End-to-end Hadoop for search suggestions

Guest lecture at RuG

WEB AND CLOUD COMPUTING

Jaargang 2 No. 9 | 25 september 2012 | VAKBLAD VOOR MANAGERS EN ICT PROFESSIONALS

Public cloud, private cloud, legacy
Een uitdagende puzzel

Voorstanders van cloud computing beloven zo ongeveer de hemel. Maar in de praktijk blijkt de cloud toch iets weerbarstiger. Om van droom naar daad te komen, moeten er nogal wat hobbels worden genomen. Een van de obstakels wordt zelden genoemd: legacy. Hoe verbind je het logge verleden met de luchtige

De innovatie-fabriek

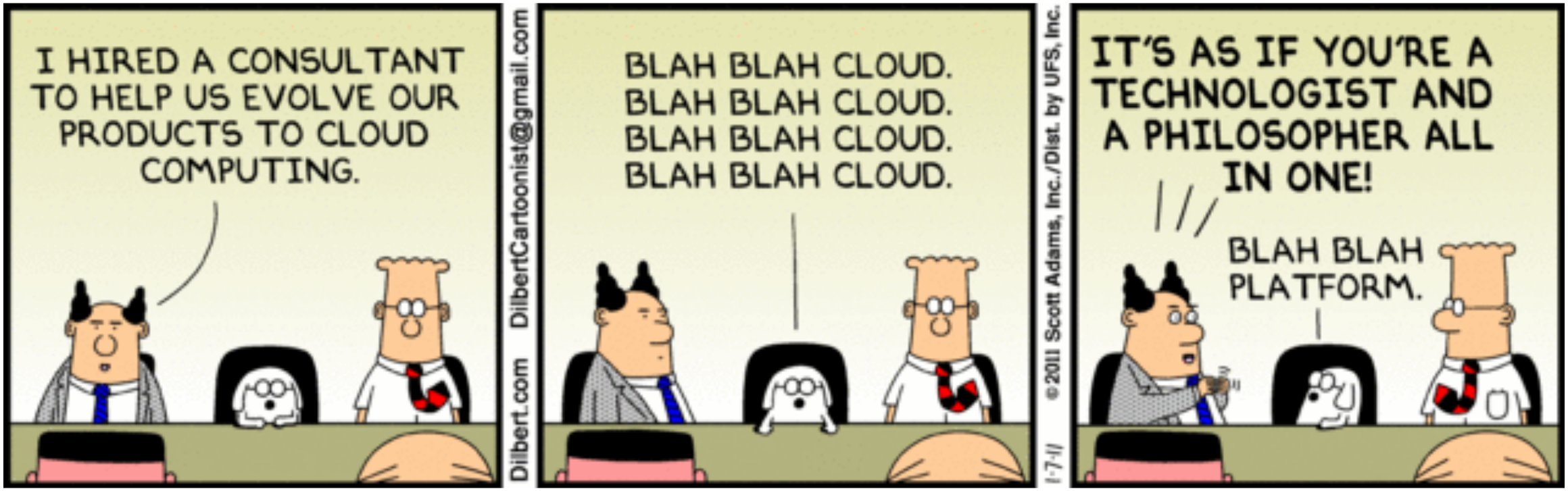
Wat is er toch aan de hand met innovatietrajecten? Enerzijds klagen we steen en been over het gebrek aan werkelijke innovaties en willen we niets liever dan innovatieve medewerkers, producten en processen. Maar aan de andere kant stuiten echt innovatieve medewerkers op weerstand en moeilijkheden en passen hun werkprocessen zelden in de bestaande structuren. Hoe is deze tegenstrijdigheid te verklaren?

☞ pagina 10

De controller a

De klassieke controller is een uitster
leen kan hij zijn taak niet meer ver
welke ontwikkelingen van invloed
moet hij weten of die software up

regeling illustreert Gerard Bottem
controller anno 2012.



http://dilbert.com/strips/comic/2011-01-07/

CLOUD COMPUTING

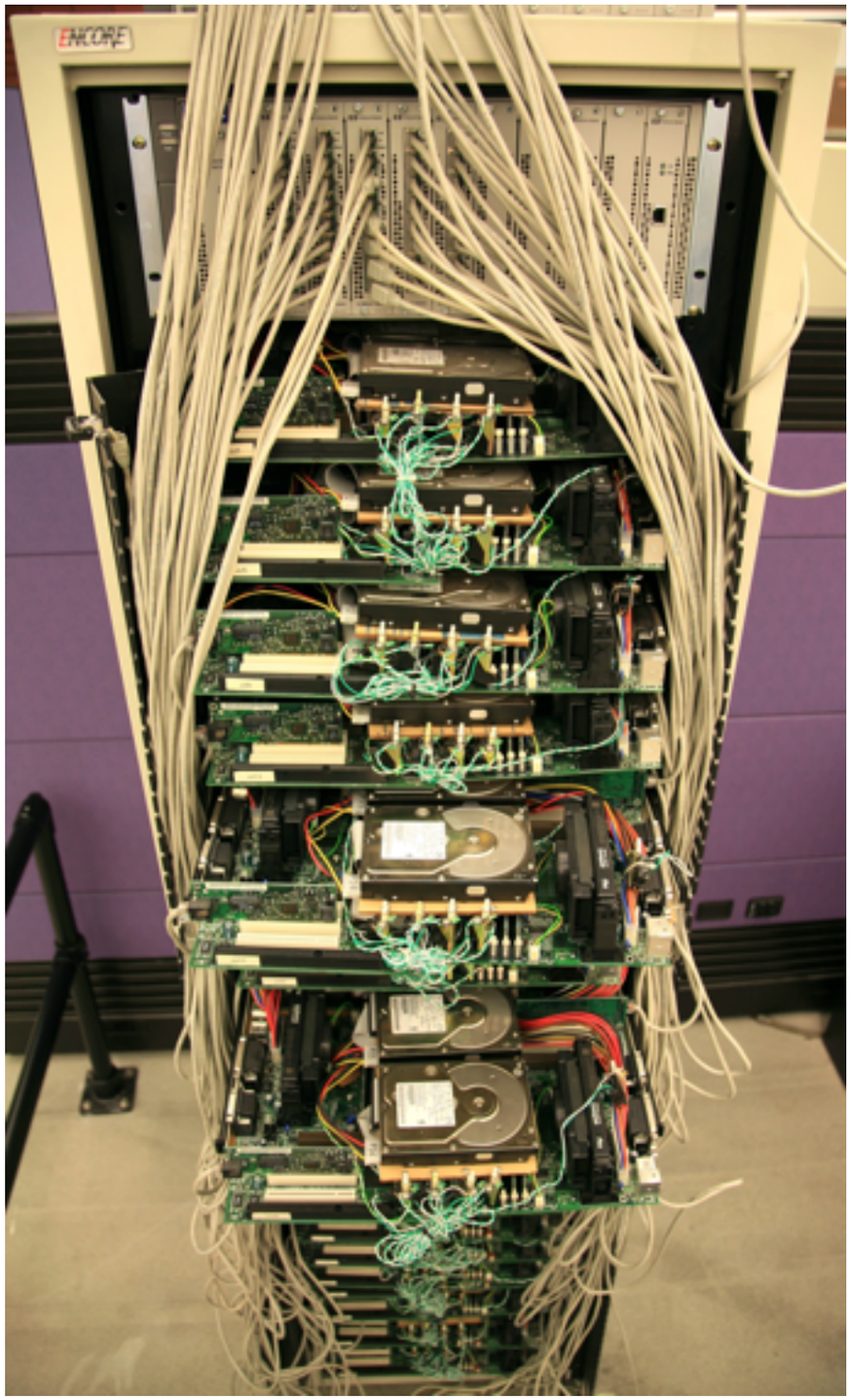


image: flickr user jurvetson

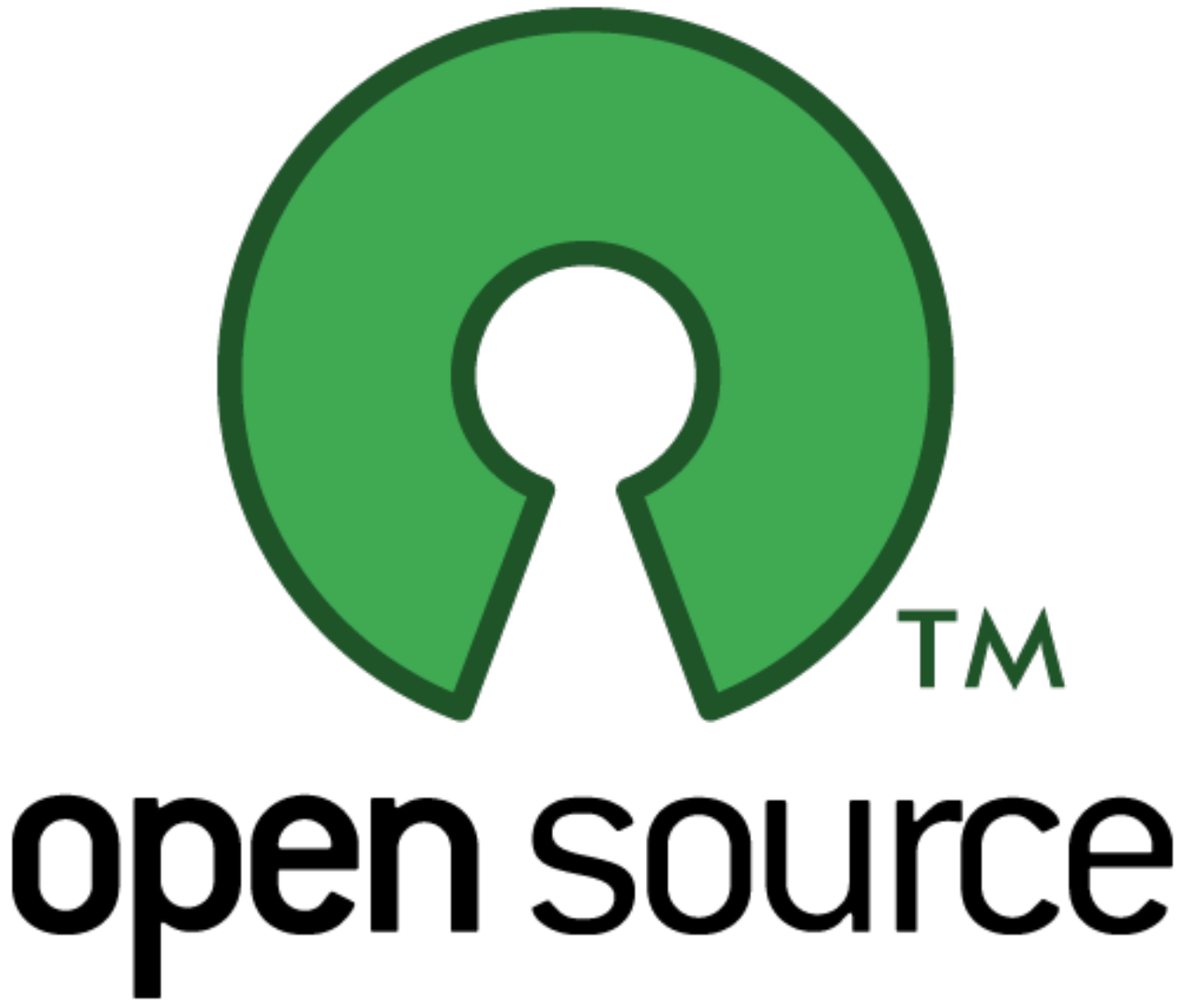
FOR RENT

Instagram

30+ million users in
two years

With 3 engineers!

Also,



A quick survey...

This talk:

Build working search suggestions (like on bol.com)

That update automatically based on user behavior

Using a lot of open source (elasticsearch, Hadoop, Redis, Bootstrap, Flume, node.js, jquery)

With a code example, not just slides

And a live demo of the working thing in action

Introducing Bootstrap.

Need reasons to love Bootstrap? Look no further.

By nerds, for nerds.

Made for everyone.

Packed with features.

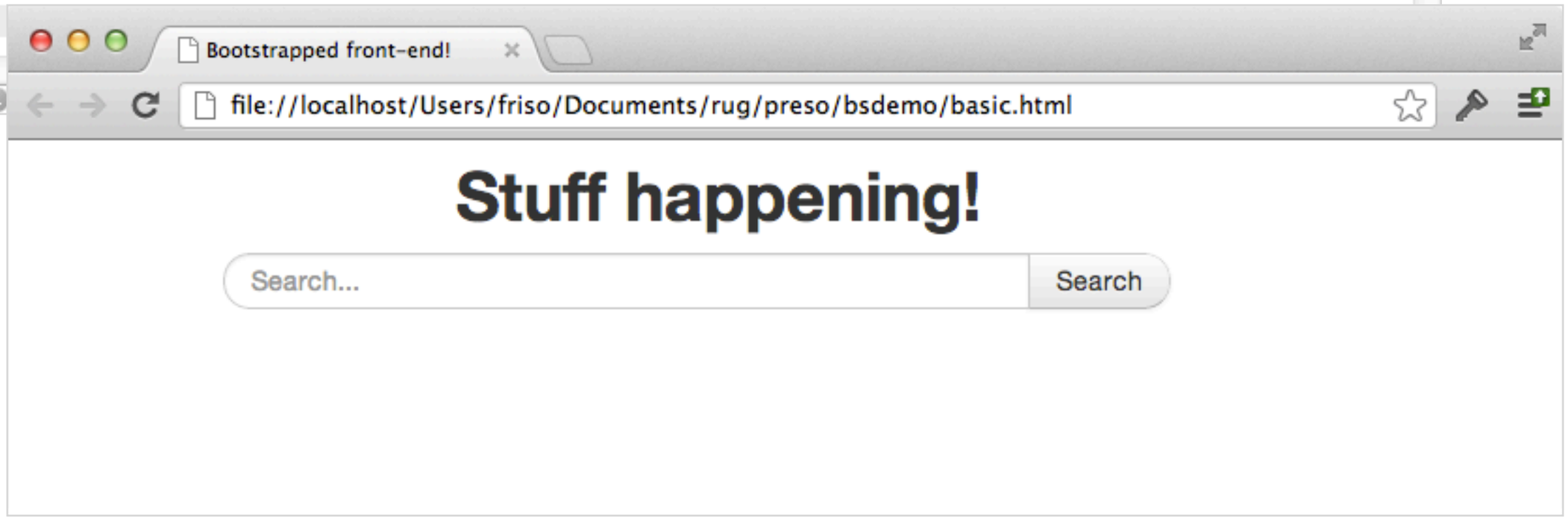
http://twitter.github.com/bootstrap/

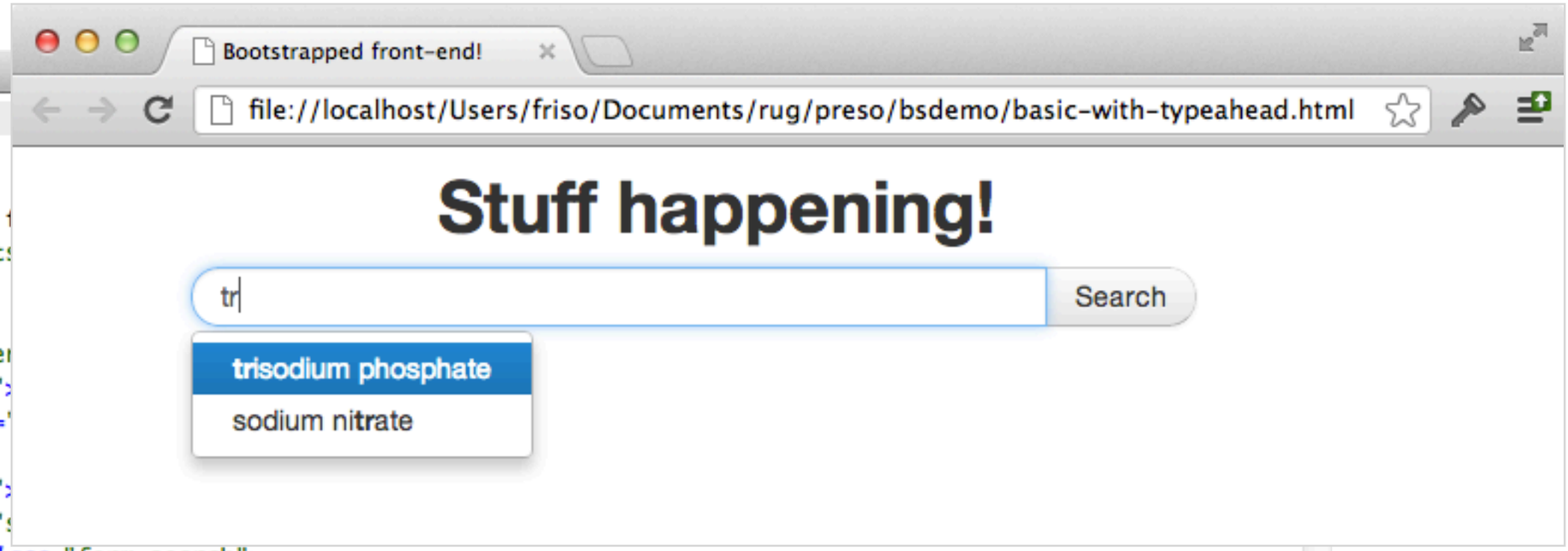
basic.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Bootstrapped front-end!</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container">
        <div class="row">
            <div class ="span6 offset3"><h1>Stuff happening!</h1></div>
        </div>
        <div class="row">
            <div class="span10 offset1">
                <form class="form-search">
                    <div class="input-append">
                        <input type="text" class="span7 search-query" placeholder="Search..." id="searchInput">
                        <button class="btn" data-loading-text="Searching..." id="searchButton">Search</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
    <script src="js/jquery-1.8.2.min.js"></script>
    <script src="js/bootstrap.min.js"></script>
</html>

```

Line: 26 Column: 1






```
<!DOCTYPE html>
<html>
<head>
    <title>Bootstrapped f
    <link href="css/boots
</head>
<body>
    <div class="containe
        <div class="row">
            <div class ="
        </div>
        <div class="row">
            <div class="s
                <form class="form-search">
                    <div class="input-append">
                        <input type="text" class="span7 search-query" placeholder="Search..." id="searchInput">
                        <button class="btn" data-loading-text="Searching..." id="searchButton">Search</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
    <script src="js/jquery-1.8.2.min.js"></script>
    <script src="js/bootstrap.min.js"></script>

    <script>
        $('#searchInput').typeahead({
            source: function(query, callback) {
                return ['ham', 'pork', 'sugar', 'salt', 'water', 'potato starch',
                'sodium nitrate', 'trisodium phosphate', 'sodium ascorbate']
            }
        })
    </script>
</html>
```

Line: 1 Column: 1 HTML Tab Size: 4

How do we populate this array?

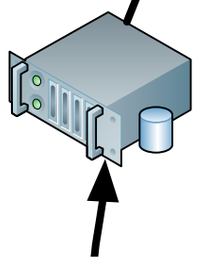
possibly many of these

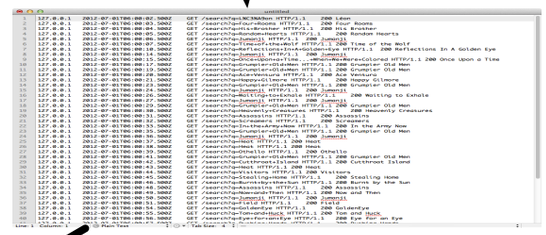
HTTP request:
/search?q=trisodium+phosphate

write to log file:
2012-07-01T06:00:02.500Z /search?q= trisodium%20phosphate

USER

serve suggestions for web requests

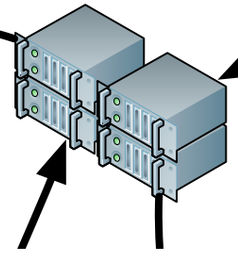




transport logs to compute cluster

push suggestions to DB server

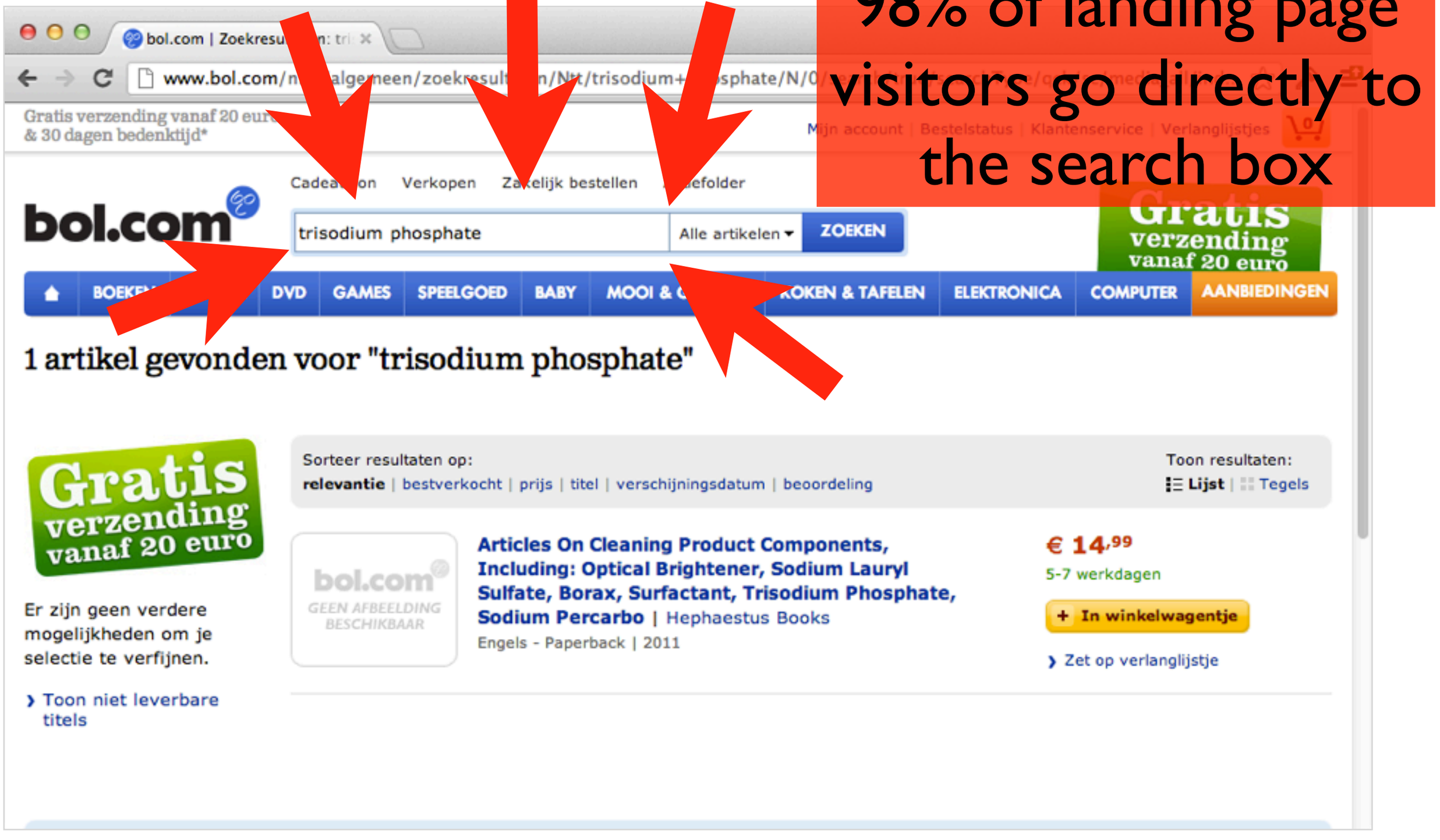
this thing must be really, really fast



transform logs to search suggestions

possibly terabytes of these

Search is important!
98% of landing page visitors go directly to the search box



MoSe!

localhost:3000/search.html

Movie Search

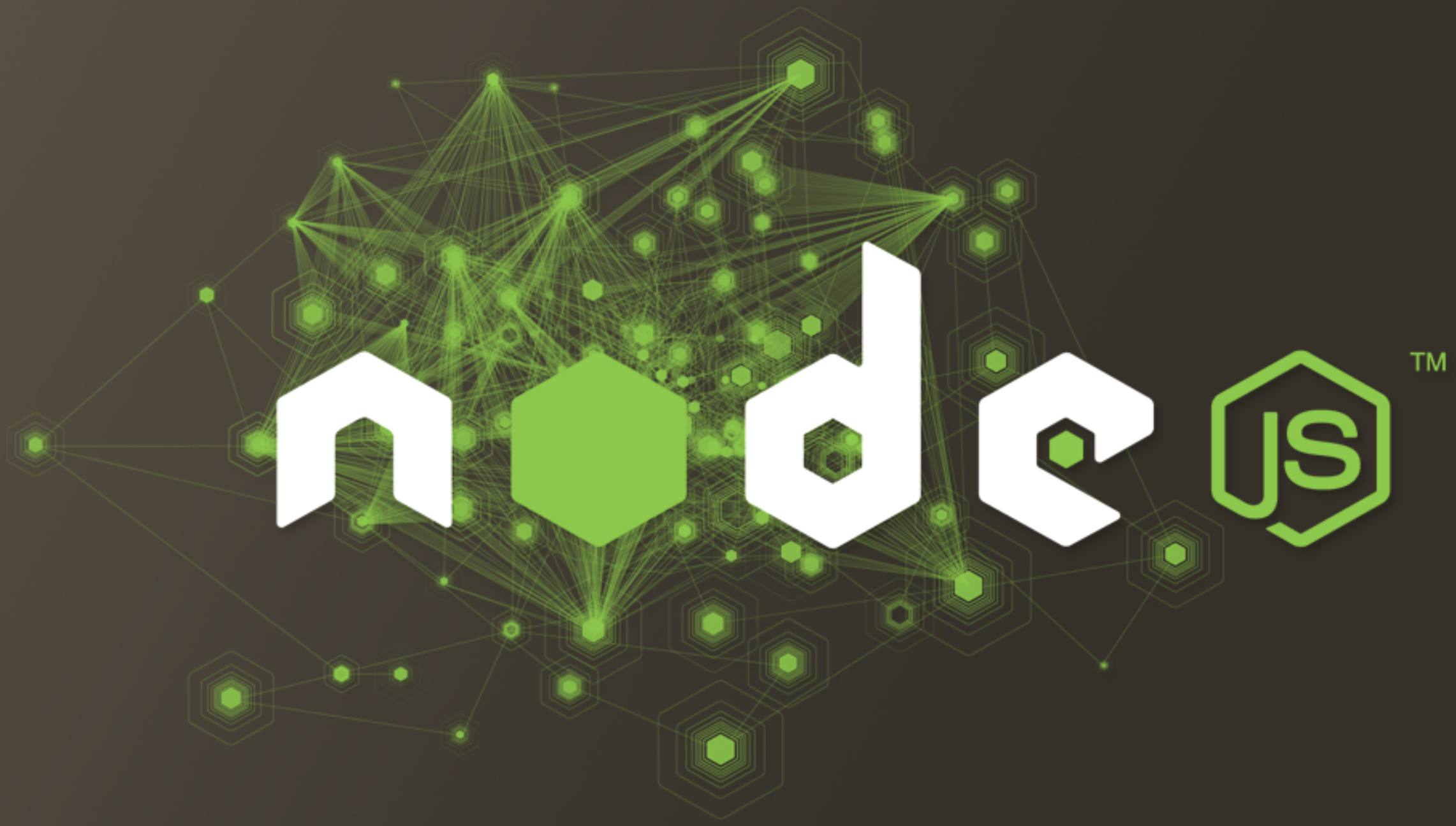
stuff Search

Title	
Stuff, The (1985)	details
Right Stuff, The (1983)	details
How to Stuff a Wild Bikini (1965)	details

http://www.elasticsearch.org/

friso — bash — 86×39

```
friso@fvv:~$ curl -XPOST http://localhost:9200/test/movie/_search?pretty=true -d \
> '
> {
>   "query": {
>     "query_string": {
>       "query":"stuff"
>     }
>   }
> }
> '
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 3,
    "max_score" : 4.995634,
    "hits" : [ {
      "_index" : "test",
      "_type" : "movie",
      "_id" : "59ddqE8ATfue2jnDq_UOEg",
      "_score" : 4.995634, "_source" : { "title":"Stuff, The (1985)" }
    }, {
      "_index" : "test",
      "_type" : "movie",
      "_id" : "Vlgg0AABT42UL_qdDTsvhQ",
      "_score" : 3.9739683, "_source" : { "title":"Right Stuff, The (1983)" }
    }, {
      "_index" : "test",
      "_type" : "movie",
      "_id" : "8dRSQiCMQ0-i7uzrQ5qFMw",
      "_score" : 3.498948, "_source" : { "title":"How to Stuff a Wild Bikini (1965)" }
    } ]
  }
friso@fvv:~$
```



http://nodejs.org/

Node.js

Server-side JavaScript

Built on top of the V8 JavaScript engine

Completely asynchronous / non-blocking / event-driven

Non-blocking means function calls never block

untitled 2

```
1  var fs = require('fs')
2
3  fs.readFile('/etc/passwd') // <== this returns immediately (it doesn't block)
4
5  // We have to pass a call back, to get the result
6  fs.readFile('/etc/passwd', function(err, data) {
7      if (err) {
8          console.error(err)
9      } else {
10         console.log(data)
11     }
12 })
13
14 // Let's create a HTTP server
15 var http = require('http');
16 http.createServer(function (req, res) {
17     res.writeHead(200, {'Content-Type': 'text/plain'});
18     res.end('SPAM!\n');
19 }).listen(3000, '127.0.0.1');
20
21 console.log('Server running at http://127.0.0.1:3000/');
22
```

Line: 22 Column: 1 | JavaScript | Tab Size: 4 | console


hadoop


hadoop
HDFS


hadoop
MapReduce


HIVE


APACHE
HBASE


mahout


AVRO
TM


ZOO


FLUME

Distributed filesystem

Consists of a single master node and multiple (many) data nodes

Files are split up blocks (64MB by default)

Blocks are spread across data nodes in the cluster

Each block is replicated multiple times to different data nodes in the cluster (typically 3 times)

Master node keeps track of which blocks belong to a file

Accessible through Java API

FUSE (filesystem in user space) driver available to mount as regular FS

C API available

Basic command line tools in Hadoop distribution

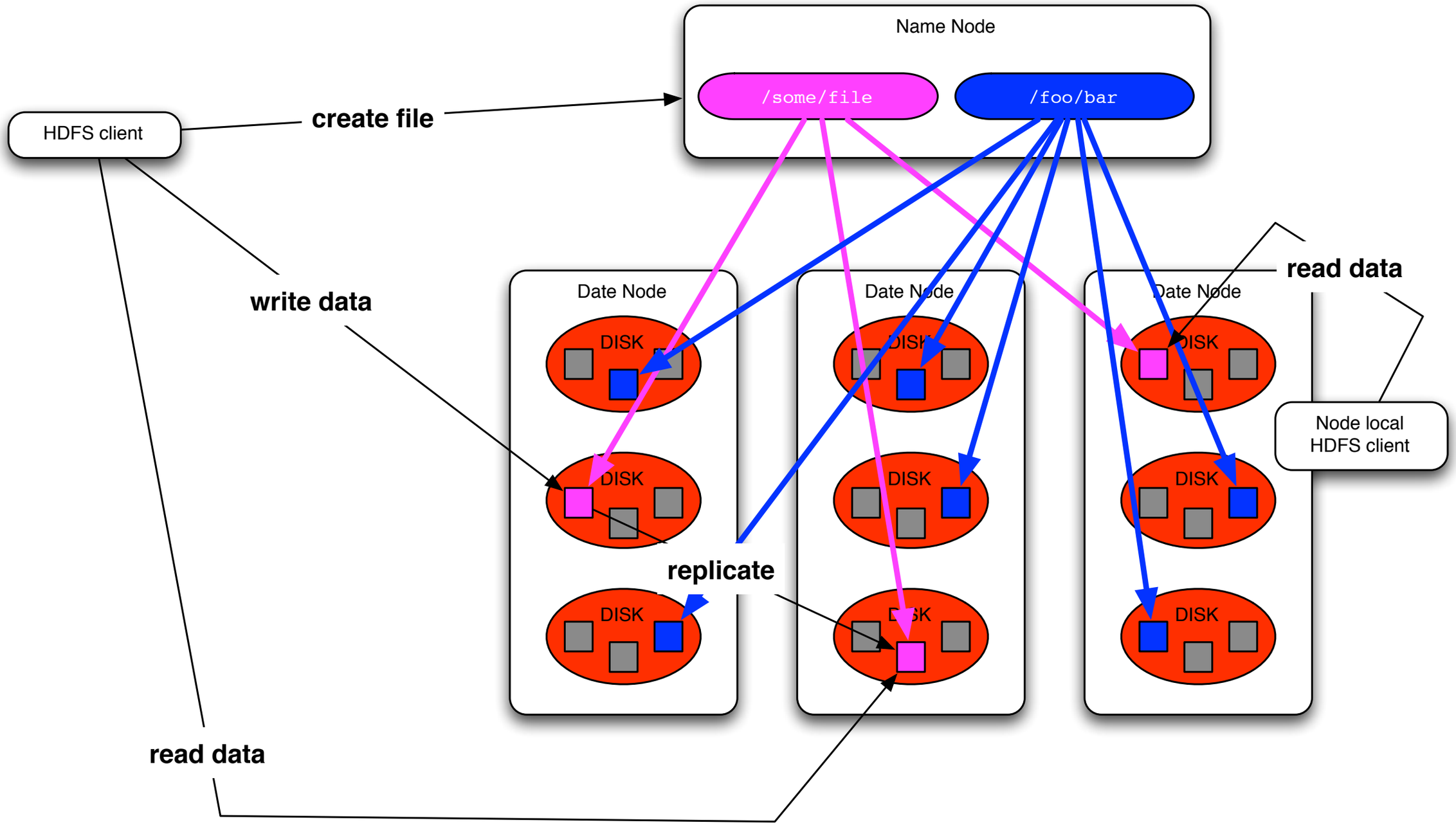
Web interface

File creation, directory listing and other meta data actions go through the master node (e.g. ls, du, fsck, create file)

Data goes directly to and from data nodes (read, write, append)

Local read path optimization: clients located on same machine as data node will always access local replica when possible

hadoop HDFS


Name Node
/some/file
/foo/bar
HDFS client
create file
write data
read data
Date Node
DISK
DISK
DISK
Date Node
DISK
DISK
DISK
Date Node
DISK
DISK
DISK
read data
Node local
HDFS client
replicate
read data

NameNode daemon

Filesystem master node

Keeps track of directories, files and block locations

Assigns blocks to data nodes

Keeps track of live nodes (through heartbeats)

Initiates re-replication in case of data node loss

Block meta data is held in memory

Will run out of memory when too many files exist

DataNode daemon

Filesystem worker node / “Block server”

Uses underlying regular FS for storage (e.g. ext4)

Takes care of distribution of blocks across disks

Don’t use RAID

More disks means more IO throughput

Sends heartbeats to NameNode

Reports blocks to NameNode (on startup)

Does not know about the rest of the cluster (shared nothing)

HDFS trivia

HDFS is write once, read many, but has append support in newer versions

Has built in compression at the block level

Does end-to-end checksumming on all data (on read and periodically)

HDFS is best used for large, unstructured volumes of raw data in BIG files used for batch operations

Optimized for sequential reads, not random access

Job input comes from files on HDFS

Other formats are possible; requires specialized InputFormat implementation

Built in support for text files (convenient for logs, csv, etc.)

Files must be splittable for parallelization to work

Not all compression formats have this property (e.g. gzip)

JobTracker daemon

MapReduce master node

Takes care of scheduling and job submission

Splits jobs into tasks (Mappers and Reducers)

Assigns tasks to worker nodes

Reassigns tasks in case of failure

Keeps track of job progress

Keeps track of worker nodes through heartbeats

TaskTracker daemon

MapReduce worker process

Starts Mappers en Reducers assigned by JobTracker

Sends heart beats to the JobTracker

Sends task progress to the JobTracker

Does not know about the rest of the cluster (shared nothing)

NameNode
JobTracker

DataNode
TaskTracker

DataNode
TaskTracker

DataNode
TaskTracker

DataNode
TaskTracker

"Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data."

http://flume.apache.org/

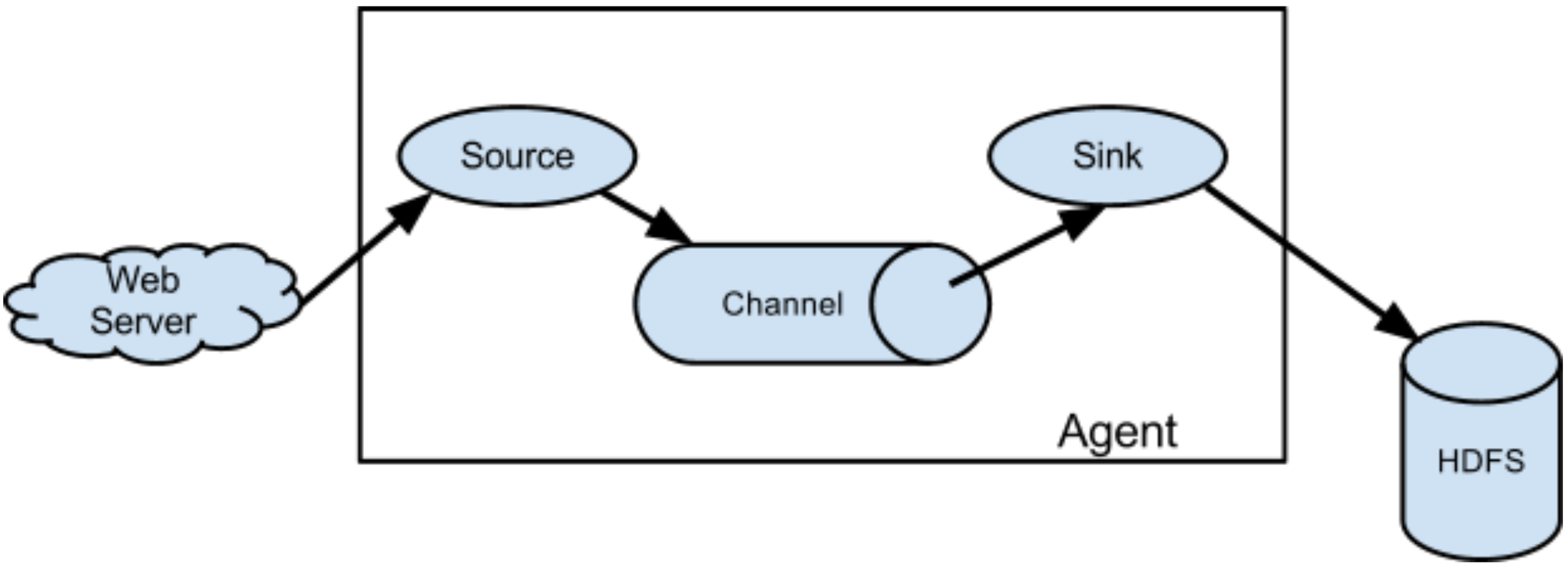


image from http://flume.apache.org

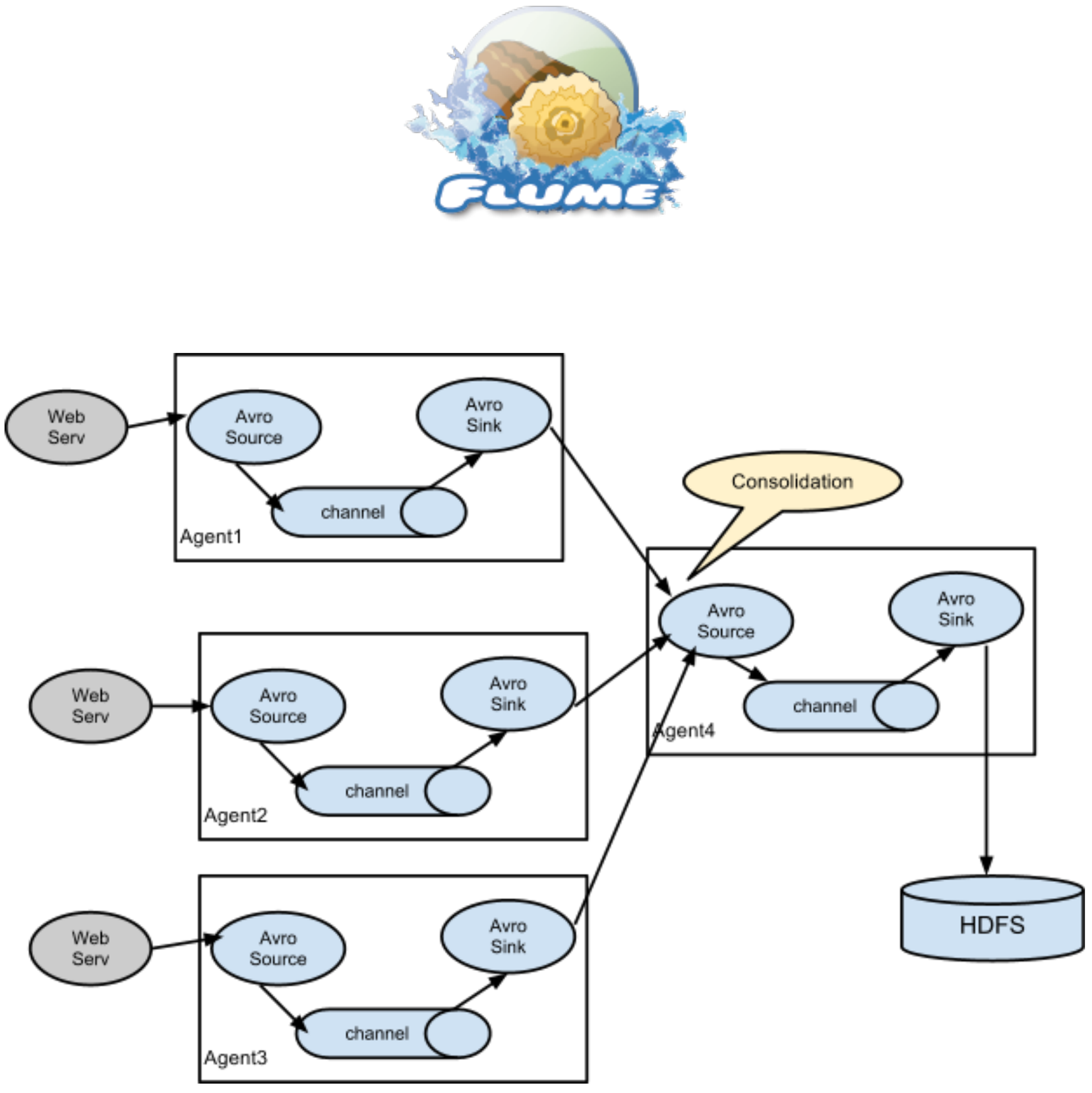


image from http://flume.apache.org

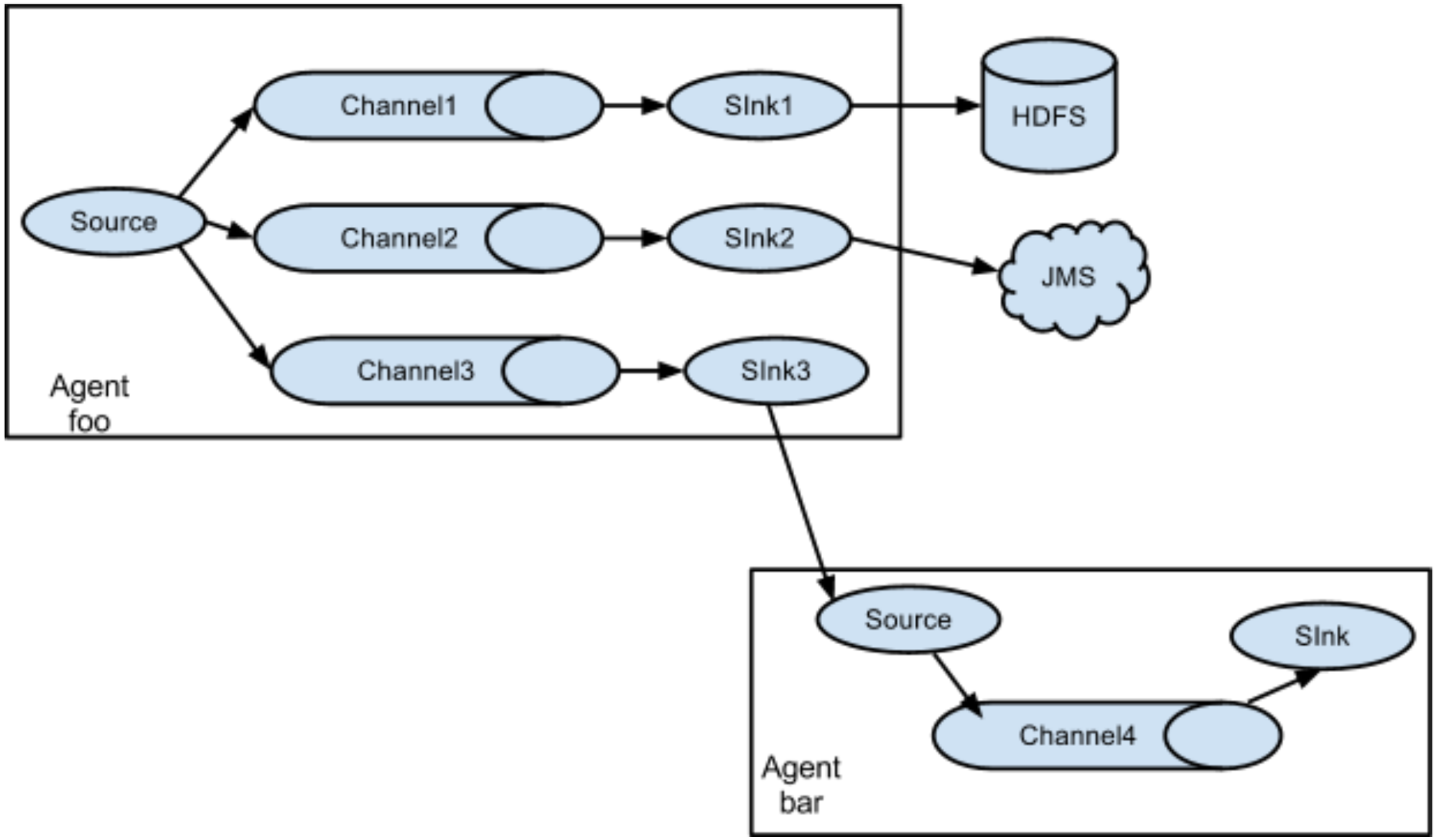


image from http://flume.apache.org

Processing the logs, a simple approach:

Divide time into 15 minute buckets

Count # times a search term appears in each bucket

Divide counts by (current time - time of start of bucket)

Rank terms according to sum of divided counts

MapReduce, Job 1:

```
FIFTEEN_MINUTES = 1000 * 60 * 15

map (key, value) {
  term = getTermFromLogLine(value)
  time = parseTimestampFromLogLine(value)
  bucket = time - (time % FIFTEEN_MINUTES)

  outputKey = (bucket, term)
  outputValue = 1

  emit(outputKey, outputValue)
}

reduce(key, values) {
  outputKey = getTermFromKey(key)
  outputValue = (getBucketFromKey(key), sum(values))

  emit(outputKey, outputValue)
}
```

MapReduce, Job 2:

```
map (key, value) {
  for each prefix of key {
    outputKey = prefix
    count = getCountFromValue(value)
    score = count / (now - getBucketFromValue(value))
  }
}

reduce(key, values) {
  for each term in values {
    storeOrUpdateTermScore(value)
  }

  order terms by descending scores

  emit(key, top 10 terms)
}
```

Processing the logs, a simple solution:

Run Job 1 on new input (written by Flume)

When finished processing new logs, move files to another location

Run Job 2 after Job 1 on new and historic output of Job 1

Push output of Job 2 to a database

Cleanup historic outputs of Job 1 after N days

A database. Sort of...

redis

"Redis is an open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets."

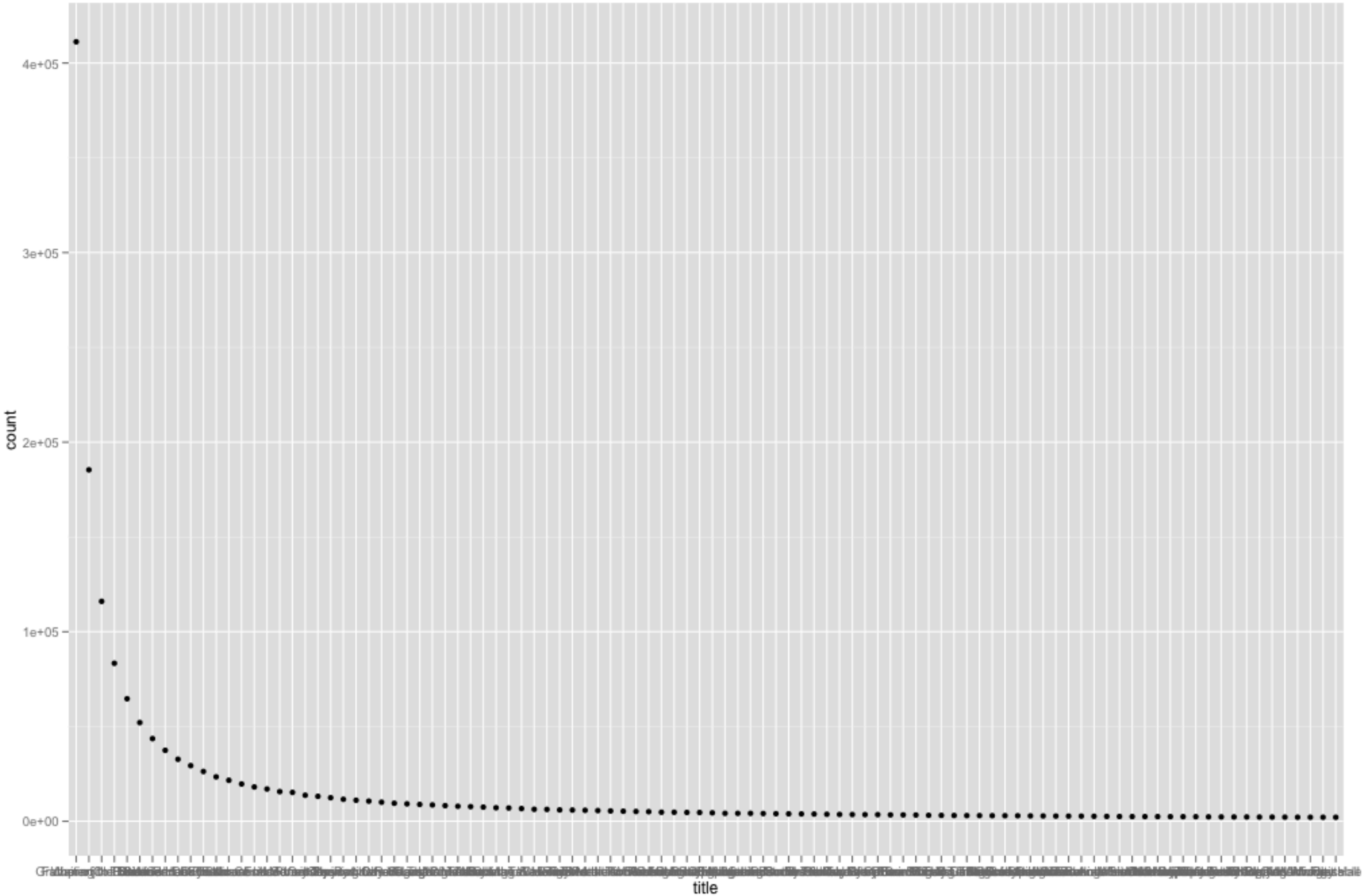
http://redis.io/

Insert into Redis:

Update all keys after each job

Set a TTL on all keys of one hour; obsolete entries go away automatically

How much history do we need?


4e+05
3e+05
2e+05
1e+05
0e+00
count
title

Improvements:

Use HTTP keep-alive to make sure the tcp stays open

Create a diff in Hadoop and only update relevant entries in Redis

Production hardening (error handling, packaging, deployment, etc.)

Options for better algorithms:

Use more signals than only search:

- Did a user type the text or click a suggestion?

Look at cause-effect relationships within sessions:

- Was it a successful search?
- Did the user buy the product?

Personalize:

- Filtering of irrelevant results
- Re-ranking based on personal behavior

Untitled

Line Numbers | Format Bar | Numbers | Reference | Insert Answer Token | Answer Palette

Line: 6 | Decimal | 10 dp | , | Notation Auto On Off | ° π

```
1  html = 71 lines                                  71
2  javascript = 54 + 68 lines                       122
3  flume_config = 32 lines                          32
4  java = 541 lines                                 541
5  html + javascript + flume_config + java          766
6
```

Total: 1.532

Q&A

code is at
https://github.com/friso/rug

fvanvollenhoven@xebia.com
@fzk