

liveanal 0.1.0-SNAPSHOT

An investigation into a day's worth of IceCube Live message traffic

dependencies

org.clojure/clojure	1.5.1
org.clojure/data.json	0.2.3
incanter/jfreechart	1.0.13-no-gnujapx

namespaces

[liveanal.core](#)

liveanal.core toc

[Clojure bookkeeping ...]

First look at a day's worth of messages

I've unpacked one day's worth of SPADE Priority 3 files from the Data Warehouse (everything from `/data/exp/IceCube/2013/internal-system/I3Live/1120/I3Live-Msgs-Prio3*`). These files, once unzipped/untarred, should have all messages for the entire day.

We want to see the relative abundance of messages by service name.

This is a very short Clojure or Python program:

```
import os
import toolz
import json

def messages_from_file(fname):
    with file(fname) as ff:
        return json.loads(ff.read())

def gen():
    for fname in os.listdir("../resources"):
        if not fname.startswith("I3Live"):
            continue
        for m in messages_from_file("../resources/" + fname):
            yield m["service"]

print toolz.frequencies(gen())

print "OK"
```

The Clojure code is enclosed in a `comment` expression, since this is a [literate program](#) which we may choose to execute later.

Execution time is **6.8 seconds in Clojure**, 53 seconds in Python. The 13 messages that have `nil` (in Python, `None`) as a service name are alerts (not user-alerts) and comments.

(this space intentionally left almost blank)

```
(ns liveanal.core
  (:require [clojure.data.json :as json]
            [clojure.pprint :refer [pprint]]))
```

```
(comment
  (frequencies (for [file (-> "/Users/jacobsen/Desktop/liveanal/resources/"
                             clojure.java.io/file
                             file-seq)
                    :when (.isFile file)
                    message (-> file
                                clojure.java.io/reader
                                json/read)]
                (message "service"))))
```

```
{nil 13,
 "I3DAQDispatch" 11283,
 "pdaq" 29835,
 "OpticalFollowUp" 39762,
 "PFRawWriter" 10220,
 "I3MoniPhysA" 287,
 "TFRateMonitor" 479,
 "uptimer" 1696,
 "sn-email" 12,
 "I3MoniDomMon" 288.}
```

Refactoring the above code slightly, we pull out a function to extract all messages from the files in our directory.

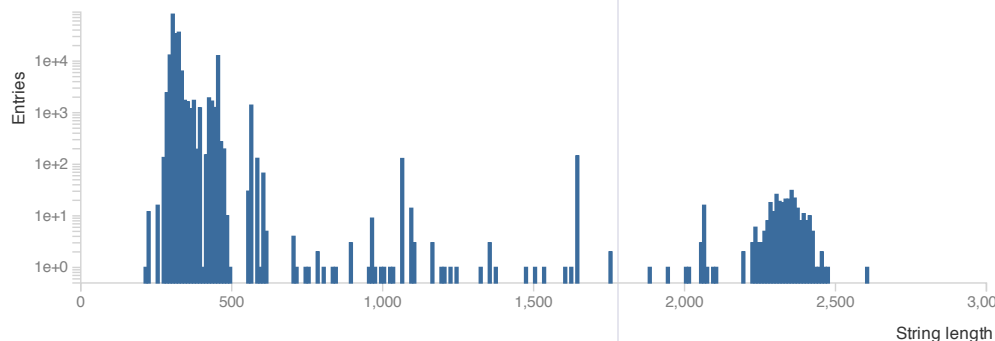
the total number of messages is just

=> 201311

Now we want to take a first look at message lengths. We use the histogramming function taken from [this blog post](#)....

Convert the output to JavaScript

And use [i3d3](#) to show the distribution of lengths.



```

"livecontrol" 2595,
"diskmon-expcont" 192,
"GammaFollowUp" 39752,
"PFFiltWriter" 15200,
"temperature" 882,
"I3MoniDomTcal" 287,
"I3MoniDomSn" 287,
"meteorology" 144,
"PFServer1" 8508,
"PFServer2" 8474,
"sndaq" 1037,
"PFServer3" 8460,
"I3MoniMover" 667,
"HSiface" 9713,
"PFFiltDispatch" 2165,
"PFServer4" 8449,
"DB" 624}

(defn day-msgs [dirname]
  (for [file (-> dirname
                  clojure.java.io/file
                  file-seq)
        :when (.isFile file)]
    (message (-> file
                  clojure.java.io/reader
                  json/read))
    message))

(defn dirname "/Users/jacobsen/Desktop/liveanal/resources/")

(comment
  (count (day-msgs dirname)))

(defn make-hist
  [xmin xmax nbins xs]
  (let [;; "base" histogram (zeros):
        zero-map (into (sorted-map)
                        (map (fn [x] [x 0]) (range nbins))))
        ;; get actual bin values for every input in xs:
        xbins (map #(int (* nbins (/ (- % xmin)
                                     (- xmax xmin))))
                    xs)
        ;; strip out underflows & overflows:
        no-overflows (-> xbins
                          (remove #(< % 0))
                          (remove #(>= % nbins)))]
    ;; yield histogram as array of [ibin, height] pairs:
    (into [] (reduce #(update-in %1 [%2] inc) zero-map no-overflows))))

(defn js-vec [v]
  (apply str (concat "[" (interpose "," v) "]")))

(comment (-> dirname
              day-msgs
              (map (comp count str))
              (make-hist 0 3000 300)
              (map second)
              to-js
              println))

```