Date: 8/28/2014
Place: Twitter HQ
Participants: Joep Rottinghuis, Maysam Yabandeh, Sangjin Lee (Twitter), Jon Eagles, Jason Lowe (Yahoo!), Ram Venkatesh, Zhijie Shen, Vinod Kumar Vavilapalli (Hortonworks), Eric Wohlstadter (Altiscale), Karthik Kambala, Robert Kanter (Cloudera), Mohammad Islam, Carl Steinbach (LinkedIn), Mayank Bansal (eBay)

[agenda]
- introduction
- discussion on use cases & needs
- current status & vision of ATS
- design & scalability concerns with ATS

[use cases]
(Vinod)
- after mapreduce, we started seeing things like tez and spark
- we took some shortcuts for MR in terms of history server
- when we thought of ATS, we had 3 major use cases in mind
(1) we need to have application data after the fact (historical) across all applications without apps reinventing the wheels
(2) we want to understand what's happening in the cluster across all applications without apps doing their own things
(3) drive a coherent single UI (as opposed to replicated effort for AM UI and JHS UI)
- now there seems to be demand for more use cases for ATS
  - for examples, node-level metrics if feasible
  - long-running services
  - logs

(Joep/Sangjin)
- at Twitter we have hRaven that handles MR historical data at the flow level and scales
- the notions of a flow and a run are essential in hRaven, and shape things like historical trending and chargeback
- we also use hRaven to do things like reducer estimation, training applications based on historical data of the flow
- hRaven uses HBase as the backend, and it's scaled well for us
- the current gap is with non-MR apps as they are becoming more important (e.g. spark)
- we also desire a single coherent system that can cover near-realtime information
- we could build tools like Ambrose on top
- we got real excited about ATS because we thought that ATS could address the gap that we had a need for
- we're completely open-minded about the future of hRaven, and be happy to switch to ATS and contribute to ATS if it fits our needs and scales at our scale

- scalability is one of our overriding concern: being able to run it with thousands of nodes, hundreds of active apps and tens of thousands of containers running
- again, to us the notion of a flow should be a first-class concept

(Jason/Jon)
- we (Yahoo!) also have our internal tool: starling
- we're also concerned about scale
- when things get slow, how do things behave? events queued? what are the guarantees? in other words, flow control
- scale issue (not another job tracker)
- if it is hbase, how big should it be?
- the main use case for ATS so far has been for tez
- we do like the one stop shop for the UI backend
- we're hesitant to put MR on ATS until it reaches a full parity with JHS
- we would also love to see flows, but not the most important
- we're concerned ATS may grow too big in terms of features
- it would be OK if ATS doesn't do all the things hRaven does
- another secondary use case: storm on yarn
- chargeback for long running jobs is important
- i want the view into capacity consumption to be the yarn thing
- can we also capture some basic metrics like hdfs bytes?

(Ram)
- we've been focusing on yarn metrics (e.g. megabyte millis), but you're asking hadoop metrics (e.g. HDFS bytes reads/writes, app metrics)

(Vinod)
- Hortonworks has something called yarntop

(Jon/Jason)
- yarn will never build app-level history server

(Karthik)
- at Cloudera, we care deeply about lifecycle events, just like JHS
- we also want to store some app-specific info
- metrics: final values matter
- we care more about some time series data for running apps
- another question we can ask is, would it make sense to take this into account for scheduling?
- we're interested in things like min/max/current
- constraint: job performance should not suffer
- it's a big concern if ATS is a single end point for writes
- want something that would scale to different rates
- we also care about flows

- spark streaming
- UI: like the plugin idea
- we have reservations about HBase: adding complexity to deployments for customers
- it might be good to have a possible backend UI
- we like the idea of clients writing to hdfs as opposed to writing through to a single entity (solving the availability problem and distributed writes)
- we need the ability of deleting older values
- care about both app metrics and node metrics in the context of being able to attribute utilization to apps
- doing everything openTSDB does within ATS would not be practical

(Jason)
- storm has a prototype that pushes things to ATS

(Mayank)
- we are using HBase for in-house system metrics system
- would be good to support both app metrics and system metrics with ATS
- correlating apps and system metrics is useful

(Zhijie)
- currently ATS isn't considering time series

(Ram)
- you want to be able to thread system and app metrics
- think of ATS as storage

(Karthik)
- concerned about the storage cost if both are kept

(Eric)
- it would be useful to do machine learning and data mining about the hadoop cluster itself
- ATS would become the data store for hadoop itself

(Carl)
- it would be nice to discover metrics: taxonomy & ontology

(Joep)
- even though the flexible data model is good, some first class anchors are still needed

(Mohammad)
- non-MR apps is a must: tez and spark
- not sure if everything fits into ATS
- should be careful not to overload ATS and break core features

- definitely need app metrics
- example: user job declared 8 G, but used only 2 G: how do you find out?
- same thing with cpu's
- flow aggregation is very important: pig, hive, oozie
- system level metrics is desired in the context of correlating

(Carl)
- Dr. elephant analyzes jobs and makes suggestions
- we do things like versioning (hadoop version, software version, config change, etc.) to identify changes and correlate performance change
- exception fingerprinting (hblog)
- throttling hdfs is also interesting
- we can think of things like A/B testing at the cluster level
- can feed it into yarn container scheduling

(Eric)
- capacity planning, job tuning
- have a hard time mapping from job names to which applications
- which type? pig? hive? tags? application type?
- are yarn tags an answer?

(Mohammad)
- some kind of pluggability is desired

[Current status]
(Vinod)
- YARN-1530
- there is a debate between REST and RPC for writes
- we haven't exposed the REST as public end point
- it is supposed to go through the TimelineClient API
- entities can be nested
- how does AM know it is part of an app?
- there needs to be a top-level entity that needs to be written by the client that initiates things, then things can be related
- writes do go through a single instance (?)
- reads can be served by a cluster of stateless instances
- in terms of data model, fancy ad hoc queries can be a problem
- one answer for scalability: aggregating at the AM level (MR)
- value of client libraries is so that it can handle unavailability
- security

(Mayank)
- with phoenix you can do ad hoc queries

(Joep)
- we can do efficient range scans given good key definitions
- we're definitely concerned about a single writer, and aren't sure if it will perform at our scale
- typical numbers: 10,000 tasks * 100 counters = 1,000,000 counters
- with hRaven (HBase client) we've observed 2 - 3k puts / second
- individual puts v. batched puts: has pros and cons on both sides (# of calls v. size of the payload)
- reads for running apps can be served both from the backend and the AM: it would be good to have a single read API that does that
- offline aggregation (rolling up metrics to the flow level) doesn't work at scale
- concept of flows should be made explicit