Using Summingbird for aggregating eye tracking data to find patterns in images in a multi-user environment

Johan Fogelström and Remzi Can Aksoy

School of Computer Science and Communication (CSC), Royal Institute of Technology KTH, Stockholm, Sweden

Abstract. Hadoop which is an implementation of MapReduce is a powerful batch processing tool to handle big data problems. Abstractions like Pig or Cascading on top of Hadoop provide the way for analytics and building data products in data warehouses. Over the years, real time data analysis also gain importance for the companies. The standard approach to process large data streams is to run periodic batch jobs (e.g., hourly). Batch processing paradigms care with throughput of jobs but generally have difficult handling latency sensitive jobs and this requires on-line data processing platforms like Storm. Having Storm allows developer to create interactive products to answer data related questions in real time. However, it decreases efficiency of developer, if both batch and on-line solutions are required because the code is written twice. Summingbird is critical tool which integrates batch and on-line computations for efficient development by allowing to run the same code on both platform. Our aim in the project is to develop a Summingbird application which can run both on-line and batch mode to process eye tracking data. Unfortunately, various issues during early development forced us to end the project without satisfactory results.

1 Background

1.1 Hadoop

Hadoop is an Apache project and a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.[2]

Data scientists typically use a higher-level data-flow language such as Pig or Scalding (Twitter's Scala API to Cascading) for ease of development.

1.2 Storm

Storm is an Apache project for on-line big data calculations using a cluster of nodes connected in a graph determined by the calculations one wish to be performed. It has been in active development since 2011 and is today used by many big companies, including Twitter, Spotify, Yahoo and others.[3]

Storm use Zookeeper to manage the cluster, where each worker node (Known in the Storm project as a "Bolt") performs a small piece of the overall calculation before passing it on to its own output streams, which other nodes subscribes to, in order to perform the next steps of the calculations. Data is fed into the system using a set of "Spouts" which fetches the data from some source. Bolts may then subscribe to the various spouts output streams, which in their view behaves as a bolt node.

1.3 Summingbird

Hadoop is the standard for batch analytics, although it is mostly accessed via higher-level abstractions such as Scalding and Pig. For on-line processing and real time analytics, Storm has emerged as the standard execution framework.

It is clear that Hadoop and Storm both have their place, but merely standardizing on the two processing frameworks does not solve the problem of a developer needing to write everything twice. It would be desirable to have an abstraction for expressing analytical queries that is agnostic to batch or on-line processing, and have a system automatically generate Hadoop jobs or storm topologies as appropriate. Summingbird does exactly this.

Summingbird is an open source domain specific language which is implemented with Scala in Twitter to integrate on-line and batch MapReduce computations. Summingbird programs are written using data-flow abstractions such as sources, sink and stores. They can run on different execution platforms like Scalding (batch), Storm(on-line) and can also operate in hybrid processing mode.

1.4 Eye Tracking

Eye Tracking is a biometrics sensor for computing systems to determine the state of a user's eyes and their movement. There are several manufacturers, but one of the largest on the European market, and the one most familiar with the authors is Tobii Technologies REX trackers.

The technology is today most used in market analysis and providing aid for people with disabilities, but the technology have much potential for other use cases.

From this projects point of view, eye trackers is a good source of lots of data due to their high sampling rate. Generating the amount of data necessary to see the difference between traditional and big data algorithms is difficult. A biometrics sensor can provide real data with relatively few users, as opposed to faked or mocked data based on statistical analysis that may be flawed in bias or similar.

2 Method

The main focus of the project was to develop an application that could be deployed in a mixed-mode using both on-line and batch processing of eye tracking data from both live trackers and stored data from a database. As seen in figure 1, this is the large piece to the right.

In order to make this as smooth as possible, the raw eye tracking data was to be processed separately in a wrapper first for two primary reasons

1. Decrease code duplication since doing this preprocessing in the spout would make it unable to be reused when streaming to database
2. There has been no public release of a Java API for the eye trackers so a wrapper would have to be written to bridge this gap

By placing this common procedures in a separate process, we could work around these issues.

Additionally a master program was envisioned that would coordinate all programs for the experiments, display images for the user and provide additional metadata to the data produced by the eye trackers.

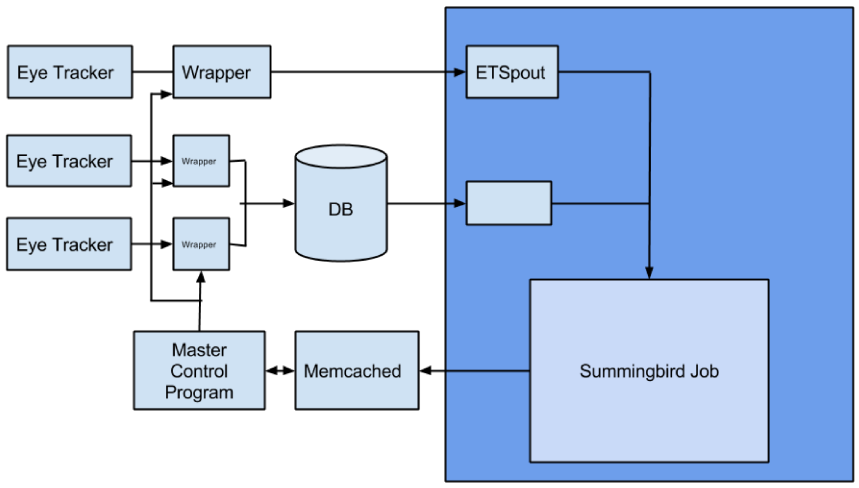

Fig. 1. the project plan layout

3 Issues

The first issue encountered was getting the example program to run on our systems. Summingbird comes with a example application using Storm to do word counting on a twitter feed. Unfortunately, the development system we setup in preparation was unable to compute this example.

Setting everything up in accordance with the instructions (The only difference being the chosen operating system, Arch Linux instead of OSX) the example did not seem to find any word at all in the feed, including words known to appear in the feed.

While reading up some on the Summingbird mailing list, we found that most users of Summingbird runs it on Ubuntu, so we decided to try that instead. This meant we had to redo the system setup in Ubuntu instead. This resulted in a lot of time wasted debugging, searing for answers and reinstalling the system

Once the example was running, we wanted to be able to write our own application. In order to make the build-process and dependency management as easy as possible, we went for the Gradle build system. Unfortunately, Summingbird consists of lots of loosely dependent projects, where you have type dependencies from sub-projects that are not actually dependencies in the package managers, so several dependencies had to be hunted down manually and added.. In the end, one dependency required was too new for all repositories available to Gradle. The only was to successfully build anything was to use the build-system used by Summingbird itself and included with the example, SBT. The reason this was not considered before was because the complexity of the system and the fact it used the for us unfamiliar language Scala to define the build process. Once all other options was eliminated, we returned to SBT.

As mentioned, the Summingbird libraries make use various types from multiple types from a variety of libraries. It also, naturally since it is developed in Scala, makes use of several Scala types. Some of these are not naturally used in Java, like passing functions as values. This itself is possible in Java using anonymous types as mediators, but the Scala generic versions for this proved difficult to get right with multiple layers of generics stacked on each other. It became clear Summingbird was designed for use with Scala, and the Java bindings was not prioritised.

To save time, the project was switched to Scala, since combating the rigid type-system of java was not the main focus of the project.

In the end, we encountered a issue deep withing Summingbird itself. After a while, both our own algorithms and the provided example stopped working properly, instantly stopping execution shortly after internal setup of the nodes, without any error message, stacktrace or similar. We first hypothesised that is was due to expending API calls to the Twitter API, but this was soon debunked by speeding up progress on the custom spout in order to verify issue without dependency on the Twitter API.

4 Results

In the end, we could not solve the final issue. We do not know what caused it so suddenly. We attempted several solutions, including re-pulling Summingbird from source control and as stated above, change the spout. This lead us to believe Summingbird stores state in a undocumented location, and something happened to this.

Of what we initially set out to do, the only thing achieved was a survey of Summingbird and its usage of Storm. Nothing of the original plan (Figure 1) was implemented, since the critical component, the Summingbird-based component, never got off the ground due to the various issues discussed above.

5 Conclusions

Our opinions of this project is mixed. While Summingbird seems nice, and many people and companies, especially Twitter seems to get great use and performance from Summingbird, this was not something we found. It might be that when applied to a proper cluster and/or data center these issues do not materialise.

The only explanation we have of the complete disfunction in our first development environment, the Arch Linux installation, is that due to a misunderstanding of the instructions, Storm was installed separately using the package manager, rather than being pulled as a dependency by SBT. Why this would manifest itself in that way is beyond our current understanding but nevertheless a possibility.

If one is to even attempt at using something like Summingbird, one should have previous experience with Big Data and Storm/Hadoop since there is a lot one is expected to know to make use of the system effectively. Not all terminology used is explained in the documentation. Instead one is expected to already know these in advance.

References

1. *Oscar Boykin, Sam Ritchie, Ian O'Connell, and Jimmy Lin.* 2014. Summingbird: A Framework for Integrating Batch and Online MapReduce Computations. In Proceedings of the VLDB Endowment, Volume 7, 2013-2014.
2. Hadoop homepage, https://hadoop.apache.org/, fetched 2015-05-27
3. Storm homepage, https://storm.apache.org/, fetched 2015-05-27