



Top Five Lessons Learned in Building Streaming Applications at Microsoft Bing Scale

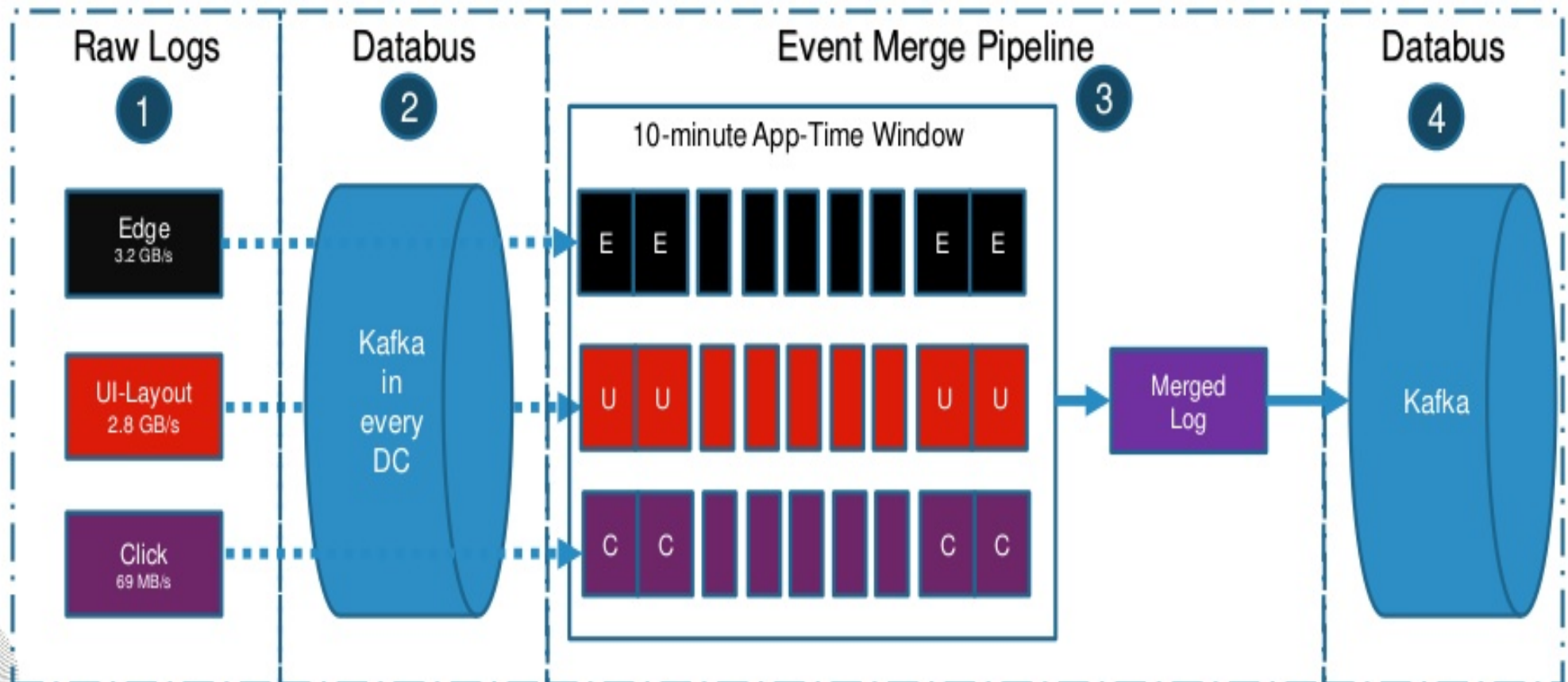
Renyi Xiong
Microsoft



SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE

Bing Scale Problem – Log Merging

- Merge Bing query events with click events
- Lambda architecture: batch- and stream-processing shares the same C# library
- Spark Streaming in C#



Mobius Talk

- **Tomorrow afternoon** 3pm at Imperial
 - [Mobius: C# Language Binding for Spark](#)



Lesson 1: Use *UpdateStateByKey* to join DStreams

• Problem

- Application time is not supported in Spark 1.6.

• Solution – UpdateStateByKey

- **UpdateStateByKey** takes a custom **JoinFunction** as input parameter;
- Custom **JoinFunction** enforces time window based on **Application Time**;
- **UpdateStateByKey** maintains partially joined events as the **state**

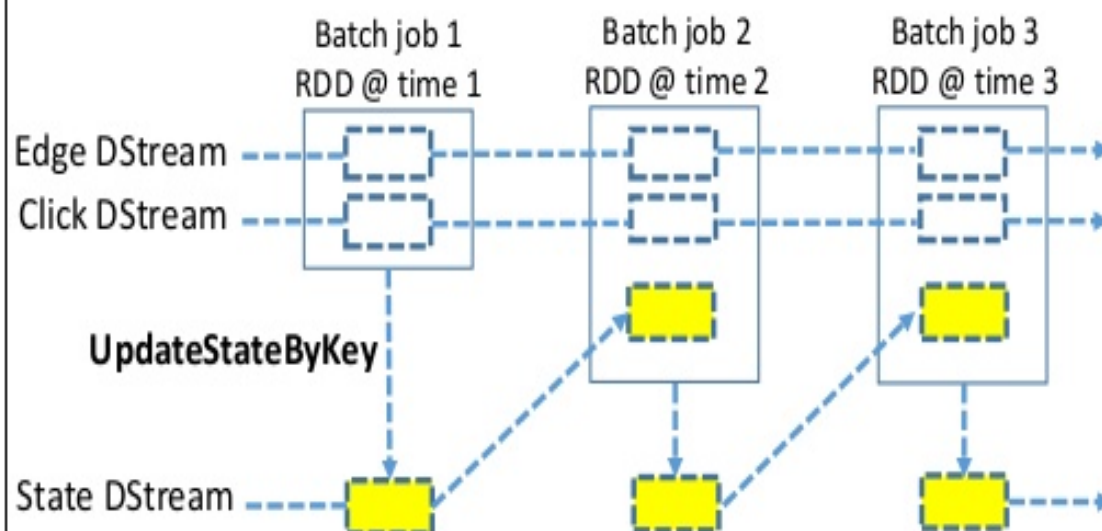
Pseudo Code

```
Iterator[(K, S)] JoinFunction(
  int pid, Iterator[(K:key, Iterator[V]:newEvents, S:oldState)] events)
{
  val currentTime = events.newEvents.max(e => e.eventTime);

  foreach (var e in events) {
    val newState = <oldState join newEvents>
    if (oldState.min(s => s.eventTime) + TimeWindow < currentTime) // TimeWindow 10minutes
      <output to external storage>
    else
      return (key, newState)
  }
}
```

UpdateStateByKey C# API

PairDStreamFunctions.cs, <https://github.com/Microsoft/Mobius>



Lesson 2: Dynamic Repartition with Kafka Direct Approach

• Problem

- Unbalanced Kafka partitions caused delay in the pipeline

• Solution – Dynamic Repartition

1. Repartition data from one Kafka partition into multiple RDDs without extra shuffling cost of DStream.Repartition
2. Repartition threshold is configurable per topic

Pseudo Code

```
class DynamicPartitionKafkaRDD(kafkaPartitionOffsetRanges)
  override def getPartitions {
    // repartition threshold per topic loaded from config
    val maxRddPartitionSize = Map<topic, partitionSize>

    // apply max repartition threshold
    kafkaPartitionOffsetRanges.flatMap { case o =>
      val rddPartitionSize = maxRddPartitionSize(o.topic)
      (o.fromOffset until o.untilOffset by rddPartitionSize).map(
        s => (o.topic, o.partition, s, (o.untilOffset, s + rddPartitionSize)))
    }
  }
}
```

Source Code

DynamicPartitionKafkaRDD.scala - <https://github.com/Microsoft/Mobius>

2-minute interval

Processing Time (?)
Avg: 3 minutes 10
seconds

► Input Rate

Avg: 100371.54
events/sec

Processing Time (?)
Avg: 1 minute 24
seconds

Before Dynamic Repartition



After Dynamic Repartition



Lesson 3: On-time Kafka fetch job submission

• Problem

- Overall job delay accumulates due to transient slow/hot Kafka broker issue.

• Solution

- Submit Kafka Fetch job on batch interval in a separate thread, even when previous batch delayed.

Pseudo Code

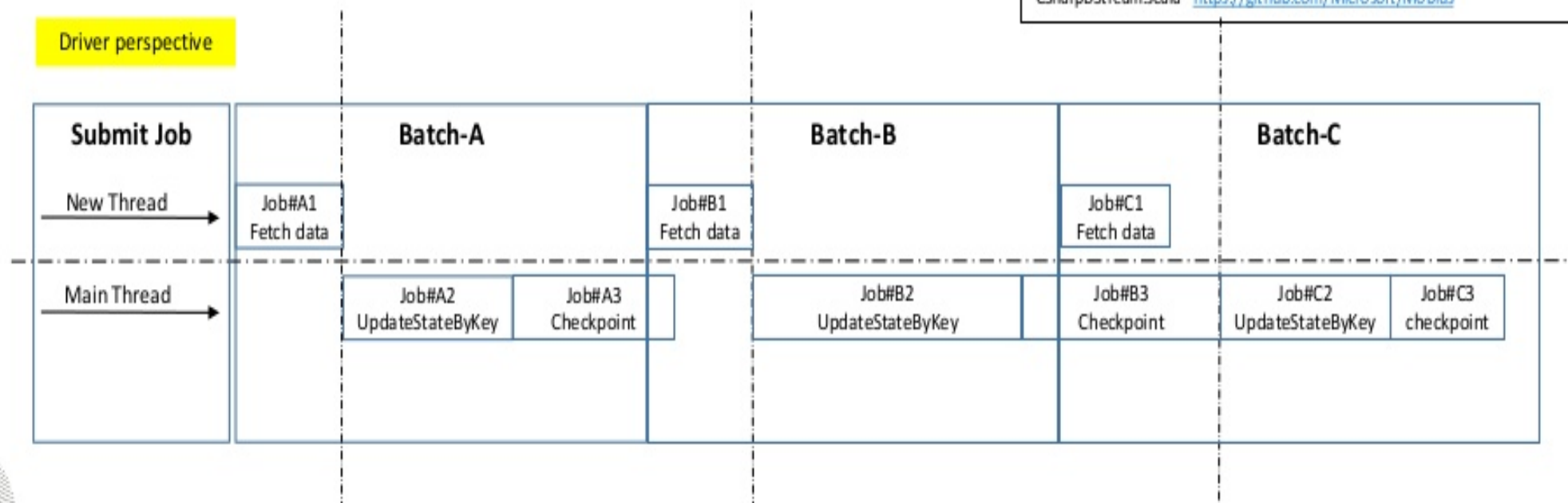
```
class CSharpStateDStream
  override def compute {
    val lastState = getOrCompute (validTime - batchInterval)

    val rdd = parent.getOrCompute(validTime)
    if (!lastBatchCompleted) {
      // if last batch not complete yet
      // run Fetch data job to materialize rdd in a separate thread
      rdd.cache()
      ThreadPools.execute(sc.runJob(rdd))
      // wait for job to complete
    }

    <compute UpdateStateByKey DStream>
  }
```

Source Code

CSharpDStream.scala - <https://github.com/Microsoft/Mobius>



Lesson 4: Parallel Kafka metadata refresh

• Problem

- Fetching Kafka metadata from multiple data centers often takes more time than expected.

• Solution

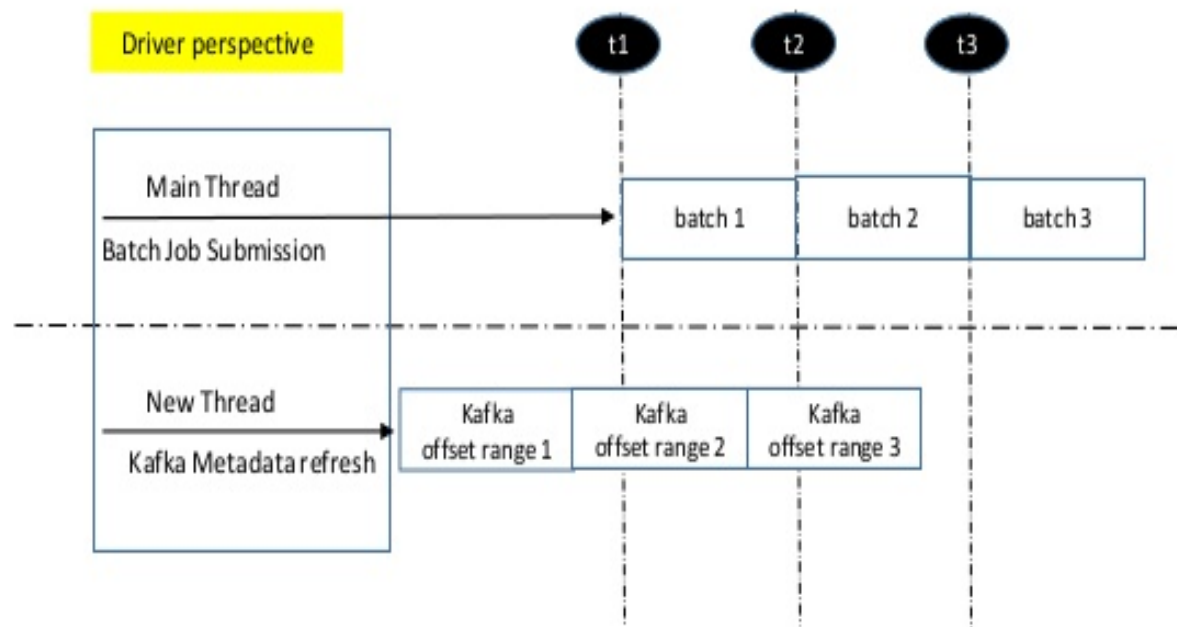
- Customize *DirectKafkaInputDStream*, move metadata refresh for each {topic, data-center} to a separate thread

Pseudo Code

```
class DynamicPartitionKafkaInputDStream {  
  // starts a separate schedule thread at refreshOffsetsInterval  
  refreshOffsetsScheduler.scheduleAtFixedRate(  
    <get offset ranges>  
    <enqueue offset ranges>  
  )  
  
  override def compute {  
    <dequeue offset ranges nonblockingly>  
    <generate kafka RDD>  
  }  
}
```

Source Code

DynamicPartitionKafkaInputDStream.scala - <https://github.com/Microsoft/Mobius>



Lesson 5: Parallel Kafka metadata refresh + RDD materialization

• Problem

- Kafka data fetch and data processing not in parallel

• Solution

- Take Lesson 4 further
- In metadata refresh thread, materialize and cache Kafka RDD

Pseudo Code

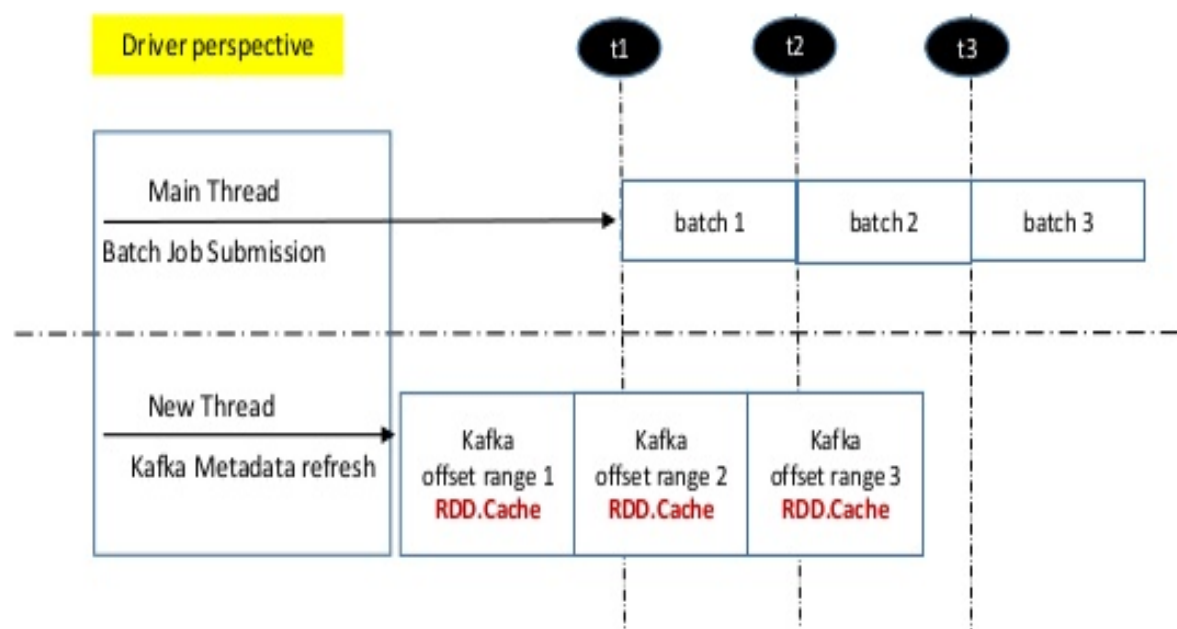
```
class DynamicPartitionKafkaInputDStream

// starts a separate schedule thread at refreshOffsetsInterval
refreshOffsetsScheduler.scheduleAtFixedRate(
  <get offset ranges>
  <generate kafka RDD>
  // materialize and cache
  sc.runJob(kafkaRdd.cache)
  <enqueue kafka RDD>
)

override def compute {
  <dequeue kafka RDD nonblockingly>
}
```

Source Code

DynamicPartitionKafkaInputDStream.scala - <https://github.com/Microsoft/Mobius>





THANK YOU.

- Special thanks to TD and Ram from Databricks for all the support
- Contact us
 - Renyi Xiong, renyix@microsoft.com
 - Kaarthik Sivashanmugam, ksivas@microsoft.com
 - mobiuscore@microsoft.com

