exists) { // update existing state
          // Get existing state
          val oldSession = state.get
          // Update and sort the event list based on timestamp
          val eventList_new = (oldSession.eventList ::: eventList).sortWith(_.event_time.getTime < _.event_time.getTime)

          //
          // Iterate thru event list to calculate off time, then update to updatedSessionInfo
          //

        } else { // setup inital state
          // Sort the event list based on timestamp
          val eventList_new = eventList.sortWith(_.event_time.getTime < _.event_time.getTime)

          //
          // Iterate thru event list to calculate off time, then set inital value to updatedSessionInfo
          //

        } // end if (state.exists)
        state.update(updatedSession)
        // Set timeout such that the session will be expired if no data received for 5 minutes
        state.setTimeoutDuration("5 minutes")
        SessionUpdate(deviceId, state.get.offTime, expired = false)
      } // end if (state.hasTimedOut)
  } // end mapGroupsWithState
```

Case 2: Find out machine off time from event logs (3)

- Event time ordering
  - Every subsequent event depends on previous event
  - Required to sort events based on timestamp due to logical dependence

- Handling late events
  - Late data will break previous calculation result, forced to maintain all events and recalculate
  - Solution: manual watermarking for late data, remove old events if timeout, incremental calculation

Improvement Proposals

- Flexible window and trigger operation
- Complex Event Processing (CEP) APIs
  - Stream Join
  - Session Tracking
  - Pattern
  - Alert
- Streaming Domain Specific Language (DSL)
  - SQL-like language for Streams
- We are working on them!

See Also from Intel team

- Today 14:00 @ AUDITORIUM

An Adaptive Execution Engine For Apache Spark SQL

by Carson Wang

- Tomorrow 11:00 @ LIFFEY HALL

FPGA-Based Acceleration Architecture for Spark SQL

by Qi Xie

Legal Notices and Disclaimers

- You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

- Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

References

- Surface mount process: http://www.surfacemountprocess.com/
- Structured Streaming Programming Guide: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
- Five Spark SQL Utility Functions to Extract and Explore Complex Data Types: https://databricks.com/blog/2017/06/13/five-spark-sql-utility-functions-extract-explore-complex-data-types.html
- Spark Structured Sessionization example: https://github.com/apache/spark/blob/v2.2.0/examples/src/main/scala/org/apache/spark/examples/sql/streaming/StructuredSessionization.scala