

Crawling the Web for **Common Crawl**



Sebastian Nagel

snagel@apache.org

sebastian@commoncrawl.org

Apache Big Data Europe 2016

About Me

computational linguist

software developer, search and data matching

since 2016 crawl engineer at Common Crawl

Apache Nutch user since 2008, committer and PMC since 2012

Common Crawl



We're a non-profit that makes web data accessible to anyone

- founded in 2007 by Gil Elbaz
- lower the barrier to “the web as a dataset”
- running your own large-scale crawl is expensive and challenging
- innovation occurs by using the data, rarely through new collection methods

Crawled Data

Data published on AWS S3 as Public Data Set

- released for free
“without additional intellectual property restrictions”
- 1.4 PiB of crawl archives back to 2008
- since 2014 monthly sample snapshots of the web,
every snapshot about 2 billion pages or 50 TiB of data
- HTML pages (small percentage of other document formats)
- for data mining, data and web science, natural language
processing, prototyping, testing, etc.
- URL index for all crawl archives since 2012:
API and index files

Crawled Data

Dedicated datasets (since August/September 2016)

- robots.txt files/responses
- “content-less” server responses: redirects, 404s, etc.
- articles and pages from news sites

News Crawl

- monthly crawl schedule not well-adapted to the news genre
- continuously release freshly crawled data
- incorporate new seeds quickly and efficiently
- based on **StormCrawler** running on **Apache Storm**
- test and evaluate a different crawler architecture with constant/ongoing use of hardware
- at present, 50,000 pages daily, but we want to get 100,000s

What makes us most excited? Experiments!

Use the data! Run your experiments!

- generate value from the data
- including scientific work
- analytics: servers, libraries, metadata, tracking, vulnerabilities, document formats
- web graph analysis
- extraction and aggregation: tables, RDFa, Microdata
- Natural Language Processing
- ... and share your experience! Write a blog post, a paper, ...

What makes us most excited? ... Derived Datasets!

Processing Terabytes of web archives is still not trivial, create derived datasets with single focus on ...

- **Web Data Commons**
 - hyperlink web graph
 - semantic web triples from RDFa, microformats, microdata
 - tables from web pages, relations from is-a-phrases
- **N-gram counts and language models** for statistical machine translation
- **GloVe: Global Vectors for Word Representation** (word2vec)
- **C4Corpus: Multilingual Web-Size Corpus with Free License**
 - detect pages with Creative Commons license marker
 - extracted text and remove boilerplate

Format of the Crawl Archives

WARC (Web ARChive)

- HTTP request and response headers
- payload (HTML or binary)
- metadata: date, IP address, etc.
- in short: HTTP traffic between crawler and web servers

WAT (1/3rd size of WARC)

- HTML meta data
- HTTP header fields
- extracted links

WET (1/10th size of WARC)

- extracted plain text

WARC in Detail

WARC is a sustainable format

- ISO standard since 2009
- **wrappers for many programming languages**
(Python, Java, Go, R, ...)
- if not, it's easy to read:
text header + payload, usually gzipped

WARC in Detail

Random access WARC archives

- gzip spec allows multiple deflate blocks
- each WARC record compressed in a single block (10% larger than full gzip)
- allows to pull out single documents by file offset
- index server / memento / wayback machine

Accessing WARC File by Index

Query the URL index for the ApacheCon EU web page in 2013:

```
% curl -XGET 'http://index.commoncrawl.org/CC-MAIN-2013-20-index?url=apachecon.eu&output=json'  
{ "urlkey": "eu,apachecon)/", "timestamp": "20130519081324", "status": "200",  
  "url": "http://www.apachecon.eu/",  
  "filename":  
    "crawl-data/CC-MAIN-2013-20/segments/1368696400149/warc/CC-MAIN-20130516092640-00065-ip-10-60-113-1",  
  "length": "6033", "mime": "text/html",  
  "offset": "318480812", "digest": "A3MAVLQNZD3NSLJSXD5TPEXSMQQNJL2T" }
```

Accessing WARC File by Index

...and get the WARC record by filename, offset and length:

```
% curl -s -r318480812-$(318480812+6033-1) \  
  "https://commoncrawl.s3.amazonaws.com/crawl-data/CC-MAIN-2013-20/segments/1368696400149/warc/\  
CC-MAIN-20130516092640-00065-ip-10-60-113-184.ec2.internal.warc.gz" | gzip -dc  
WARC/1.0  
WARC-Type: response  
WARC-Date: 2013-05-19T08:13:24Z  
Content-Length: 25891  
  
HTTP/1.0 200 OK  
Date: Sun, 19 May 2013 08:13:23 GMT  
Content-Type: text/html; charset=utf-8  
  
<!DOCTYPE html>  
<html lang="en-us">  
  <head>  
    <meta charset="utf-8" />  
    <title>ApacheCon Europe 2012 | Welcome</title>
```

Accessing WARC File by Index

...another option to get the desired content out of the archive:

<http://index.commoncrawl.org/CC-MAIN-2013-20/http://www.apachecon.eu/>

Challenge Yourself

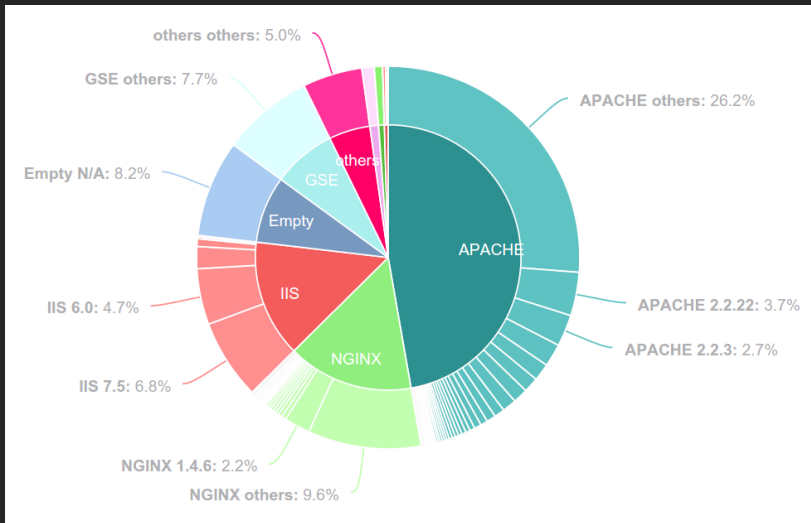
There's an amazing dataset at your fingertips, and getting started has never been simpler!

We provide Hadoop MapReduce examples and libraries to crunch the crawl archives for

- **Java**
- **Python** (based on mrjob)

... or use examples and code from our users

Challenge Yourself - Example



Analyzing Web Domain Vulnerability (Sietse T. Au and Wing Lung Ngai, 2014)

Challenge Yourself - Robots.txt Dataset

Web server name sent in HTTP response header

```
HTTP/1.1 200 OK
```

```
Date: Thu, 20 Oct 2016 19:32:52 GMT
```

```
Server: Apache/1.3.41 (Unix) mod_gzip/1.3.19.1a PHP/4.4.8
```

```
...
```

- could process WARC files
- or WAT (better because less data)
- or robots.txt dataset
 - robots.txt fetched once or few times per server
 - quite small
 - get an approximation of unique server counts

Challenge Yourself - Robots.txt Dataset - Simple Approach

```
aws s3 cp s3://commoncrawl/crawl-data/CC-MAIN-2016-44/robotstxt.paths.gz .
```

```
zcat robotstxt.paths.gz \  
  | while read path; do  
    aws s3 cp s3://commoncrawl/$path - \  
      | gunzip \  
      | perl -lne 'print $1 if m/^Server:\s*(.+?)\s*$/i'  
done \  
  | sort --compress-program=gzip \  
  | uniq -c \  
  | sort -k1,1nr \  
>server_counts.txt.gz
```

... will run about 12 hours on a small AWS EC2 instance

Challenge Yourself - Robots.txt Dataset - MapReduce on EMR

```
import re
from mrcc import CCJob

class ServerCount(CCJob):

    def process_record(self, record):
        if record['WARC-Type'] != 'response':
            return # we're only interested in the HTTP responses
        for line in record.payload:
            match = re.match('^server:\s*(.+)$', line, re.I)
            if match:
                yield match.group(1).strip(), 1
            return
        elif line.strip() == '':
            # empty line indicates end of HTTP response header
            yield '(no server in HTTP header)', 1
            return

if __name__ == '__main__':
    ServerCount.run()
```

Challenge Yourself - Robots.txt Dataset - MapReduce on EMR

Setup and configuration

```
git clone https://github.com/commoncrawl/cc-mrjob
cd cc-mrjob
```

```
pip install -r requirements.txt
```

```
# edit configuration in mrjob.conf
```

```
python server_count_warc.py -r emr --conf-path mrjob.conf robotstxt.paths.gz
```

Challenge Yourself - Server Counts

count	%	server name
53777944	25.8	Apache
27882022	13.4	nginx
16199473	7.8	GSE
9451948	4.5	(no server in HTTP header)
9439349	4.5	Microsoft-IIS/7.5
7077091	3.4	cloudflare-nginx
7001670	3.4	nginx/1.10.2
6131570	2.9	Microsoft-IIS/8.5
3363401	1.6	Apache/2
3214835	1.5	Microsoft-IIS/6.0
2917288	1.4	Apache/2.2.22 (Debian)
2282076	1.1	LiteSpeed
2183608	1.0	Apache/2.2.15 (CentOS)
2113086	1.0	Apache/2.4.7 (Ubuntu)
2019900	1.0	Microsoft-IIS/8.0
1856272	0.9	nginx/1.10.1
1847059	0.9	Apache/2.4.10 (Debian)
1729729	0.8	Apache/2.2.31 (Unix)
1665280	0.8	Microsoft-IIS/7.0
1240475	0.6	Apache/2.2.3 (CentOS)

Crawling


sample crawls,
not a comprehensive crawl

Historic Crawler

until 2012, [commoncrawl-crawler.git](https://github.com/commoncrawl-crawler), Ahad Rana

- Java, Hadoop, batch processing
- periodically
 - export crawl archives
 - update web graph and crawl database
 - generate fetch lists

Apache Nutch

since 2013 we use  *nutch*

blekko seed donations from end 2012 until early 2015

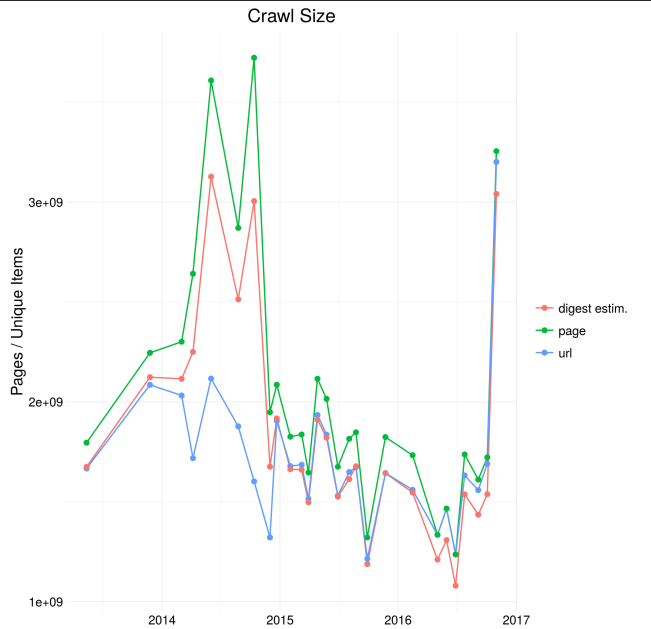
- what's nice: only care about how to crawl, not what to crawl
 - focus on crawler development and operations
 - Nutch used as a scalable distributed and polite fetcher
 - not bound to a specific hardware setup
 - runs on a cluster of AWS EC2 spot instances
- what's bad: without ongoing seed donations both freshness and coverage of the data drop over time

Crawl Archives

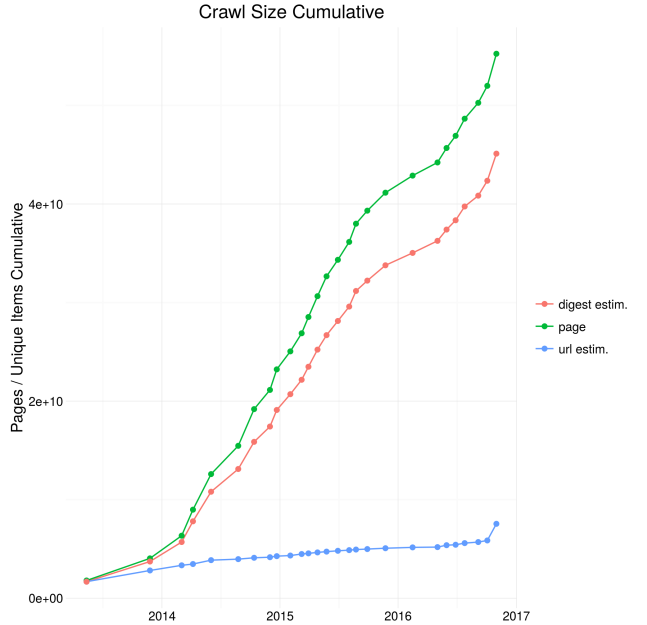
size of released crawl archives:

TiB	Year
8	2008
4	2009
25	2010
110	2012
85	2013
475	2014
400	2015
350	2016 (November)

Crawl Archives - Size since 2013



Crawl Archives - Cumulative Size



Why URL Frontier Discovery Matters

cumulative size of crawls from 2013 to 2016

- 55 billion fetched pages
- 7.5 billion unique URLs
- 45 billion unique content (by MD5 of “raw” binary content)

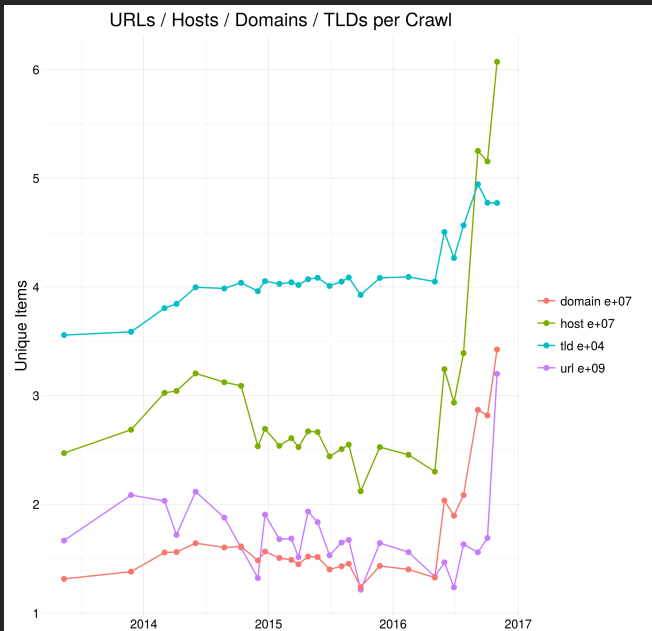
why it's difficult?

- there are many many outlinks
- a lot of duplicates and spam

at present, we rely on

- seed donations
- sitemaps

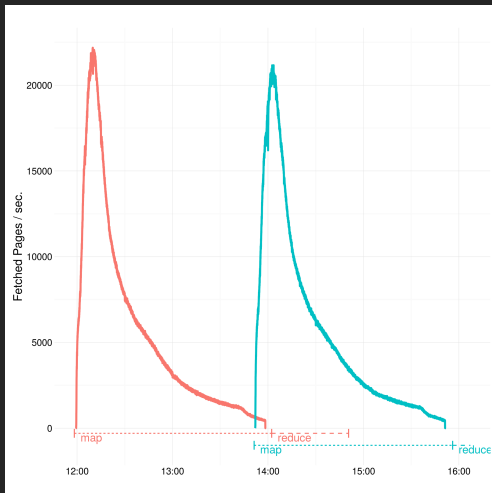
Crawl Archives - Host, Domain, TLD



Nutch Monthly Crawls - Cloud Setup

- CrawlDb (URLs with status information) persistent on S3:
1 TiB for 10 billion records
- Hadoop cluster (Cloudera CDH) on AWS EC2 spot instances
- small jobs (max. few hours): 100 batches or “segments”
- upload data to AWS S3 on completion to minimize data loss
in case of spot instance termination
- automated job launch and control by a shell script

Apache Nutch - Fetching



- avg. 1.5 TiB / hour or 7,000 pages / sec.
- 100 overlapping jobs
- 400 map tasks fetching
- limits on max. fetch time (2 hours) and min. throughput
- queuing per host for politeness
- dominated by long and short queues, too few mid-size queues

Politeness

be polite and

- respect robots.txt (exclusions and delays)
- limit load on crawled servers, guaranteed delay between fetches from the same host

implement host-level blocking on task-level (in JVM)

- all URLs must be in the same partition (fetch task)
- if locality is guaranteed, these are no bottlenecks:
 - DNS resolution
 - robots.txt fetching, parsing, caching
 - only 10^3 robots rules per fetch task: cache in memory

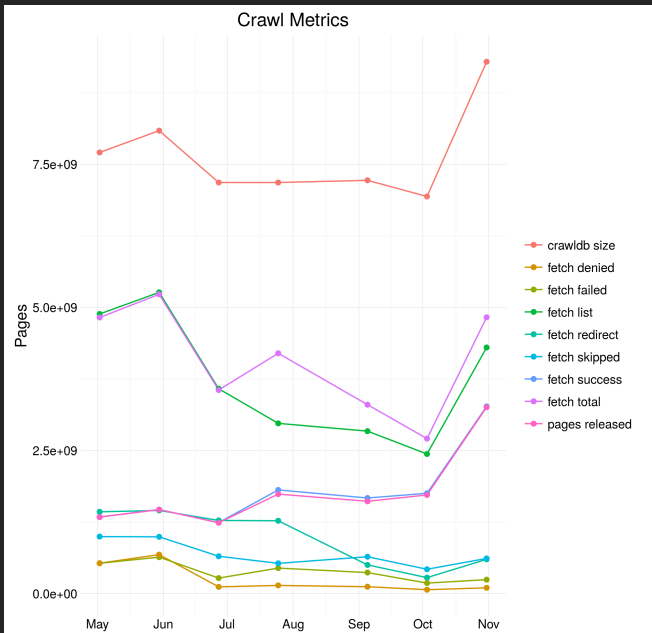
politeness and performance are not in contradiction!

Crawler vs. Fetcher

What's makes a crawler?

- following links
 - not ready yet: we need to take a sample of relevant pages with few duplicates and spam
- prioritization and scheduling
 - track and predict fetch status to get higher success rate
- manage CrawlDb
 - flag URLs which cause duplicates
- support web “standards”: sitemaps, canonical URLs, etc.

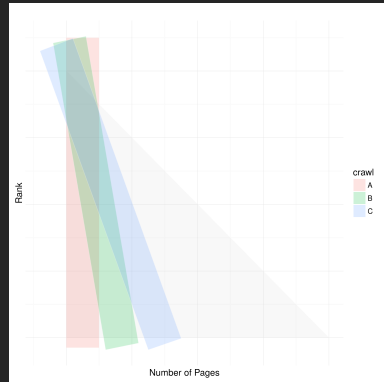
Predicting Fetch Status



Challenges and Plans

focus is on data quality

1. large and representative sample of the web
2. relevant content (few duplicates and spam)
3. language diversity (or countries resp. ccTLDs)
4. varying samples to cover different and more content over time
 - 30 billion pages fetched 2017
 - 15–20 billion different URLs
 - page-level ranks for sampling
5. find recently published content
6. improve user-friendly “secondary” formats (WAT, WET, ...)
7. support “end-users”: examples, tutorials, libraries, ...



Questions?

snagel@apache.org
sebastian@commoncrawl.org
<http://commoncrawl.org/>