Functional Programming and Big Data

by Glenn Engstrand (September 2014)

http://glennengstrand.info/analytics/fp

What is Functional Programming? It is a style of programming that emphasizes immutable state, higher order functions, the processing of data as streams, lambda expressions, and concurrency through Software Transactional Memory.

What is Clojure? That is a Functional Programming language similar to Lisp that runs in the Java Virtual Machine. Additional features include; dynamic programming, lazy sequences, persistent data structures, multi-method dispatch, deep Java interoperation, atoms, agents, and references.

In a previous blog, I shared a learning adventure where I wrote a news feed service in Clojure and tested it, under load, on AWS. One of the biggest impressions that Clojure made on me was how its collection oriented functions like union, intersection, difference, map, and reduce made it well suited for data processing. I wondered how well that this data processing suitability would translate to a distributed computing environment.

http://clojure.org/

Recently, I attended OSCON 2014 which is a tech conference that focuses on open source software. One of the tracks was on Functional Programming. Two of the presentations that I attended talked about big data and mentioned open source projects that integrate Clojure with Hadoop. This blog explores functional programming and Hadoop by evaluating those two open source projects.

What is Big Data? Systems whose data flows at a high volume, at fast velocity, comes in a wide variety, and whose veracity is significant are attributed as Big Data. Usually this translates to petabytes per day.

Big data must be processed in a distributed manner. Big data clusters usually scale horizontally on commodity hardware. The Apache Hadoop project is the most popular enabling technology. A typical Big Data project is built with either Cloudera or Hortonworks offerings. Usually OLAP serves as the front end where the processed data is analyzed.

http://glennengstrand.info/software/architecture/oss/clojure

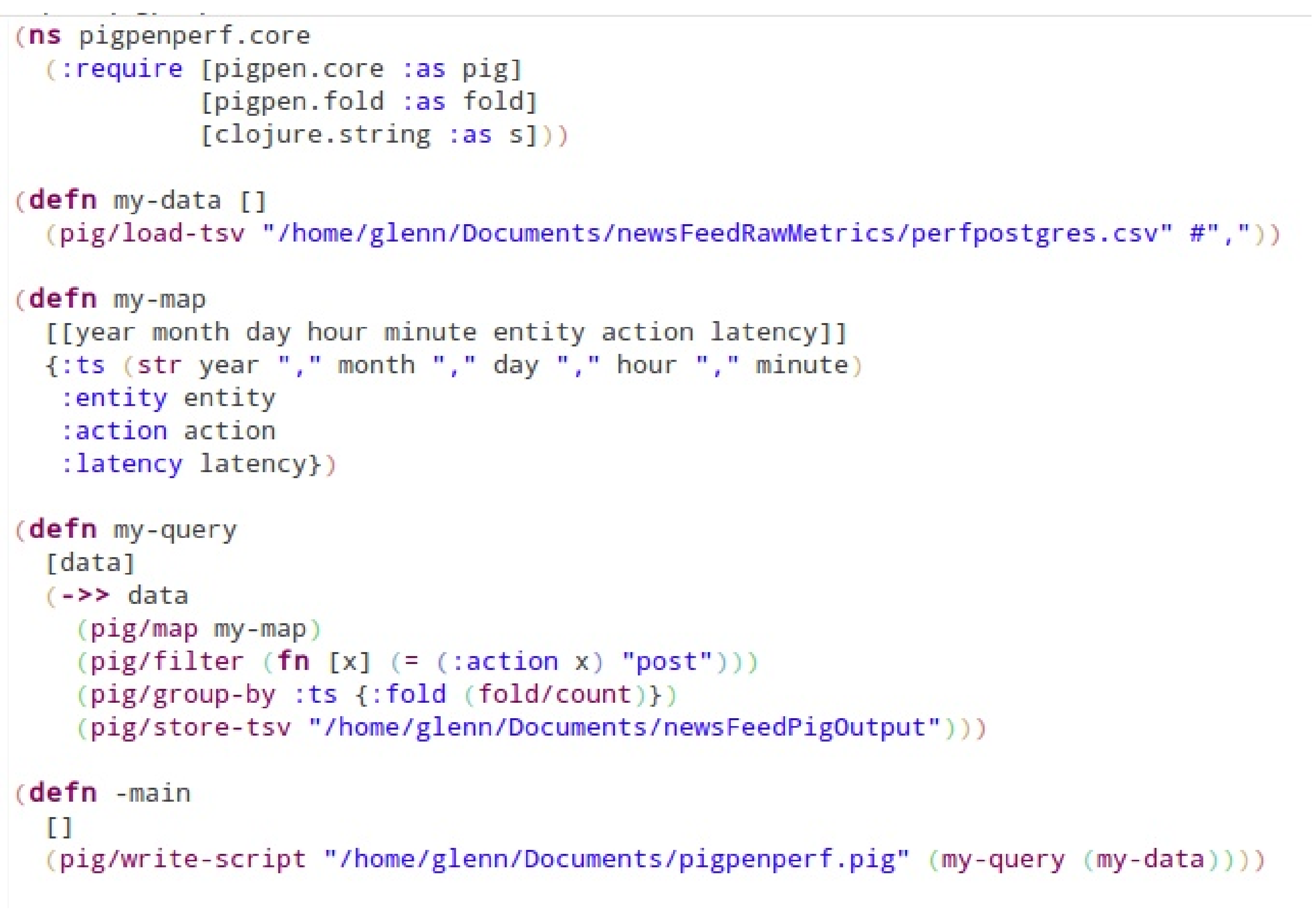
In one of those presentations, entitled Big Data Pipeline and Analytics Platform Using NetflixOSS and Other Open Source Libraries, two engineers at Netflix talked about a lot of open source projects that they use for their big data pipe. One of those projects was called Pig Pen.

For anyone with experience with report writers or SQL, Pig Pen will look like it is pretty easy to understand. Commands like load-tsv, map, filter, group-by, and store seem pretty intuitive at first glance.

https://github.com/Netflix/PigPen/

From an architectural perspective, Pig Pen is an evil hack. Your Clojure code gets interpreted as a User Defined Function in a Pig script that loads and iterates through the specified input file. First, you have to run your code to generate the script. Then, you have to remove the part of the project specification that identifes the main procedure and compile your code to create an uberjar and copy it to a place where that Pig script can load it.

That, and the fact that this was the slowest running and worse documented project that I evaluated, makes Pig Pen pretty much a non starter as far as I am concerned.



```
(ns pigpenperf.core
  (:require [pigpen.core :as pig]
            [pigpen.fold :as fold]
            [clojure.string :as s]))

(defn my-data []
  (pig/load-tsv "/home/glenn/Documents/newsFeedRawMetrics/perfpostgres.csv" #","))

(defn my-map
  [[year month day hour minute entity action latency]]
  {:ts (str year "," month "," day "," hour "," minute)
   :entity entity
   :action action
   :latency latency})

(defn my-query
  [data]
  (->> data
    (pig/map my-map)
    (pig/filter (fn [x] (= (:action x) "post")))
    (pig/group-by :ts {:fold (fold/count)})
    (pig/store-tsv "/home/glenn/Documents/newsFeedPigOutput")))

(defn -main
  []
  (pig/write-script "/home/glenn/Documents/pigpenperf.pig" (my-query (my-data))))
```

Big Data Pipeline and Analytics Platform Using NetflixOSS and Other Open Source Libraries
http://www.oscon.com/oscon2014/public/schedule/detail/34159

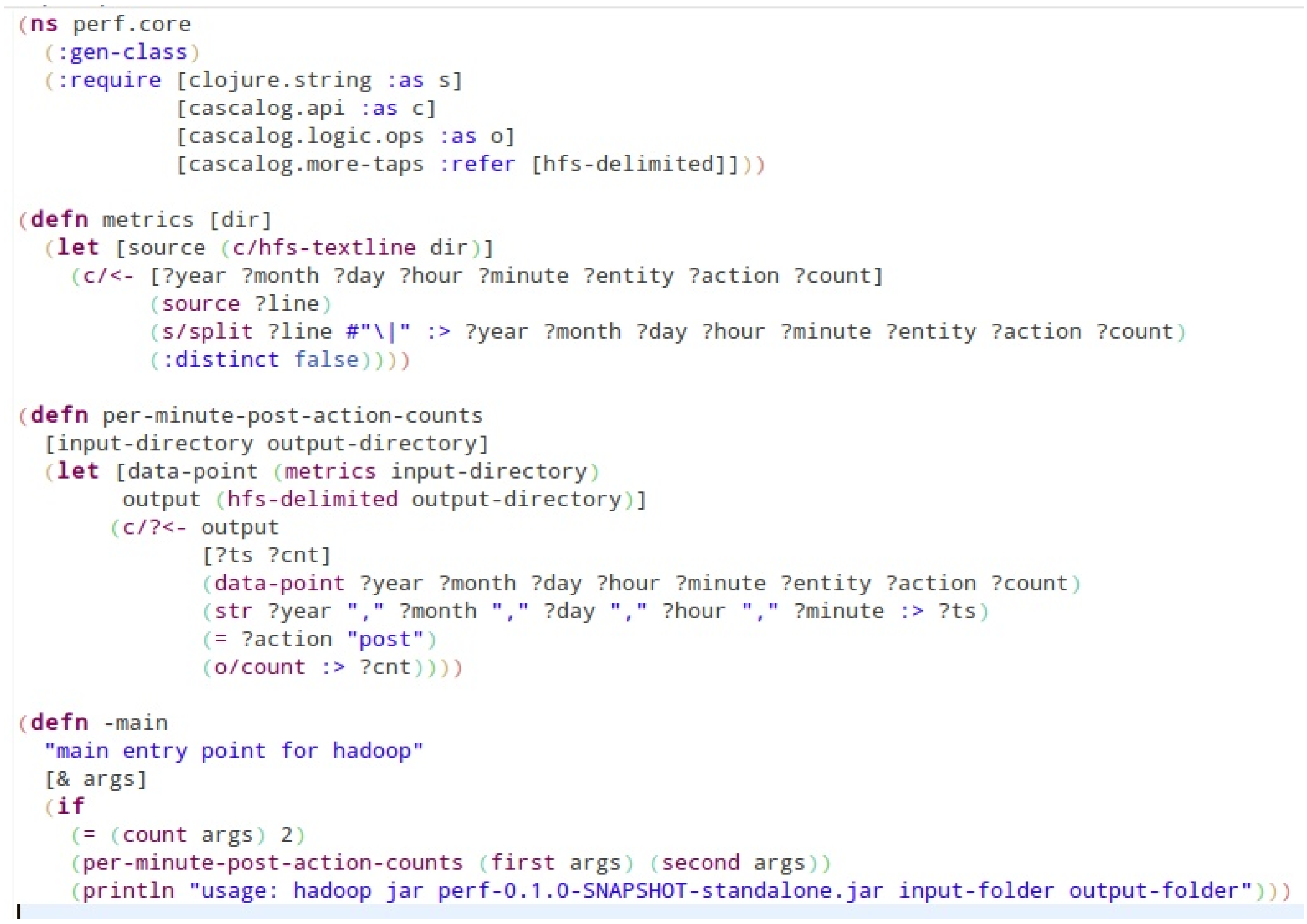
Cascalog was one of the recommended open source projects that integrates Clojure with Hadoop. Cascalog is built on top of Cascading which is another open source project for managing ETL within Hadoop.

http://cascalog.org/

Cascalog takes advantage of Clojure homoiconicity where you declare your query to be run later in a multi-pass process. Many things, such as grouping are not explicitly specified. The system just figures it out. Order of statements doesn't matter either.

http://www.cascading.org/

At first glance, Cascalog looks like it is ideomatic Clojure but it really isn't. It is declarative instead of imperative. It looks to be pure Functional Programming but, under the covers, it isn't. More on this in a future blog.



```
(ns perf.core
  (:gen-class)
  (:require [clojure.string :as s]
            [cascalog.api :as c]
            [cascalog.logic.ops :as o]
            [cascalog.more-taps :refer [hfs-delimited]]))

(defn metrics [dir]
  (let [source (c/hfs-textline dir)]
    (c/<- [?year ?month ?day ?hour ?minute ?entity ?action ?count]
          (source ?line)
          (s/split ?line #"\|" :> ?year ?month ?day ?hour ?minute ?entity ?action ?count)
          (:distinct false))))

(defn per-minute-post-action-counts
  [input-directory output-directory]
  (let [data-point (metrics input-directory)
        output (hfs-delimited output-directory)]
       (c/?<- output
              [?ts ?cnt]
              (data-point ?year ?month ?day ?hour ?minute ?entity ?action ?count)
              (str ?year "," ?month "," ?day "," ?hour "," ?minute :> ?ts)
              (= ?action "post")
              (o/count :> ?cnt))))

(defn -main
  "main entry point for hadoop"
  [& args]
  (if
    (= (count args) 2)
    (per-minute-post-action-counts (first args) (second args))
    (println "usage: hadoop jar perf-0.1.0-SNAPSHOT-standalone.jar input-folder output-folder")))
```

Data Workflows for Machine Learning

http://www.oscon.com/oscon2014/public/schedule/detail/34913

Apache Spark is a relatively new project that is gaining a lot of traction in the Big Data world. It can integrate with Hadoop but can also stand alone. The only Functional Programming language supported by Spark is Scala.

http://www.scala-lang.org/

If you are coming to Scala from Java or just have little FP experience, then what you might notice the most about Scala is lambda expressions, streams, and for comprehensions. If you are already familiar with Clojure, then what you might notice the most about Scala is strong typing.

https://spark.apache.org/

Spark has three main flavors; map reduce, streaming, and machine learning. What I am covering here is the map reduce functionality. I used the Spark shell which is more about learning and interactively exploring locally stored data.

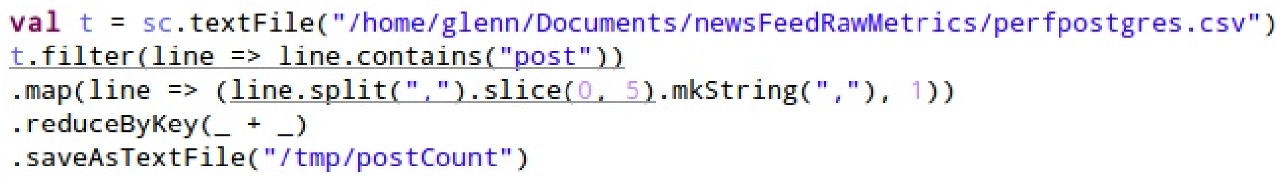
Spark was, by far, the fastest technology that took the fewest lines of code to run the same query. Like Pig Pen, Map Reduce Spark was pretty limited on reduce side aggregation. All I could do was maintain counters in order to generate histograms. Cascalog had much more functional and capable reduce side functionality.

What is reduce side aggregation?

There are two parts to the map reduce approach to distributed computing, map side processing and reduce side processing. The map side is concerned with taking the input data in its raw format and mapping that data to one or more key value pairs. The infrastructure performs the grouping functionality where the values are grouped together by key. The reduce side then does whatever processing it needs to do on that per key grouping of values.

Why is map reduce Spark more limited in its reduce side aggregation capability than Cascalog?

In the map reduce flavor of Apache Spark, the reducer function takes two values and is expected to return one value which is the reduced aggregation of the two input values. The infrastructure then keeps calling your reducer function until only one value is left (or one value per key depending on which reduce method you use). The reducer function in Cascalog gets called only once for each key but the input parameter is a sequence of tuples, each tuple representing the mapped data that is a value associated with that key.



```
val t = sc.textFile("/home/glenn/Documents/newsFeedRawMetrics/perfpostgres.csv")
t.filter(line => line.contains("post"))
.map(line => (line.split(",").slice(0, 5).mkString(","), 1))
.reduceByKey(_ + _)
.saveAsTextFile("/tmp/postCount")
```

I wrote the same job, generating the same report, in all three technologies but Spark and Scala was the fastest. Is that a fair comparison? You have to run both the Pig Pen job and the Cascalog job in Hadoop. For my evaluation, I was using single node mode. Spark is different. You don't run it inside of Hadoop and I am pretty sure that it isn't doing much or anything with Hadoop locally.

Large Spark map reduce jobs would have to integrate with Hadoop so this may not be a fair comparison.

But the real test with any technology isn't how you can use it but rather how you can extend its usefulness. For Cascalog, that comes in the form of custom reduce side functions. In this area, Pig Pen was completely broken.

In the next blog, I will review my findings when I reproduced the News Feed Performance map reduce job, used to load the Clojure news feed performance data into OLAP, with a custom function written in Cascalog.

In the next blog, I will also cover how Spark streaming can be made to reproduce the News Feed Performance map reduce job in real-time.

In this article, I explored Functional Programming in Big Data by evaluating three different FP technologies for Big Data. I did this by writing the same Big Data report job in those technologies then comparing and contrasting my results. Hive is still the most popular way to aggregate Big Data in Hadoop. What would that job, used to evaluate those technologies, have looked like in Hive? First, you would have had to load the raw text data into HDFS.

```
bin/hadoop dfs  -put perfpostgres.csv /user/hive/warehouse/perfpostgres
```

Then you would have had to define that raw text data to Hive.

```
create external table news_feed_perf_raw (year string, month string, day string,
hour string, minute string, entity string, action string, duration string) row
format of delimited fields terminated by ',' lines terminated by '\n' stored as
textfile location '/user/hive/warehouse/perfpostgres';
```

Here is the map part of the processing.

```
insert overwrite local directory 'postActions' select concat(year, ',', month,
',', day, ',', hour, ',', minute) as ts from news_feed_perf_raw where action =
'post';
```

Here is the reduce part of the processing.

```
insert overwrite local directory 'perMinutePostActionCounts' select ts, count(1)
from postActions group by ts;
```