Streaming MapReduce with Summingbird

Today we are open sourcing **Summingbird** on GitHub under the ALv2.

---

**Twitter Open Source**

@TwitterOSS

we're thrilled to open source @summingbird, streaming mapreduce with @scalding and @stormprocessor
#hadoop blog.twitter.com/2013/streaming...

9:04 AM - 3 Sep 2013

**Streaming MapReduce with Summingbird | Twitter Blogs**

By Chris Aniszczyk @cra

Today we are open sourcing Summingbird on GitHub under the ALv2. Summingbird is a library that lets you write streaming MapReduce programs th......

**Twitter Engineering** @TwitterEng

59 RETWEETS 46 FAVORITES
Oscar Boykin - @aposco
Sam Ritchie - @sritchie
Ashu Singhal - @daashu
- What is Summingbird?
- What can it do today?
- Functional Design
- Currently deployed systems
- Upcoming Features
Vision
Write your logic once.
Twitter’s Scale

- 200M+ Active Monthly Users
- 500M Tweets / Day
- Several 1K+ node Hadoop clusters
Solve systems problems once.
Make **non-trivial**
realtime compute
as accessible
as Scalding.
What is Summingbird?

- Declarative Streaming Map/Reduce DSL
- Realtime platform that runs on Storm.
- Batch platform that runs on Hadoop.
- Batch / Realtime Hybrid platform
val impressionCounts =
impressionHose.flatMap(extractCounts(_))

val engagementCounts =
engagementHose.filter(_.isValid)
  .flatMap(engagementCounts(_))

val totalCounts =
  (impressionCounts ++ engagementCounts)
  .flatMap(fanoutByTime(_))
  .sumByKey(onlineStore)

val stormTopology =
Storm.remote("stormName").plan(totalCounts)

val hadoopJob =
Scalding("scaldingName").plan(totalCounts)
FlatMap

flatMap: T => TraversableOnce[U]

// g: (x: T => U)
map(x) = flatMap(x => List(g(x))

// pred: T => Boolean
filter(x) = flatMap { x =>
    if (pred(x)) List(x) else Nil
}
- Source[+T]
- Store[-K, V]
- Sink[-T]
- Service[-K, +V]
The Four Ss!

- Source[+T]
- Store[-K, V]
- Sink[-T]
- Service[-K, +V]
Store[-K, V]:
What values are allowed?
trait Monoid[T] {
  def zero: T
  def plus(l: T, r: T): T
}
- Tons O’Monoids:
- CMS, HyperLogLog, ExponentialMA, BloomFilter, Moments, MinHash, TopK
Did you just tell me to go fuck myself?

I believe I did, Bob.
Associativity
7 steps

\[ a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 \]
;; 7 steps
(+ a0 a1 a2 a3 a4 a5 a6 a7)
5 steps

\[ (+ (+ a_0 a_1)) \]
\[ (+ a_2 a_3) \]
\[ (+ a_4 a_5) \]
\[ (+ a_6 a_7)) \]
3 steps

(\( + ( + ( + \ a_0 \ a_1 ) \)
(\( + \ a_2 \ a_3 ) ) \)
(\( + ( + \ a_4 \ a_5 ) \)
(\( + \ a_6 \ a_7 ) ) ) \)
Parallelism

❤️

Associativity
Batch / Realtime

Noisy:

Realtime sums from 0, each batch

fault tolerant:

BatchID: 0 1 2 3

Sunday, September 22, 13
Batch / Realtime

Noisy:

Hadoop
Log
RT
Hadoop
Log
RT
Hadoop
Log
RT
Hadoop
Log

fault tolerant:

Hadoop
Hadoop
Hadoop
Hadoop

BatchID:

0
1
2
3

Hadoop keeps a total sum (reliably)
Batch / Realtime

Noisy:

Sum of RT Batch(i) + Hadoop Batch(i-1) has bounded noise, bounded read/write size

fault tolerant:

BatchID: 0 1 2 3
Tweet Embed Counts

How to Embed a Tweet on your Website

Every Tweet on twitter.com and Tweetdeck has a set of Tweet actions at the bottom, including Reply, Retweet, Favorite, and More. Click the "More" Tweet action and select "Embed Tweet":

Example Tweet:
Barack Obama @BarackObama
Four more years. pic.twitter.com/bAJE6vom

Embed Tweet Button

Example Tweet:
Barack Obama @BarackObama
We’re all in this together. That’s how we campaigned, and that’s how we’re going to govern.
This one pretty much speaks for itself. A simple message that acknowledges what everyone reading the tweet will already know.

Barack Obama
@BarackObama

Four more years. pic.twitter.com/bAJE6Vom

11:16 PM - 6 Nov 2012

President Obama's message became the most shared tweet of all time within a day of it going out.

9. Bill Gates’ tribute to Steve Jobs
(TweetID, Map[URL, Long])

groupBy TweetID

reduce: (x,y) => MapMonoid

Tweets

(HDFS/Queue)

(Flat)Mappers

Reducers

HDFS/Queue

Sunday, September 22, 13
object OuroborosJob {
  def apply[P < Platform[P]](source: Producer[P, ClientEvent], sink: P#Store[OuroborosKey, OuroborosValue]) =
    source.filter(filterEvents(_))
      .flatMap { event =>
        val widgetDetails = event.getWidget_details
        val referUrl: String = widgetDetails.getWidget_origin
        val timestamp: Long = event.getLog_base.getTimestamp
        val widgetFrameUrlOpt: Option[String] = Option(widgetDetails.getWidget_frame)
        for {
          tweetId: java.lang.Long <- javaToScalaSafe(event.getEvent_details.getItem_ids)
          timeBucketOption: Option[TimeBucket] <- timeBucketsForTimestamp(timestamp)
        } yield {
          val urlHllOption = canonicalUrl(referUrl).map(hllMonoid.create(_))
          val widgetFrameUrlsOption = widgetFrameUrlOpt map { widgetUrl: String =>
            widgetFrameUrlsSmMonoid.create((referUrl, (widgetFrameUrlSetSmMonoid.create((widgetUrl, 1L)), 1L)))
          }
          val impressionsValue: OuroborosValue = RawImpressions(
            impressions = 1L,
            approxUniqueUrls = urlHllOption,
            urlCounts = Some(embedCountSmMonoid.create((referUrl, 1L))),
            urlDates = Some(embedDateSmMonoid.create((referUrl, timestamp))),
            frameUrls = widgetFrameUrlsOption
          ).as[OuroborosValue]
          Seq(
            (OuroborosKey.ImpressionsKey(ImpressionsKey(tweetId.longValue, timeBucketOption)), impressionsValue),
            (OuroborosKey.TopTweetsKey(TopTweetsKey(timeBucketOption)), topTweetsValue)
          )
        }
      }.sumByKey(store)
        .set(MonoidIsCommutative(true))
}
object OuroborosJob {
  def apply[P <: Platform[P]](source: Producer[P, ClientEvent], sink: P#Store[OuroborosKey, OuroborosValue]) =
    source.filter(filterEvents(_:_))
      .flatMap {
          event =>
            val widgetDetails = event.getWidget_details
            val referUrl: String = widgetDetails.getWidget_origin
            val timestamp: Long = event.getLog_base.getTimestamp
            val widgetFrameUrlOpt: Option[String] = Option(widgetDetails.getWidget_frame)
            for {
              tweetId: java.lang.Long <- javaToScalaSafe(event.getEvent_details.getItem_ids)
              timeBucketOption: Option[TimeBucket] <- timeBucketsForTimestamp(timestamp)
            } yield {
              val urlHllOption = canonicalUrl(referUrl).map(hllMonoid.create(_))
              val widgetFrameUrlsOption = widgetFrameUrlOpt map {
                widgetUrl: String =>
                widgetFrameUrlsSmMonoid.create((referUrl, (widgetFrameUrlSetSmMonoid.create((widgetUrl, 1L)), 1L)))
              }
              val impressionsValue: OuroborosValue = RawImpressions(
                impressions = 1L,
                approxUniqueUrls = urlHllOption,
                urlCounts = Some(embedCountSmMonoid.create((referUrl, 1L))),
                urlDates = Some(embedDateSmMonoid.create((referUrl, timestamp))),
                frameUrls = widgetFrameUrlsOption
              ).as[OuroborosValue]
              Seq(
                (OuroborosKey.ImpressionsKey(ImpressionsKey(tweetId.longValue, timeBucketOption)), impressionsValue),
                (OuroborosKey.TopTweetsKey(TopTweetsKey(timeBucketOption)), topTweetsValue)
              )
            }
            .sumByKey(store)
            .set(MonoidIsCommutative(true))
        }
object OuroborosJob {
  def apply[P <=: Platform[P]](source: Producer[P, ClientEvent], sink: P#Store[OuroborosKey, OuroborosValue]) =
  source.filter(filterEvents(_))
    .flatMap { event =>
      val widgetDetails = event.getWidget_details
      val referUrl: String = widgetDetails.getWidget_origin
      val timestamp: Long = event.getLog_base.getTimestamp
      val widgetFrameUrlOpt: Option[String] = Option(widgetDetails.getWidget_frame)
      for {
        tweetId: java.lang.Long <- javaToScalaSafe(event.getEvent_details.getItem_ids)
        timeBucketOption: Option[TimeBucket] <- timeBucketsForTimestamp(timestamp)
      } yield {
        val urlHllOption = canonicalUrl(referUrl).map(hllMonoid.create(_))
        val widgetFrameUrlsOption = widgetFrameUrlOpt map { widgetUrl: String =>
          widgetFrameUrlsSmMonoid.create((referUrl, (widgetFrameUrlSetSmMonoid.create((widgetUrl, 1L)), 1L)))
        }
        val impressionsValue: OuroborosValue = RawImpressions(
          impressions = 1L,
          approxUniqueUrls = urlHllOption,
          urlCounts = Some(embedCountSmMonoid.create((referUrl, 1L))),
          urlDates = Some(embedDateSmMonoid.create((referUrl, timestamp))),
          frameUrls = widgetFrameUrlsOption
        ).as[OuroborosValue]
        Seq(
          (OuroborosKey.ImpressionsKey(ImpressionsKey(tweetId.longValue, timeBucketOption)), impressionsValue),
          (OuroborosKey.TopTweetsKey(TopTweetsKey(timeBucketOption)), topTweetsValue)
        ).sumByKey(store)
        .set(MonoidIsCommutative(true))
    }
}
object OuroborosJob {
  def apply[P <=: Platform[P]](source: Producer[P, ClientEvent], sink: P#Store[OuroborosKey, OuroborosValue]) =
    source.filter(filterEvents(_))
      .flatMap {
        event =>
          val widgetDetails = event.getWidget_details
          val referUrl: String = widgetDetails.getWidget_origin
          val timestamp: Long = event.getLog_base.getTimestamp
          val widgetFrameUrlOpt: Option[String] = Option(widgetDetails.getWidget_frame)
          for {
            tweetId: java.lang.Long <- javaToScalaSafe(event.getEvent_details.getItem_ids)
            timeBucketOption: Option[TimeBucket] <- timeBucketsForTimestamp(timestamp)
          } yield {
            val urlHllOption = canonicalUrl(referUrl).map(hllMonoid.create(_))
            val widgetFrameUrlsOption = widgetFrameUrlOpt map {
              widgetUrl: String =>
                widgetFrameUrlsSmMonoid.create((referUrl, (widgetFrameUrlSetSmMonoid.create((widgetUrl, 1L)), 1L)))
            }
            val impressionsValue: OuroborosValue = RawImpressions(
              approxUniqueUrls = urlHllOption,
              urlCounts = Some(embedCountSmMonoid.create((referUrl, 1L))),
              urlDates = Some(embedDateSmMonoid.create((referUrl, timestamp))),
              frameUrls = widgetFrameUrlsOpt
            ).as[OuroborosValue]
            Seq(
              (OuroborosKey.ImpressionsKey(ImpressionsKey(tweetId.longValue, timeBucketOption)), impressionsValue),
              (OuroborosKey.TopTweetsKey(TopTweetsKey(timeBucketOption)), topTweetsValue)
            )
          }
      }.sumByKey(store)
      .set(MonoidIsCommutative(true))
}
Brief Explanation

This job creates two types of keys:

1: ((TweetId, TimeBucket) => [URL, Impressions])
2: TimeBucket => Map[TweetId, Impressions]
What Else?
Twitter Advertising

WEDNESDAY, MARCH 13, 2013

The new Twitter Ads center

Today, we’re excited to share some changes we’ve made to the Twitter Ads center. Based on feedback from our advertisers, we’ve created a revamped experience that improves campaign reporting, provides more visibility into campaign performance analytics and spend, and also makes it easier to manage campaigns in real time.

A major focus of ours is improving campaign analytics. With this in mind, we are now reporting all engagements that Promoted Tweets receive — not just engagements that advertisers pay for, but earned media as well. This change gives marketers more complete insight into the impact Promoted Tweets have in driving engagement and exposure on Twitter.
What’s Next?
Future Plans

- Akka, Spark, Tez Platforms
- Pluggable graph optimizations
- Metadata publishing via HCatalog
- More tutorials!
Open Source!
Summary

- Summingbird is appropriate for the majority of the real-time apps we have.
- It’s all about the Monoid
- Data scientists who are not familiar with systems can deploy real-time systems.
- Systems engineers can reuse 90% of the code (batch/realtime merging).
Thank You!

Follow me at @sritchie