SUNY POLYTECHNIC INSTITUTE

Apache Apex
Development Docker Image

Alexander Jansing

Computer Science Department
State University of New York,
Polytechnic Institute
Utica, NY 13502
USA
jansina@sunyit.edu

12 December 2018

Abstract: Apache Apex is Hadoop YARN-native framework for building distributed applications and applies native streaming to the data processing pipeline [3]. The Apex project was mainly been driven by the company DataTorrent. DataTorrent shut its doors back in May of 2018[2][7].

This project is designed to help other developers write their own Apache Apex applications by providing an Docker image with Apex, JDK8, and Maven installed. The start-up script provided can be easily changed so that others can pass their project directory as a volume to a container and rapid testing can be performed.

I. Introduction

A. Apache Apex and Mahlar

Apache Apex is Hadoop YARN-native framework for building distributed applications and applies native streaming to the data processing pipeline [3]. The Apex project was mainly been driven by the company DataTorrent. DataTorrent shut its doors back in May of 2018[2][7]. The last release of Apex was put out on April 27^{th} of the same year; which wasn't that long ago as of the writing of this paper, but looking at the related github repositories and Apache Jira board doesn't do much to inspire confidence in future releases.

Apex comes with Malhar, a codec library of operators used to enable users to speed up their work and in a variety of languages (Java, JavaScript, Python, R, Ruby). Malhar operations are based off of templates and can be easily extended to perform the actions you need. Additionally, Malhar allows the developer to change the properties of an operator at runtime, so the application doesn't need to be brought offline[6].

This project is designed to help other developers write their own Apache Apex applications by providing an Docker image with Apex, JDK8, and Maven installed. The start-up script provided can be easily changed so that others can pass their project directory as a volume to a container and rapid testing can be performed (see V-A2).

B. Structure of the Paper

The summary of the work performed this semester will be laid out as follows:

1) the research involved in working with Apache Apex and its potential use case,
2) how Apache Apex was used; including
   a) what was done,
   b) and the challenges faced.
3) and finally a reflection on what was and was not accomplished.

II. Research

A. Process

Whenever researching a piece of technology to use, especially when software under Apache Software Foundation (*ASF*), plenty of documentation is available on the project's site. Usually, there is enough information there to get started in understanding how and why to use a particular product. Usually one should look beyond the product's page when the software isn't under the ASF; as many companies are looking to inflate their egos or sell a license.

In the case of this project, I used the Apache Apex project page [1], the provided ReadTheDocs pages[4] [**?**] [6], a presentation from a DataTorrent employee, Thomas Weise [3], Wikipedia and its references [2] [7] to help me understanding how the framework was designed to work. Since Apex is covered under the ASF and its primary driving company, DataTorrent, went under in May 2018, I wasn't too concerned with being convinced to buy a license.

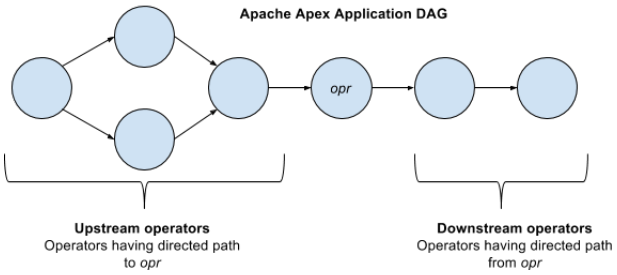

Fig. 1. Apex Application DAG and Operators [5]

B. Use Case

As was stated in the introduction, Apache Apex is Hadoop YARN-native framework for building distributed applications and applies native streaming to the data processing pipeline [3]. YARN-native means that applications developed using the Apache Apex API can be deployed directly onto an HDFS clustering using YARN without installing any new software [8].

Apache Apex can be thought of as an alternative to Apache Spark for performing analytics. It appears that Apex was inspired, at least partially, by Spark as seen by its use of a *DAGs* (Directed acyclic graphs) pattern of arranging code (see III-A).

Another of Apache Apex's lesser known features is that it supports automatic scaling with the demands placed upon itself [8]. The ability to automatically scale an application up or down based on the current needs of a system is a great feature for any framework or application to have and it a major selling point for some other software out there, i.e. Cloud Foundry [9].

Apache Apex's primary use case seems to take advantage of an existing HDFS cluster, using YARN (Mesos support was never released [10]), to perform streaming analytics; i.e. training LSTMs (Long short-term memory neural networks) or reading data that continually streams in from a source.

III. Using Apex

A. Quick Overview of the Apache Apex API

The Apache Apex API consists of two primary units; *Operators* and DAG. Operators are the most basic units of Apex applications and the vertices of the DAG. The DAG object the graph representation of the Apex application and describes how data streams in and out of Operators (Figure 1)[5].

B. Work Accomplished

During my survey of Apache Apex, I worked my way through some of the examples in the apex-malhar (see V-B2), primarily the WordCount example. Using the WordCount example as a starting point, I wrote my own Operators (WordCountInput and MapReducerFileOutputOperator (V-A1)) and added them to my application's DAG.

In addition of the basic word counting application, I expanded on the apex-sandbox (V-B3) to create apex-maven

(V-A2). The apex-maven image is a developer Docker image that eliminates the intermediate need to build your project locally, then go into the apex-sandbox to run your project. Instead, the developer can now build Apex projects from within an apex-maven container and launch it within the same terminal window.

The apex-maven image uses apex-sandbox as a starting point and further includes JDK8, Maven, and the apex-malhar git repository, built; meaning that the Malhar library code is included in the image's *~/.m2/repository/*.

IV. Conclusion

A. Challenges

Working with a new framework is always challenging. It gets easier the more frameworks you've been exposed to, but there is always a learning curve. Apache Apex follows this pattern. An overview of how Apex's applications was briefly explained in III-A. More could have been learned in the pursuit of this project, but Apex's framework was much easier to understanding since I am familiar with reading the documentation associated with Apache Foundation project and Apex shares some of its DAG pattern with Spark.

B. Reflection

For this project, I had originally intended to perform a series of work count analytics and later translate those into byte-array counts; via the shingling of the bytes as they streamed into the source Operation vertex. Then, I wanted to find combinations of bytes that did not occur within a given file and replace the shingles to compress the file; providing the series of replacements after the EOF of the original file as a way to reverse the compression.

This project idea was a bit too ambitious to complete given that I needed to learn a whole new framework in the processing and the time provided. While running through the WordCount examples and trying to write my own Operators inspired by others I found (credited in code), I realized I wasn't going to be able to finish the project in time. With that in mind, I decided to get the word-count MapReduce DAG in working order and expand on the apex-sandbox Docker image provided by the Apache Apex project (V-B2) to create an image that developers could use to speed up testing of Apex applications.

V. Appendix

A. Code snippets and links

1) MapReduce: https://github.com/apjansing/apex-maven/tree/master/src/MapReduce

2) apex-maven:

- https://github.com/apjansing/apex-maven/blob/master/src/docker/Dockerfile
- https://hub.docker.com/r/apjansing/apex-maven/

3) pull_and_start_dev_apex_docker.sh:

```
#!/bin/bash
JANSING_PROJECT_HOME=`(git rev-parse
    ↪ --show-toplevel)`
JANSING_REPO_NAME=`(basename
    ↪ $JANSING_PROJECT_HOME)`

docker pull apjansing/apex-maven
docker run -it --name=apex -v
    ↪ $JANSING_PROJECT_HOME:/home/
    ↪ apex/$JANSING_REPO_NAME -p
    ↪ 50070:50070 -p 8089:8088 apex-maven
```

4) MapReduce Output:

```
{monstrous=1, allah=1, xlv=1, xxxvii=1,
    ↪ bag=1, dutch=1, xxiv=1, vii=3,...},
{... cheerless=1, monstrous=4, flinty=1,
    ↪ been=12, gaping=1, ...},
{... ticking=1, monstrous=1, been=64,
    ↪ mostly=2, ...},
{... account=5, monstrous=6, been=60,
    ↪ unquestion=1, ...}
```

B. Source locations

1) Apex Documentations:

- http://apex.apache.org/docs/apex
- http://apex.apache.org/docs/malhar
- http://apex.apache.org/docs/apex/apex_development_setup/#creating-new-apex-$\cdot$project

2) Github Repositories:

- https://github.com/apjansing/apex-maven
- https://github.com/apache/apex-core
- https://github.com/apache/apex-malhar
- https://github.com/apache/apex-site

3) Apache Apex Downloads:

- http://apex.apache.org/downloads.html
- https://hub.docker.com/r/apacheapex/sandbox/

References

[1] Foundation, A. (2018). Apache Apex. [online] Apex.apache.org. Available at: https://apex.apache.org/ [Accessed 5 Nov. 2018].

[2] En.wikipedia.org. (2018). Apache Apex. [online] Available at: https://en.wikipedia.org/wiki/Apache_Apex[Accessed10Nov.2018].

[3] Weise, T. (2016). Architectual Comparison of Apache Apex and Spark Streaming. [online] Available at: https://www.slideshare.net/ApacheApex/architectual-comparison-of-apache-apex-and-spark-streaming [Accessed 10 Nov. 2018].

[4] Dt-docs.readthedocs.io. (2018). Beginner's Guide - DataTorrent Documentation. [online] Availableat:https://dt-docs.readthedocs.io/en/latest/beginner/ [Accessed 10 Nov. 2018].

[5] Apex.apache.org. (2018). Apache Apex Documentation. [online] Available at: http://apex.apache.org/docs/apex/ [Accessed 10 Nov. 2018].
[6] Apex.apache.org. (2018). Apache Apex Malhar Documentation. [online] Available at: http://apex.apache.org/docs/malhar [Accessed 10 Nov. 2018].
[7] Leopold, G. (2018). DataTorrent, Stream Processing Startup, Folds. [online] Datanami. Available at: https://www.datanami.com/2018/05/08/datatorrent-stream-processing-startup-folds/ [Accessed 10 Nov. 2018].
[8] Weise, T. (2018). Stream Processing use cases and applications with Apache Apex | Big Data Spain. [online] Bigdataspain.org. Available at: https://www.bigdataspain.org/2016/program/fri-stream-processing-use-cases-applications-with-apache-apex.html [Accessed 12 Dec. 2018].
[9] Yang, B. (2018). cloudfoundry-incubator/app-autoscaler. [online] GitHub. Available at: https://github.com/cloudfoundry-incubator/app-autoscaler#deploy-and-offer-auto-scaler-as-a-service [Accessed 12 Dec. 2018].
[10] Issues.apache.org. (2018). Apache Apex Core - ASF JIRA. [online] Available at: https://issues.apache.org/jira/projects/APEXCORE/summary [Accessed 12 Dec. 2018].