Data Stream Algorithms in Storm and R

Radek Maciaszek

Who Am I?

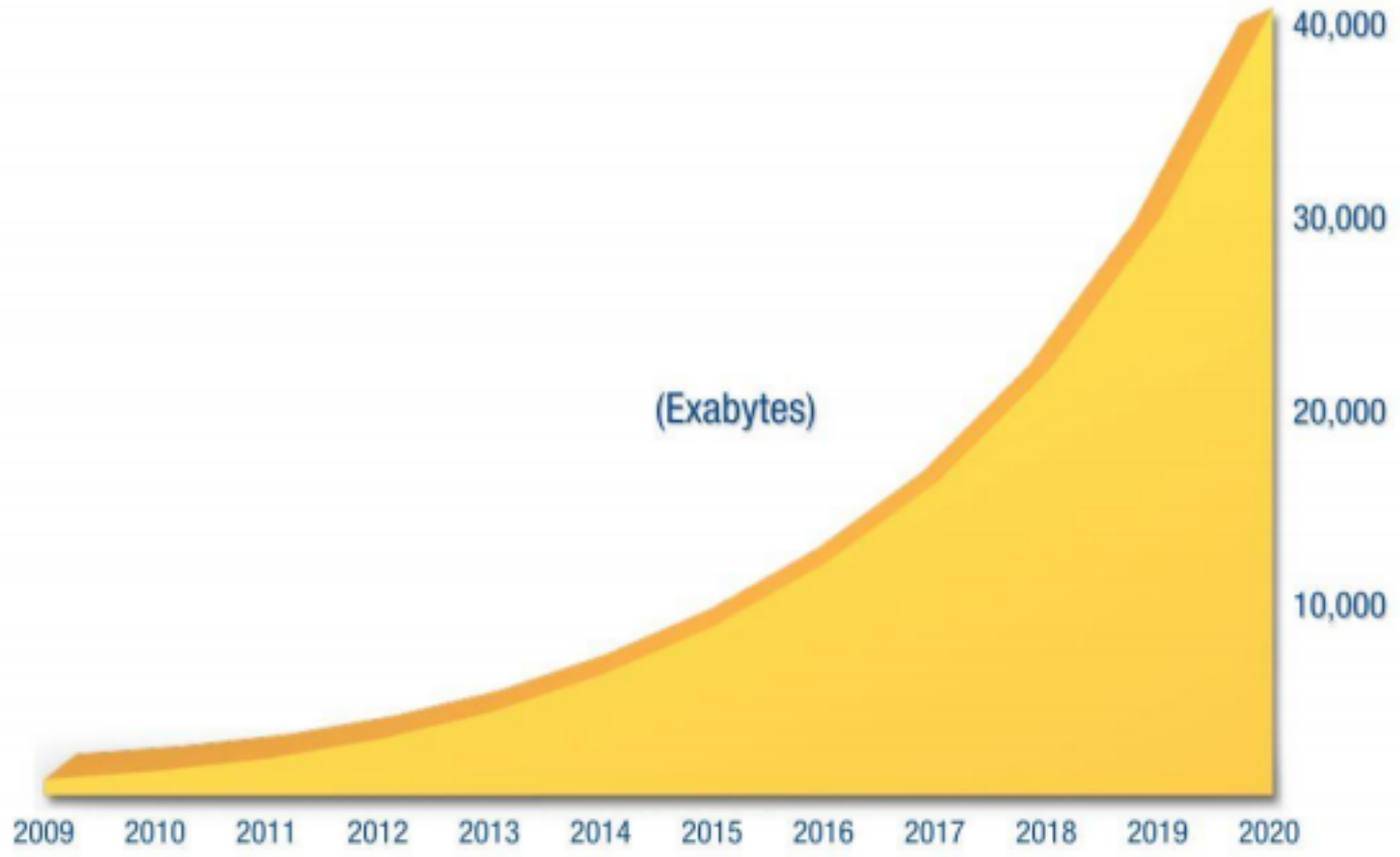
- **Radek Maciaszek**
  - Consulting at DataMine Lab (www.dataminelab.com) - Data mining, business intelligence and data warehouse consultancy.
  - Data scientist at a hedge fund in London
  - MSc in Bioinformatics
  - MSc in Cognitive and Decisions Sciences
  - BSc Computer Science
  - During the career worked with many companies on Big Data and real time processing projects; indcluding Orange, Unanimis, SkimLinks, CognitiveMatch, OpenX, ad4game, eCourier and many others.

Agenda

- Why streaming data?
- Streaming algorithms crash course
- Storm
- Storm + R
- Use Cases

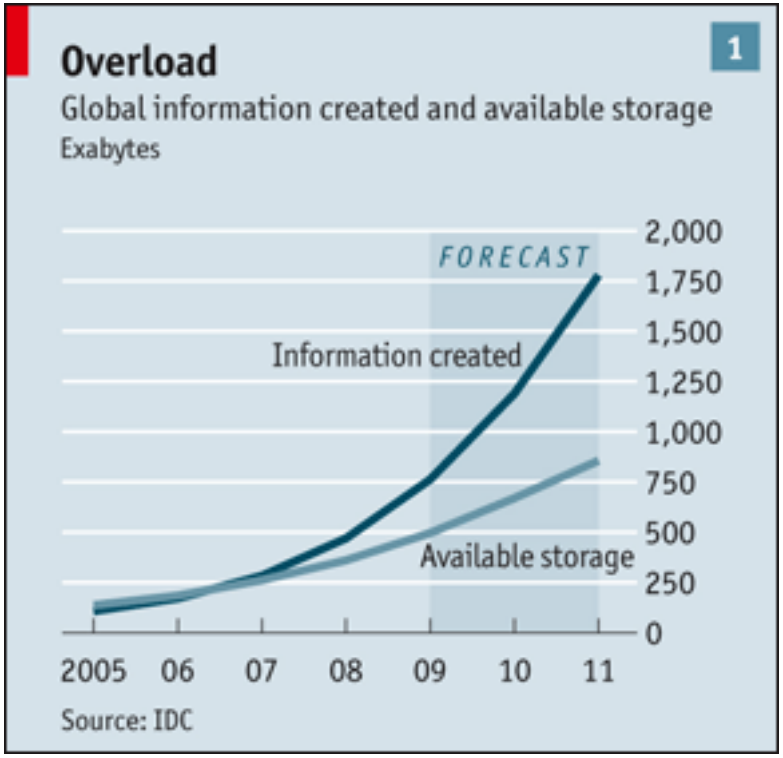
Data Explosion

- Exponential growth of information [IDC, 2012]



Source: IDC's Digital Universe Study, sponsored by EMC, December 2012

Data, data everywhere [Economist]



- "In **2013**, the available storage capacity could hold **33%** of all data. By **2020**, it will be able to store less than **15%**" [IDC, 2014]

Data Streams – crash course

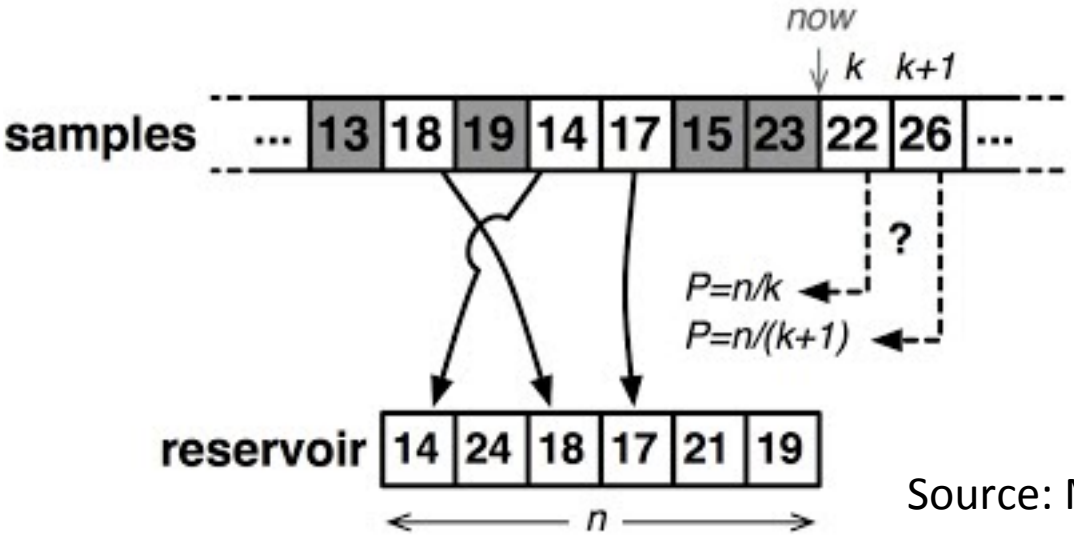
- Reasons to use data streams processing
  - Data doesn't fit into available memory and/or disk
  - Near real-time data processing
  - Scalability, cloud processing
- Examples
  - Network traffic
  - Web traffic (i.e. online advertising)
  - Fraud detection

Use Case – Dynamic Sampling

- OpenX - open-source ad server
- Millions of ad views per hour
- Challenge
  - Create ad samples for statistical analysis. E.g: A/B testing, ANOVA, etc.
  - The data doesn't fit into the memory.
- Solution
  - Reservoir Sampling – allows to find a sample of a constant length from a stream of unknown length of elements

Data Streaming algorithms

- Sampling
  - Use statistic of a sample to **estimate the statistic** of population. The bigger the sample the better the estimate.
  - **Reservoir Sampling** – sample populations, without knowing it's size.
- Algorithm:
  - Store first *n* elements into the reservoir.
  - Insert each k-th from the input stream in a random spot of the reservoir with a probability of n/k (decreasing probability)



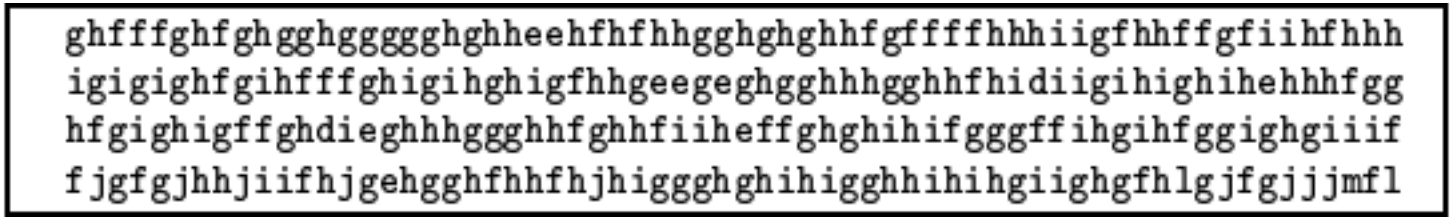
Source: Maldonado, et al; 2011

Use Case - Counting Unique Users

- 100m+ daily visits
- Challenge - number of unique visitors - one of the most important metrics in online advertising
- Hadoop MapReduce. It worked but took long time and much memory.
- Solution:
  - HyperLogLog algorithm
  - Highly effective algorithm to count distinct number of elements
  - See Redis HyperLogLog data structure
- Many other use cases:
  - Cardinality in DB queries optimisation
  - ISP estimates of traffic usage

Cardinality estimation

- Idea: Convert high-dimensional data to a smaller dimensional space. Use lower dimensional image to estimate the function of interest.
- Example: count number of words in all works of Shakespeare
- Naïve solution: keep a set where you add all new words
- Probabilistic counting – 1983. Flajolet & Martin. (first streaming algorithm)
- A better one – LogLog algorithm [Durand & Flajolet; 2003]

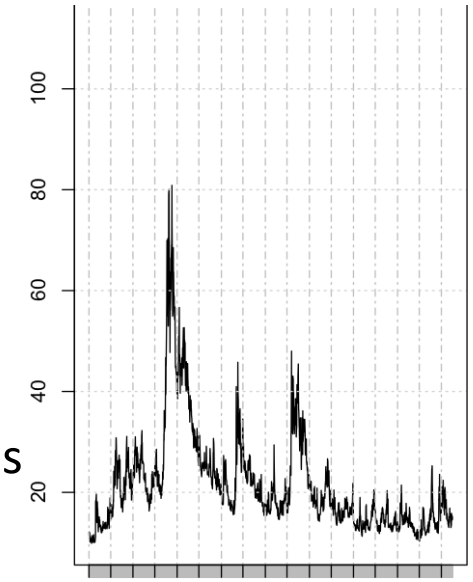


Probabilistic counting

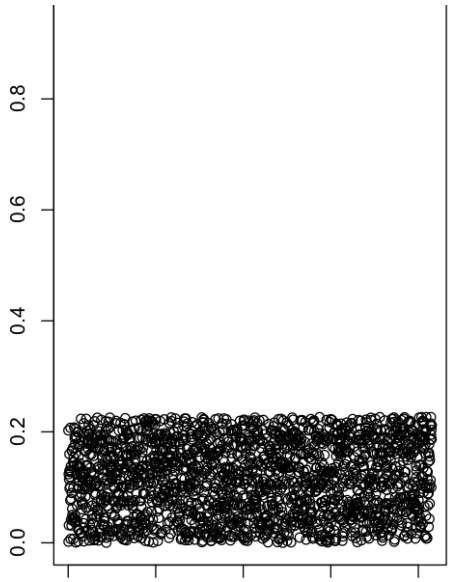
- Transform input data into i.i.d. (independent and identically distributed) **uniform random bits** of information
- Hash(x)= bit1 bit2 …
- Where P(bit1)=P(bit2)=**1/2**
- 1xxx -> P = 1/2, n >= 2
  11xx -> P = 1/4, n >= 4
  111x -> P = 1/8, n >= 8
  n >= 2^{k+1}
- $1^k xx \rightarrow P = \frac{1}{2^{k+1}}$
- Flajolet (1983) estimated the bias

$$E\left[\frac{1}{\phi}2^p\right] = n$$

- Where $\phi \approx 0.7735$ and
  p = position of a first “0”



unhashed



hashed

Source: http://git.io/veCtc

Probabilistic counting - algorithm

- p – calculates position of first zero in the bitmap

```
for i := 0 to L − 1 do BITMAP[i] := 0;
for all x in M do
        begin
        index := ρ(hash(x));
        if BITMAP[index] = 0 then BITMAP[index] := 1;
        end;
```

- Estimate the size using: $\frac{1}{\phi} 2^p$
- R proof-of-concept implementation: http://git.io/ve8la
- Example:

```
> stream = as.integer(runif(10000, 0, 2^n))
> probabilisticCounting(stream)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
> 1/0.7735 * 2^13
[1] 10590.82
```

HyperLogLog

- **LogLog** – instead of keeping track of **all 01s**, keep track only of the **largest 0**
- This will take LogLog bits, but at the cost of lost precision
- Example: **0**0000000 -> 1, 1**0**1**0**0000 -> 4, 111**0**0000 -> 4, 1111**0**01**0** -> 8
- **SuperLogLog** – remove **x%** (typically 70%) of largest number before estimating, more complex analysis
- **HyperLogLog** – harmonic mean of SuperLogLog estimates
- Fast, cheap and 98% correct
- What if we want more sophisticated analytics?

Reference: Flajolet; Fusy et al. 2007

Moving average

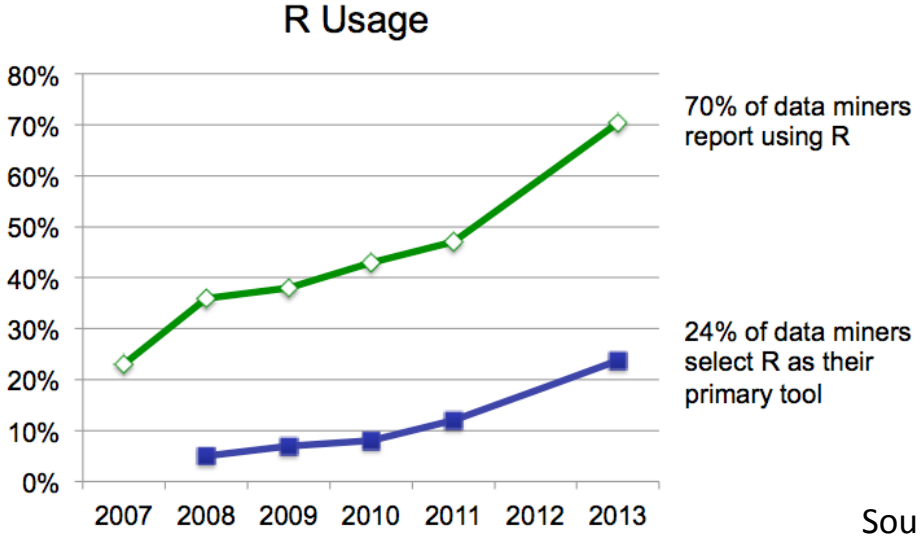
- Example, online mean of the moving average of time-series at time “t”

$$\bar{X}_{t,M} = \bar{X}_{t-1,M} - \frac{X_{t-M}}{M} + \frac{X_t}{M}$$

- Where: M – window size of the moving average.
- Any any time “t” predict “t+1” by removing last “t-M” element, and adding “t” element.
- Requires last M elements to be stored in memory
- There are many more: mean, variance, regression, percentile

R – Open Source Statistics

- Open Source = low cost of adopting. Useful in prototyping.
- Large global community - more than 2.5 million users
- ~5,000 open source *free* packages
- Extensively used for modelling and visualisations
- “Microsoft Courts Data Scientists with Revolution Analytics Buy” (WSJ)



Source: Rexer Analytics

Use Case – Real-time Machine Learning

- Gaming ad-network
- 150m+ ad impressions per day
- 10 servers Storm cluster
- Lambda architecture (fast and batch layers): used in parallel to Hadoop, NoSQL
- Challenge
  - Make real-time decision on which ad to display
  - Use sophisticated statistical environment to A/B test ads
- Solution
  - Use **Storm + R** to do real-time statistics
  - **Beta Distribution** to compare two ads

Apache Storm

- Real-time calculations – the Hadoop of real time
- Fault tolerance
- Easy to scale
- Easy to develop - has local and distributed mode
- Storm multi-lang can be used with any language, here with R

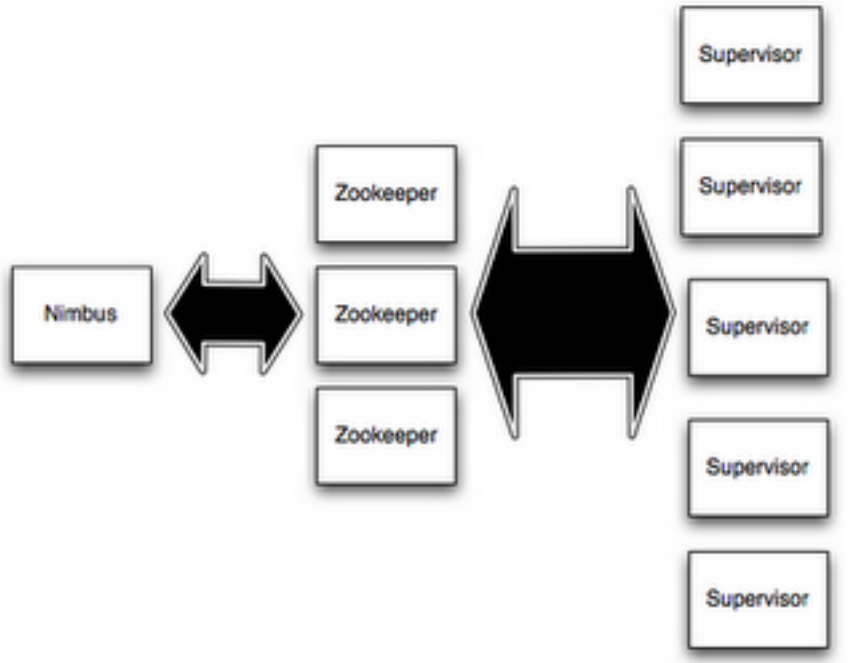
Getty Images

Apache Storm – Background

- Open sourced in September 2011
- Now part of Apache Storm
- Implementation ~15k LOC
- Written in Clojure / Java
- Stream processing – processes messages and updates database
- Continuous computation – query source, sends results to clients
- DRPC – distributed RPC
- Guaranteed message processing – no data loss
- Transactional topologies
- Storm Trident – easy to develop using abstract API

Storm Architecture

- Nimbus
  - Master - equivalent of Hadoop JobTracker
  - Distributes workload across cluster
  - Heartbeat, reallocation of workers when needed
- Supervisor
  - Runs the workers
  - Communicates with Nimbus using ZK
- Zookeeper
  - coordination, nodes discovery



Source: Apache Storm

Storm Topology

- Graph of stream computations
- Basic primitives nodes
  - Spout – source of streams (Twitter API, queue, logs)
  - Bolt – consumes streams, does the work, produces streams

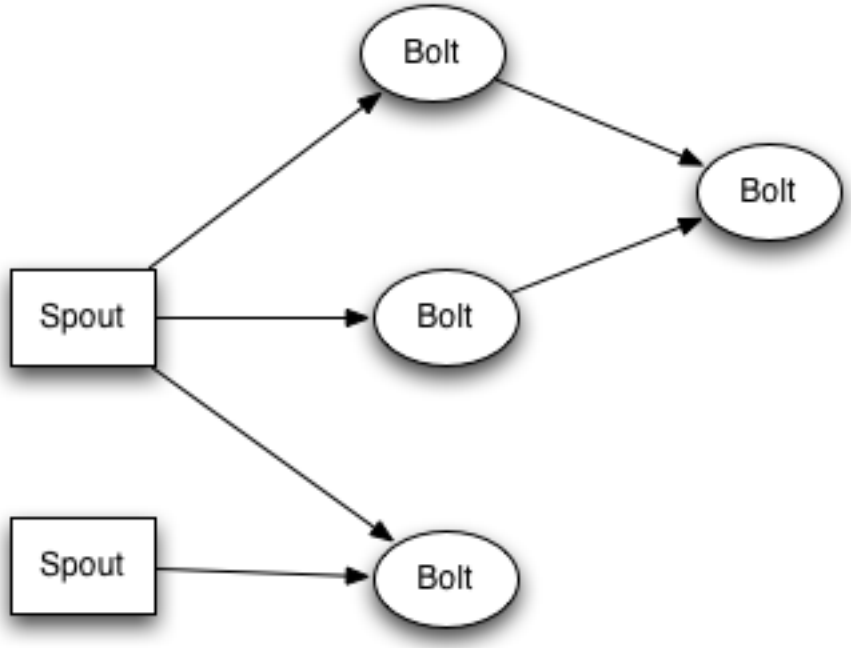


Image source: Storm github wiki

Can integrate with third party languages and databases:

- Java
- Python
- Ruby

- Redis
- Hbase
- Cassandra

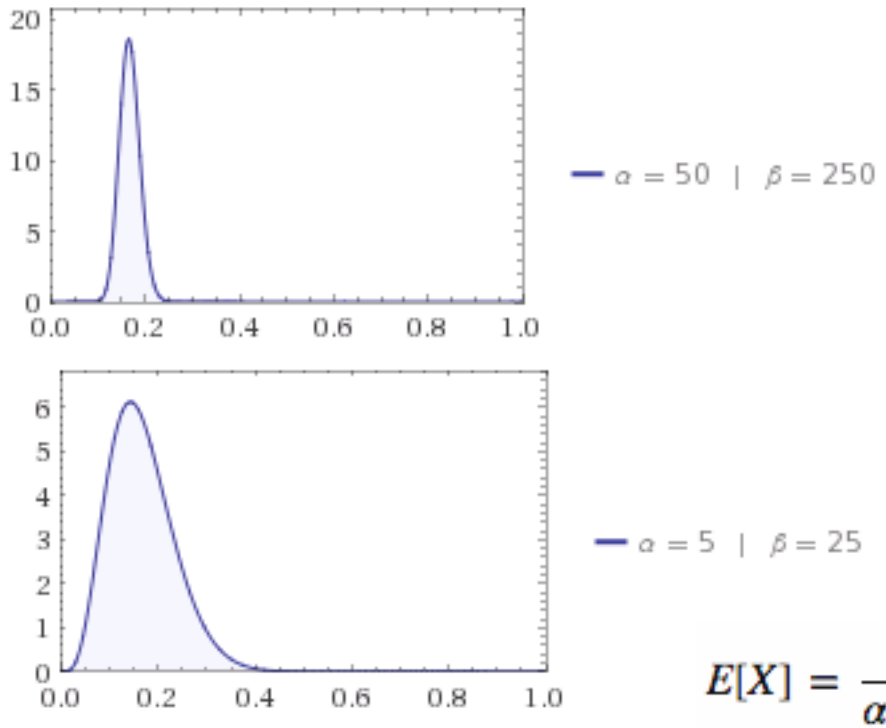
Storm + R

- Storm Multi-Language protocol
- Multiple Storm-R multi-language packages provide Storm/R plumbing
- Recommended package: http://cran.r-project.org/web/packages/Storm
- Example R code

Use Case – Beta Distributions

- Simple approach: $CTR = \frac{5}{30} = \frac{50}{300} = 0.1666667$
- Wolphram Alpha: beta distribution (5, (30-5))

Plot of PDF:



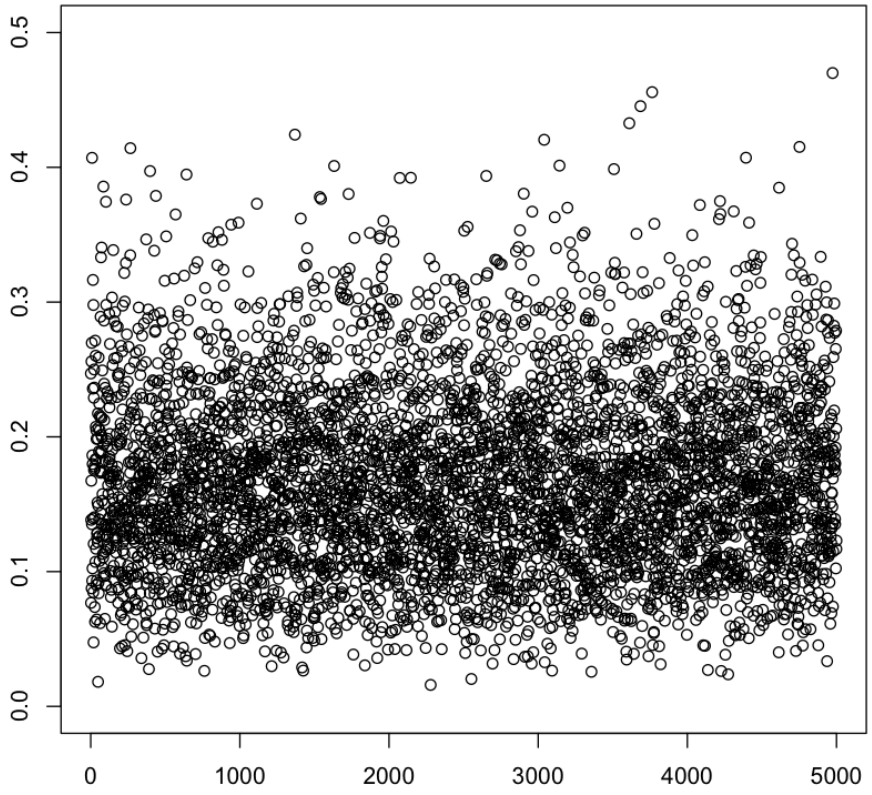
$$E[X] = \frac{\alpha}{\alpha + \beta} = \frac{5}{5 + 25} = \frac{5}{30}$$

Source: Wolphram Alpha

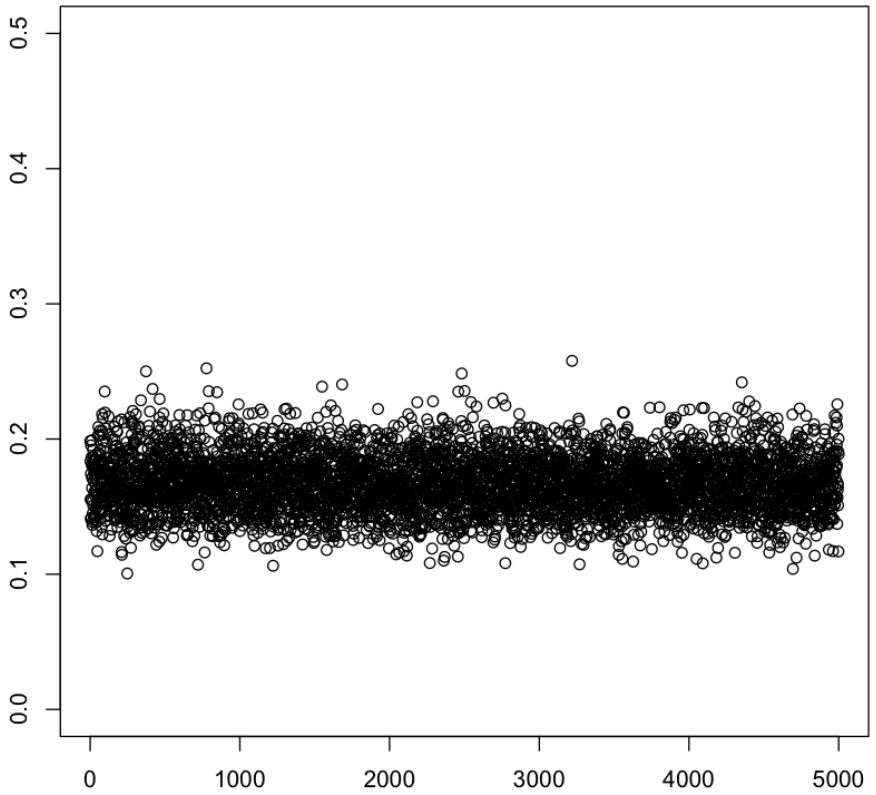
Beta distributions prototyping – the R code

- Bootstrapping in R

```
> rboot <- function(l) {rbeta(1, 5, 30-5)}
> plot(sapply(1:5000, rboot))
```




```
> rboot <- function(l) {rbeta(1, 50, 300-50)}
> plot(sapply(1:5000, rboot))
```



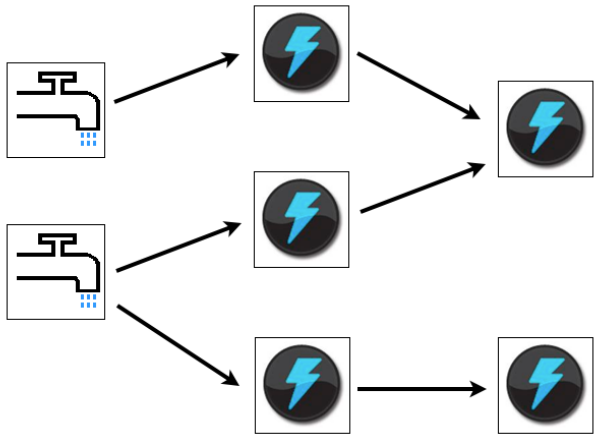
Storm and R

```
storm = Storm$new();
storm$lambda = function(s) {
  t = s$tuple;
  t$output =
vector(mode="character",length=1);
  clicks = as.numeric(t$input[1]);
  views = as.numeric(t$input[2]);
  t$output[1] = rbeta(1, clicks, views -
clicks);
  s$emit(t);
  #alternative: mark the tuple as failed.
  s$fail(t);
}
storm$run();
```

Storm and Java integration

- Define Spout/Bolt in any programming language
- Executed as subprocess – JSON over stdin/stdout

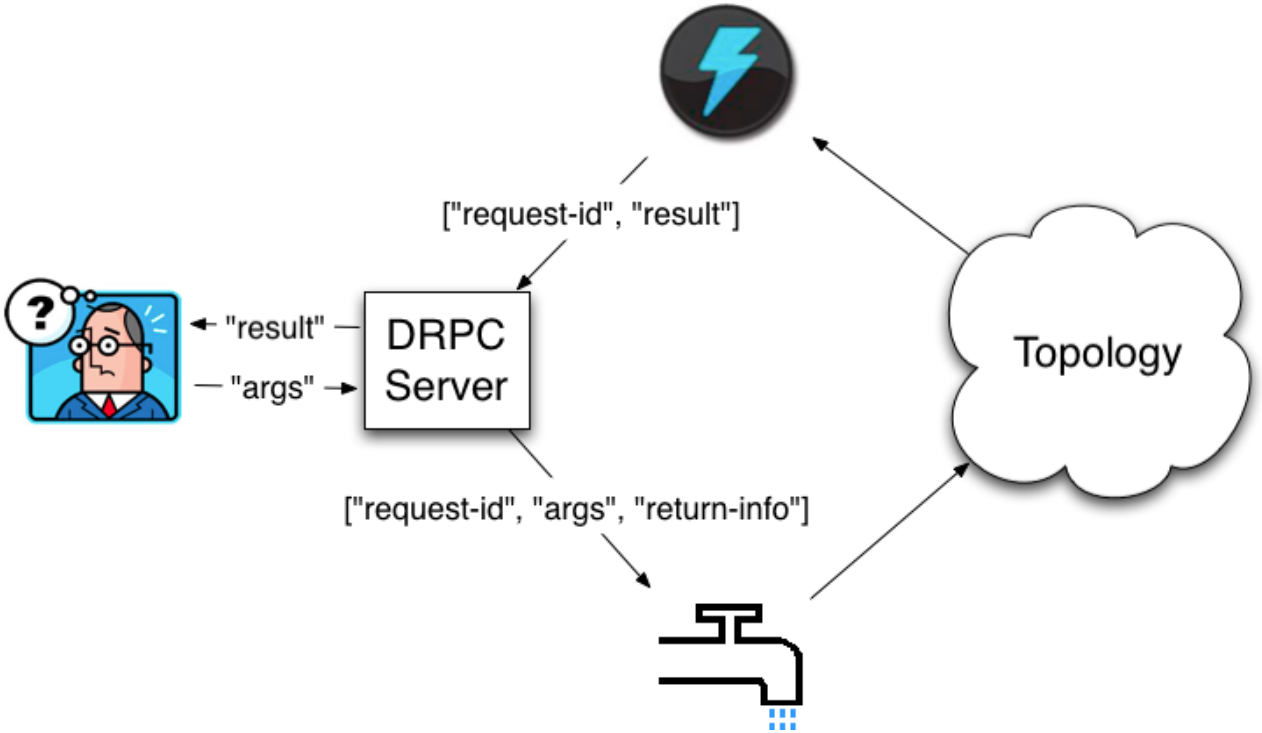
```
public static class RBolt extends ShellBolt
implements IRichBolt {
  public RBolt() {
    super("Rscript", "script.R");
  }
}
```



Source: Apache Storm

Storm + R = flexibility

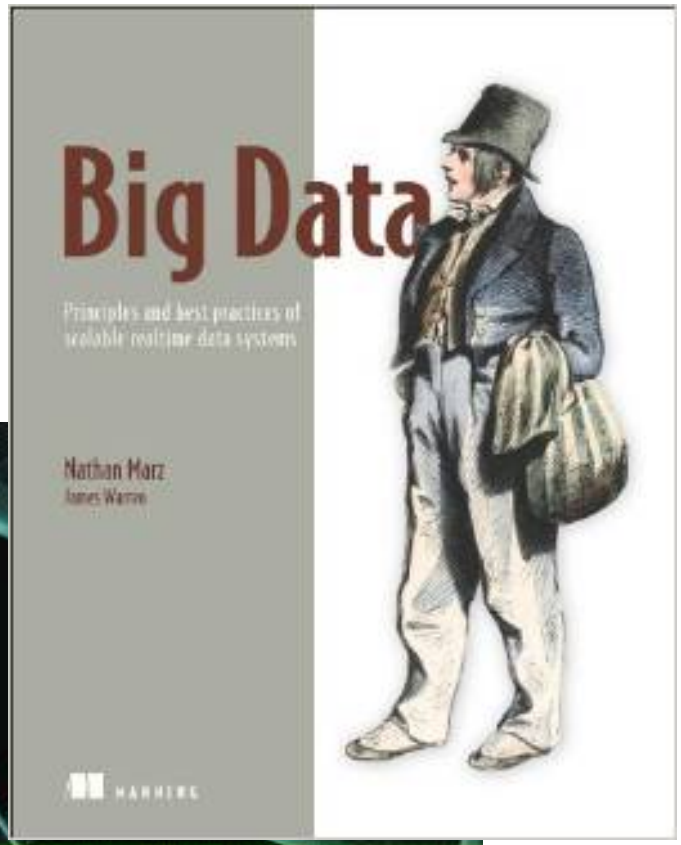
- Integration with existing Storm ecosystem – NoSQL, Kafka
- SOA framework - DRPC
- Scaling up your existing R processes
- Trident



Source: Apache Storm

Storm References

- https://storm.apache.org



Thank you

- Data Stream will save you when:
  - Big Data problems
  - Do not want to flip the coins for the next 10 years
  - Make decisions in real-time

- Questions and discussion

- https://uk.linkedin.com/in/radekmaciaszek
- http://www.dataminelab.com