Apache Beam Backend for Scalding

Navin Viswanath
Software Engineer, Twitter Inc.

BEAM SUMMIT Austin, 2022

Outline

- Data processing at Twitter
- Quick introduction to Scalding
- Beam backend for Scalding
- Current state and future work

Scale of batch data processing at Twitter

Around 50k+ batch data jobs run every day

Process 200+ petabytes of data daily

Multiple Hadoop clusters in two datacenters

50,000+ nodes across Hadoop clusters

Migrating batch pipelines

- Google Cloud Dataflow chosen for batch and streaming pipelines
- Why Dataflow?
  - Beam - unified API for batch and stream processing
  - Fully managed service
  - Leverage technologies in the Google cloud ecosystem
  - Elasticity
- Alternative: Hadoop on GCP

Migrating batch pipelines

- Two alternatives for pipeline owners:
  - Rewrite batch jobs to Apache Beam and deploy them on Dataflow
  - “Lift-and-shift” existing pipelines to Hadoop on GCP
- Manual rewrite is time-consuming : not practical at Twitter’s scale
- Lift-and-shift is straightforward : point Scalding job to a Hadoop cluster on GCP, but we are still running MapReduce under the covers
- The Beam backend for Scalding gives us the best of both worlds

What is Scalding?

- Scala library for Hadoop MapReduce
- Built on top of Cascading, a high level Java API that abstracts details of MapReduce
- Allows expressing computations on data as functional transformations such as map, filter, and reduce
- Computations are represented as an abstract syntax tree and submitted to a “backend”

Scalding API

- Central abstraction is a **`TypedPipe`** - a distributed collection
- Data processing operations are implemented as transformations on a TypedPipe
  - map/flatMap
  - filter
  - groupBy
  - join
- Sources and sinks can be implemented by extending TypedSource and TypedSink

Word count in Scalding

```
TypedPipe.from(TextLine(args("input")))
  .flatMap { line => tokenize(line) }
  .groupBy { word => word } // use each word for a key
  .size // in each group, get the size
  .write(TypedText.tsv[(String,Long)](args("output")))


// Split a piece of text into words
def tokenize(text: String): Array[String] = {
     // Lowercase each word and remove punctuation.
     text.toLowerCase.replaceAll("[^a-zA-Z0-9\\s]", "").split("\\s+")
}
```

Scalding backends

- Scalding planner represents the user code as a DAG (AST)
- Each “sink” node in the DAG is a **`Write`**
- Optimizer optimizes a list of **`Write`**s
- The list of **`Write`**s is submitted to the backend
- There are currently 4 backends for Scalding:
  - Memory
  - Cascading
  - Spark
  - Beam

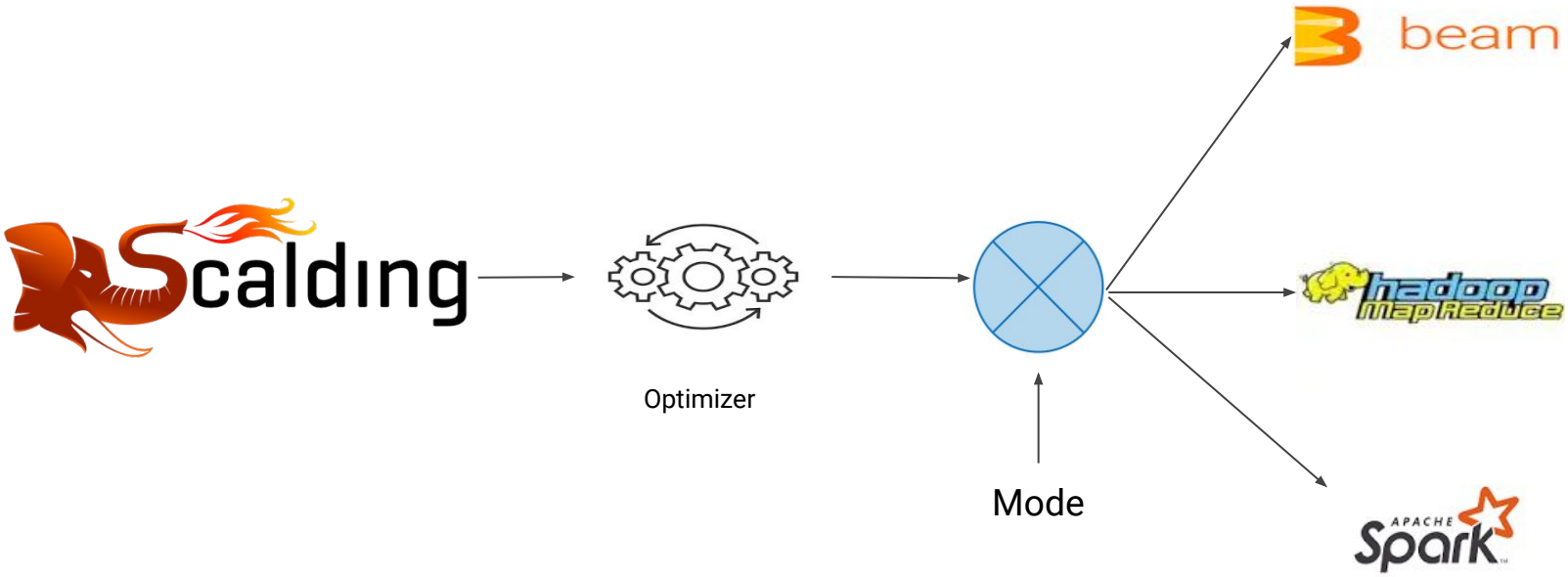
Dr. Scalding UI

0 MB Millis

100%

#	Hadoop Job ID	Status	Job Cost (MB Millis)	Features	Duration	Source Code Location
1	job_1656453935712_805895	succeeded	0	TimePartitionedDAL Each, TempHfs	5m 31s	com.twitter.accounts.scalding.account_health.HealthJob.getAllDesiredPromptEvents(HealthJob.scala:51), com.twitter.accounts.scalding.account_health.HealthJob.getAllDesiredPromptEvents(HealthJob.scala:53)
2	job_1656453935712_805896	succeeded	0	TimePartitionedDAL Each, TempHfs	2m 15s	com.twitter.accounts.scalding.account_health.EmailHealthJob.getEmailChanges(EmailHealthJob.scala:202), com.twitter.accounts.scalding.account_health.EmailHealthJob.getEmailChanges(EmailHealthJob.scala:205)
3	job_1656453935712_806384	succeeded	0	TempHfs, Each, Merge, Each, GroupBy, Every, Each, Each, Each,	5m 30s	com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:89), com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:93), com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:94), com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:102)
4	job_1656453935712_806383	succeeded	0	TempHfs, Each, Merge, Each, GroupBy, Every, Each, Each, Each,	2m 51s	com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:89), com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:93), com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:94), com.twitter.accounts.scalding.account_health.HealthJob.filterAndCountEventsFor(HealthJob.scala:102)

Scalding backends



Scalding backends

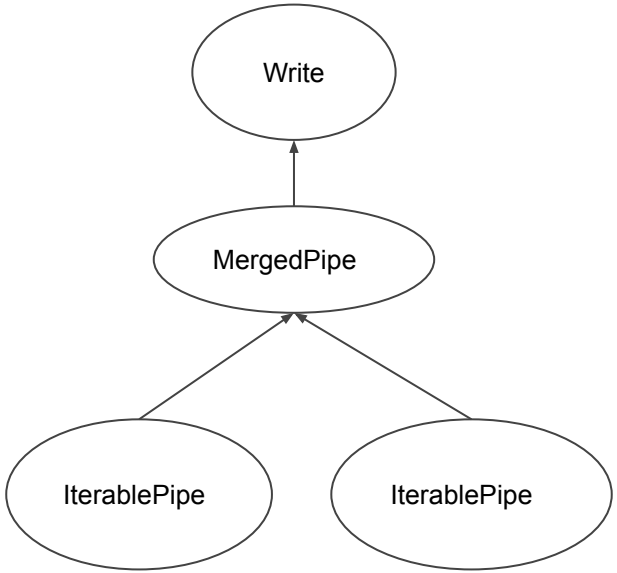
```
trait Mode extends java.io.Serializable {
     def newWriter(): Writer
}

Trait Writer {
     override def execute(conf: Config, writes: List[ToWrite[_]])(implicit
          cec: ExecutionContext
      ): CFuture[(Long, ExecutionCounters)]
}
```

Implementing a Scalding backend

- Design an AST representation for the processing engine (BeamOp)
- Implement a function Function[TypedPipe, BeamOp]
- Implement a Resolver to map TypedPipe sources and sinks to Beam sources and sinks (Resolver[TypedSource, BeamSource] and Resolver[TypedSink, BeamSink])
- Implement a writer (BeamWriter)

Scalding DAG




```
val a = TypedPipe.from(Seq(1,2,3))

val b = TypedPipe.from(Seq(4,5,6))

val union = a ++ b

union.write(TypedText.tsv[Long](args(
"output")))
```

Scalding Beam backend

```
final case class MergedBeamOp[A](first: BeamOp[A], second: BeamOp[A], tail: Seq[BeamOp[A]])
      extends BeamOp[A] {

     override def run(pipeline: Pipeline): PCollection[_ <: A] = {
      val collections = PCollectionList
        .of(first.run(pipeline))
        .and(second.run(pipeline))
        .and(tail.map(op => op.run(pipeline)))
      collections.apply(Flatten.pCollections[A]())
    }

}
```

Scalding Beam backend

```
case (m @ MergedTypedPipe(_, _), rec) =>
          OptimizationRules.unrollMerge(m) match {
            case Nil                          => rec(EmptyTypedPipe)
            case single :: Nil                => rec(single)
            case first :: second :: tail => MergedBeamOp(rec(first),
rec(second), tail.map(rec(_)))
          }
```

Current state and future work

- Current state
  - We’re testing this backend on production pipelines at Twitter
  - We have established correctness
- Future work
  - Distributed cache
  - MapReduce counters
  - Improve debuggability
  - Performance improvements

Thank you!

Twitter Careers
Twitter Engineering Blog
Twitter Open Source

BEAM SUMMIT Austin, 2022

Questions?

https://github.com/twitter/scalding

BEAM SUMMIT Austin, 2022

Section title