Large(ish) Data

Working the Middle Ground Between RAM
and a Cluster

LA Clojure Meetup Sept 13 2017

When RAM is no Longer Enough

Random Sampling

Duplicate Detection

Summary Counts

Mission: Probable

20k Random Sample

Population size: ~400 Million

I know I'll use the Database...

```
SELECT *
FROM some_huge_table
ORDER BY RANDOM()
LIMIT 20000;
```

HELLO
my name is

Run Time

GNU Tools

```
$ sort -r in.tab > out.tab
$ head -n 20000 out.tab
```

Flip a Coin?

Stream Sampling

$Pr(e) = N / M$

N = Remaining Sample Size

M = Remaining Population Size

Stream Sampling

Decrement N when a sample is taken

Decrement M for every element

Stream Sampling

$N = 20{,}000$

$M = 400{,}000{,}000$

$Pr(e) = 0.00005$

Stream Sampling

- 1 N=2 M=7 Pr(e)=0.285 MISS
- 2 N=2 M=6 Pr(e)=0.333 HIT
- 3 N=1 M=5 Pr(e)=0.20 MISS
- 4 N=1 M=4 Pr(e)=0.25 MISS
- 5 N=1 M=3 Pr(e)=0.33 MISS
- 6 N=1 M=2 Pr(e)=0.50 MISS
- 7 N=1 M=1 Pr(e)=1.0 HIT [DONE]

Stream Sampling

```
(with-open [rdr (io/reader "phones.txt")]
  (random-sample-seq
   (line-seq rdr)
   400000000                    ; population size
   20000))                      ; sample size
```

- Sample in One Pass
- Disk IO is 1x
- O(reasonable)

Related Algorithm

Reservoir Sampling

for when you don’t know your population size

Reservoir Sampling

```
(defn find-dupes-reservoir [ifname]
  (with-open [ifh (io/file ifname)]
    (let [sampler (make-reservoir 200)]
      (sampler (line-seq ifh)))))
```

Stream vs Reservoir

Stream Sampling

- Pro: Memory Efficient
- Con: Must Know Population Size a priori

Reservoir Sampling

- Con: Must Hold Reservoir in Memory
- Pro: Works on Unknown Population Size

Questions?
(time?)

Next Lurking Issue?

HELLO
my name is

Default

aka “Dummy”

- ‘NULL’ / ‘N/A’
- (610) 555-1212
- na@na.com
- 123 Main St.
- John Q. Public

Naive Counting

```
(defn find-dupes-naive [inp-seq]
  (reduce
    (fn [counts item]
      (assoc
        counts
        item
      (inc (get counts item 0))))
    {}
    inp-seq))
```


HELLO
my name is
OOM

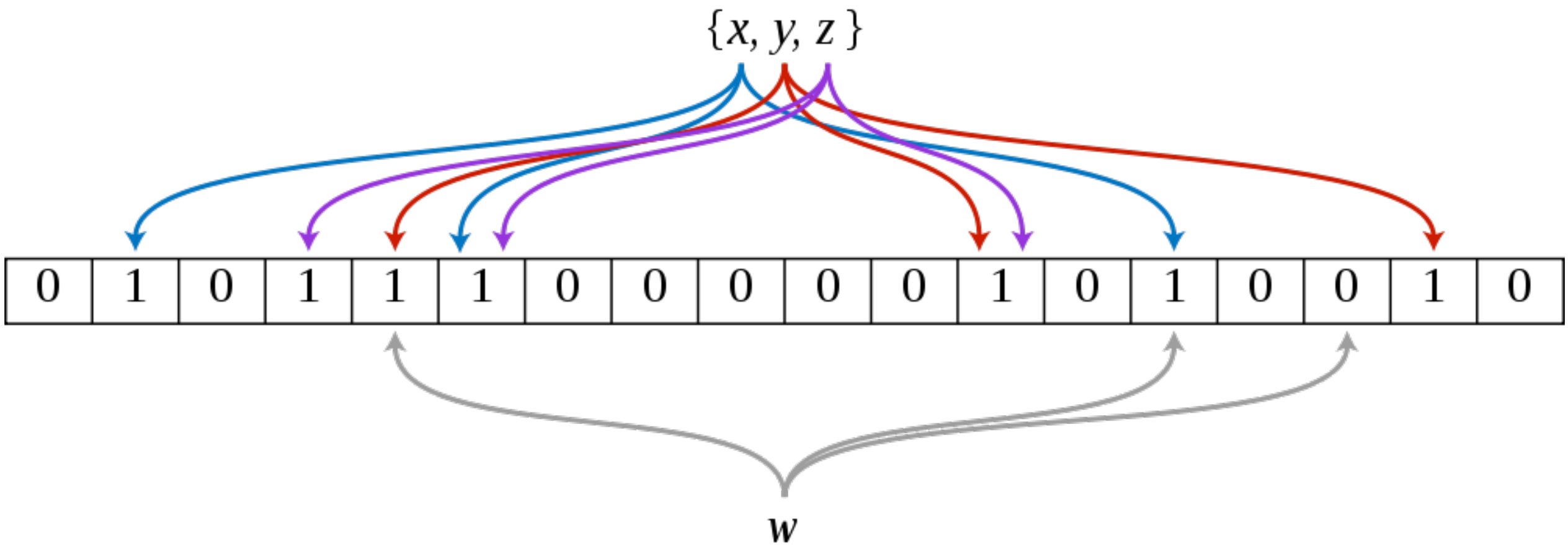
GNU Tools

```
cut -f2 | \
sort | \
uniq -c | \
grep -v '  l ' sort -nr > counts.txt
```

Is This The Only Way?

Bloom Filters

Probabilistic Set



There's Some Math...

$$\left(1-\left[1-\frac{1}{m}\right]^{kn}\right)^k \approx \left(1-e^{-kn/m}\right)^k.$$

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

$$\frac{m}{n}\ln 2 \approx \frac{9m}{13n} \approx 0.7\frac{m}{n},$$

$$\left(1-e^{-k(n+0.5)/(m-1)}\right)^k.$$

It's a Bloomin' Dupe

```
(defn find-dupes [inp-seq psize fp-rate]
  (let [flt (bloom/make-optimal-filter psize fp-rate)]
    (reduce
      (fn [counts item]
        (if-not (bloom/include? flt item)
          (do
            (bloom/add! flt item)
            counts)
          (assoc counts
            item
            (inc (get res item 0)))))
      {}
      inp-seq)))
```

- **Identify Duplicates in One Pass**
- **Memory Usage is Minimized**
- **Be suspicious of count==1 (FPs)**

Related Algorithm

HyperLogLog

probabilistic cardinality estimation

Questions?
(time?)

Summary Counts

You Know...

NY 14,735

PA 11,234

NJ 8,907

DE 5,191

Smells Like

Embarrassingly Parallel

Smells Like:

Map / Reduce

Can Haz Multi-Core?

Your machine ~~probably~~ has multiple cores

Maybe even multiple CPUs

FP Langs are Supposed to make it easier to leverage these right?

(hint: They do)

Divide and Conquer!

```
split -l 100000 \
  phone-nums-with-lfsr-ids.txt \
  working-dir/inp-

wc -l working-dir/inp-a*
  100000 working-dir/inp-aa
  100000 working-dir/inp-ab
  100000 working-dir/inp-ac
  100000 working-dir/inp-ad
  100000 working-dir/inp-ae
  ...
```

Divide and Conquer!

```
(defn count-area-codes [inp-seq]
  (reduce (fn [m line]
            (let [phnum          (second (.split line "\t"))
                  [_ area-code] (first (re-seq #"\((\d+)\)" phnum))]
              (assoc m area-code (inc (get m area-code 0)))))
          {}
          inp-seq))

(apply
 merge-with +
 (map (fn [inp-file]
        (count-area-codes (ds/read-lines inp-file)))
      (map str
           (filter #(.isFile %)
                   (.listFiles (java.io.File. "working-dir/"))))))
```

Divide and Conquer!

```
(defn cou
  (reduce
                                                  \t"))
                                                  -)\)" phnum))]
                                                   0)))))
          inp-seq))

(appl
 me  e-with +
 (pmap (fn [inp-file]
         (count-area-codes (ds/read-lines inp-file)))
       (map str
            (filter #(.isFile %)
                    (.listFiles (java.io.File. "working-dir/"))))))
```

Did you notice the letter ‘**p**’
I added right there?

Tricked You

No Need to Split

Process ‘blocks’ / ‘chunks’ in parallel

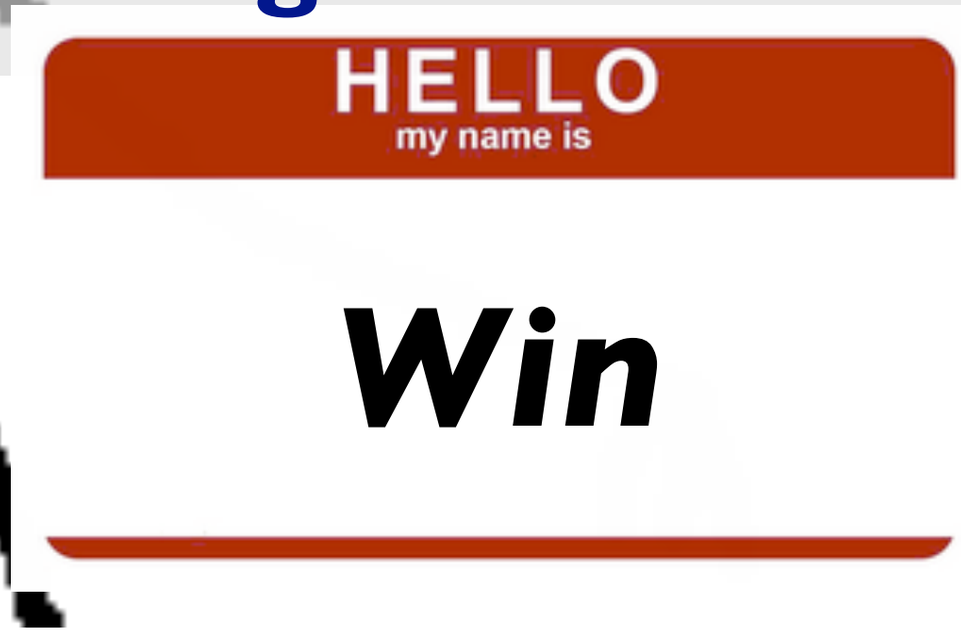
Ensure they’re aligned with line (record) separators.

(I wrapped this up into a library for you)

No Split

```
(reduce
   (fn [res counts]
     (merge-with + res counts))
   (pmap (fn [[start end]]
           (count-area-codes
             (io/read-lines-from-file-segment
              inp-file start end)))
         (partition 2 1
           (io/byte-partitions-at-line-boundaries
             inp-file
             (* 1024 1024)))))
```

- Use Your Cores
- Avoid Unnecessary I/O
- Avoid Additional Disk Usage



This Slide Left
Intentionally Blank

Clojure

Sequence Abstraction

Lazy

Composable

Clojure

Easy Concurrency

pmap

Iterate Faster

Faster Runs let you work out kinks quicker

More Iterations Reduce Bugs

Thank You

@kyleburton

github.com/kyleburton