

Spark Streaming At Bing Scale

Kaarthik Sivashanmugam

@kaarthikss



Scale

- Billions of search queries per month
- Hundreds of services power Bing stack
- Thousands of machines – several Data Centers
- Tens of TBs of events per hour
- Several data processing frameworks

Data Curation

- Events of individual services - little value
- Need correlation of events & curated datasets
 - at scale, on time, high fidelity
 - contributes directly to improving quality of services & monetization

Data Pipelines

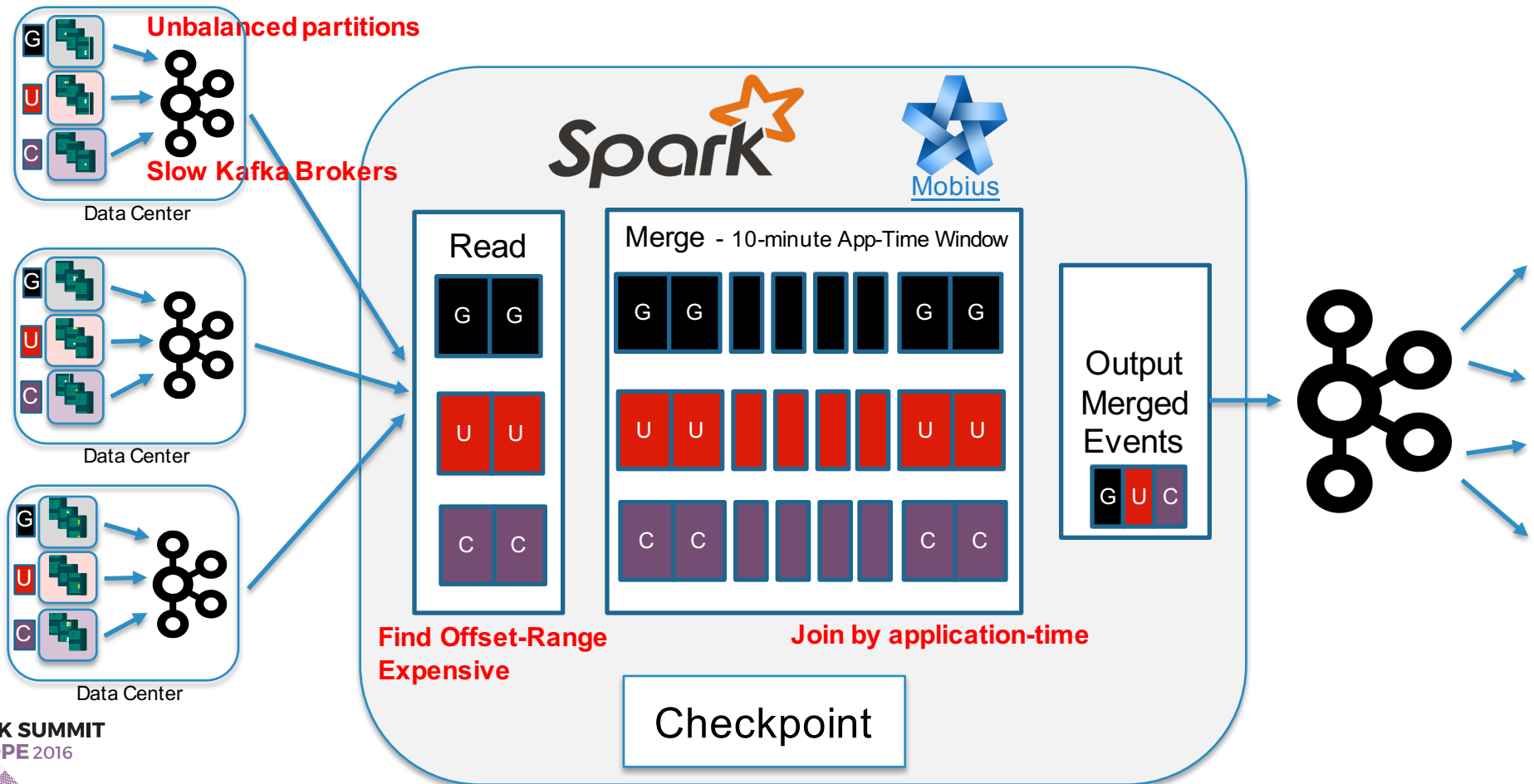
- Traditionally implemented entirely using Batch processing in COSMOS infrastructure
 - Storage – DFS (similar to HDFS)
 - Execution – Dryad (general purpose, more expressive than map-reduce)
 - Query – SCOPE (SQL 'style' scripting language that supports inline C#)
- Data pipelines are adopting near real-time processing – new issues to address

NRT Data Pipelines

Key issues to address in stream processing applications:

- Events generated in different DCs and at a rapid rate
- Events arrive out of order
- Events are delayed or get lost
- Managing state can be very expensive and hard to get right

NRT Processing Scenario – Event Merge Pipeline



Unbalanced Kafka Partitions

- Direct API - Kafka partition maps to RDD partition
- Largest partition is the long pole in processing
- Solution
 - Repartition data from one Kafka partition into multiple RDDs w/o extra shuffling cost of *DStream.Repartition()*
 - Repartition threshold is configurable per topic
 - *DynamicPartitionKafkaRDD.scala* at github.com/Microsoft/Mobius

Slow Kafka Brokers

- Slow Kafka brokers increase batch time
- Delay in starting the next batch accumulates
- Solution
 - Submit Kafka data-fetch job on-time (defined by batch interval) in a separate thread, even when previous batch delayed
 - *CSharpDStream.scala* at github.com/Microsoft/Mobius

Find Offset-Range Expensive

- Finding Offset-range for {DC X Topic X Partition} is expensive
 - Several DCs – 3 topics each – average of 170 partitions per topic
 - {Get metadata + get offset range} took 10 mins for 2 min batch window
- {Metadata refresh + Find Offset} and data processing not parallel
- Solution
 - Move find offset-range to a separate thread
 - Materialize and cache Kafka RDD in that thread
 - *DynamicPartitionKafkaInputDStream.scala* at github.com/Microsoft/Mobius

Join By Application-Time

- Application-time based join not available in Spark 1.*
- Solution
 - Use custom join function in *DStream.UpdateStateByKey()*
 - Custom join function enforces time window based on application time
 - *UpdateStateByKey* maintains partially joined events as the state
 - *PairDStreamFunctions.cs* at github.com/Microsoft/Mobius

THANK YOU.



SPARK SUMMIT
EUROPE 2016

ADDITIONAL SLIDES

Dynamic Repartition

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

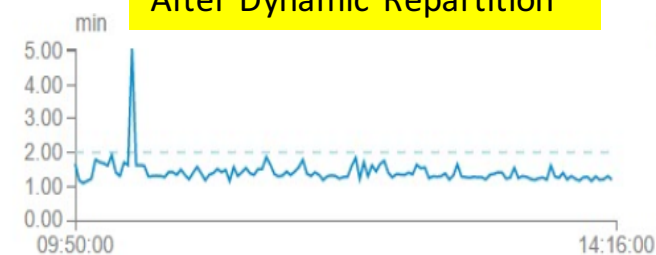
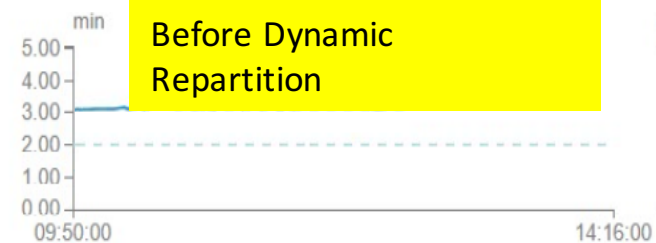
Processing Time
Avg: 3 minutes 10
seconds

► Input Rate

Avg: 100371.54
events/sec

Processing Time (?)

Avg: 1 minute 24
seconds



On-time Kafka fetch job submission

Pseudo Code

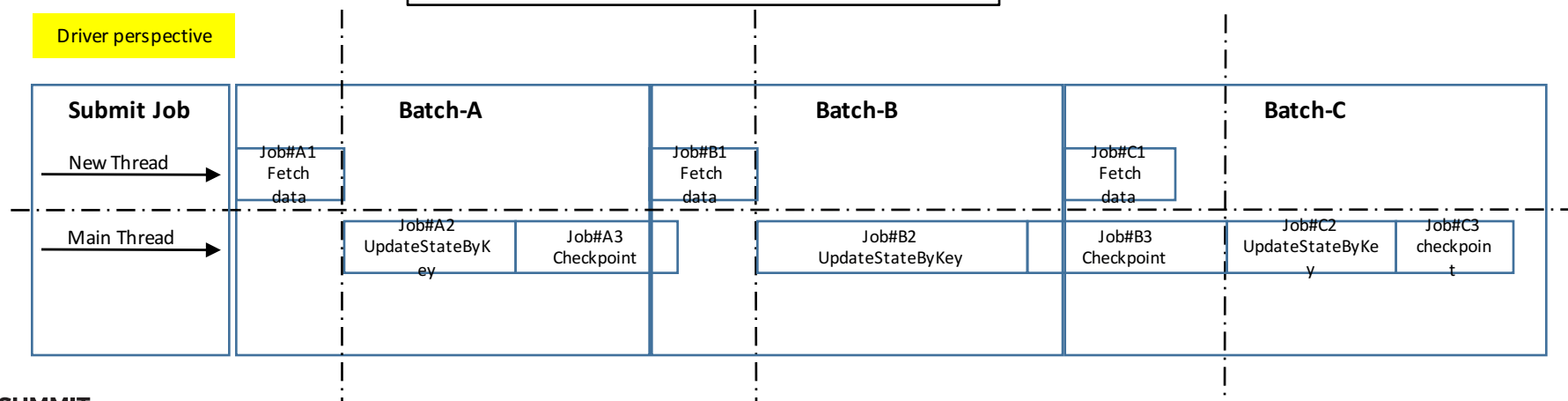
```
class CSharpStateDStream
  override def compute {
    val lastState = getOrCompute (validTime - batchInterval)

    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>



Parallel Kafka metadata refresh + RDD materialization

Pseudo Code

```
class DynamicPartitionKafkaInputStream
```

```
// 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>
```

```
)
```

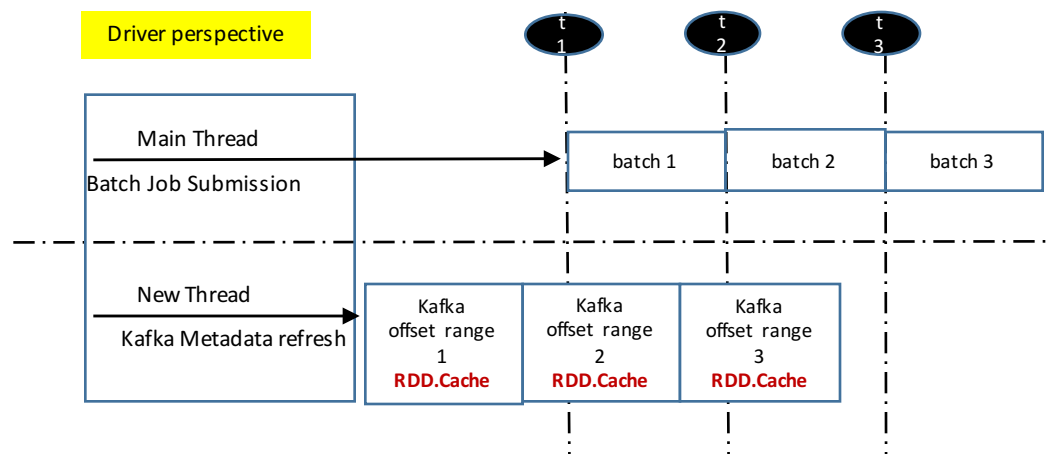
```
override def compute {
```

```
<dequeue kafka RDD nonblockingly>
```

```
}
```

Source Code

```
DynamicPartitionKafkaInputStream.scala - https://github.com/Microsoft/Mobius
```



Use *UpdateStateByKey* to join DStreams

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 10 minutes
      <output to external storage>
    else
      return (key, newState)
  }
}
```

UpdateStateByKey C# API

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

