Web Crawling with Apache Nutch

Sebastian Nagel
snagel@apache.org

exorbyte

ApacheCon EU 2014
2014-11-18
computational linguist
software developer at Exorbyte (Konstanz, Germany)
search and data matching
prepare data for indexing, cleansing noisy data,
web crawling
Nutch user since 2008
2012 Nutch committer and PMC
Nutch History

2002 started by Doug Cutting and Mike Caffarella
open source web-scale crawler and search engine
2004/05 MapReduce and distributed file system in Nutch
2005 Apache incubator, sub-project of Lucene
2006 Hadoop split from Nutch, Nutch based on Hadoop
2007 use Tika for MimeType detection, Tika Parser 2010
2008 start NutchBase, …, 2012 released 2.0 based on Gora 0.2
2009 use Solr for indexing
2010 Nutch top-level project
2012/13 ElasticSearch indexer (2.x), pluggable indexing back-ends (1.x)
What is Nutch now?

“an extensible and scalable web crawler based on Hadoop”

- runs on top of Hadoop
  - scalable: billions of pages possible
  - some overhead (if scale is not a requirement)
  - not ideal for low latency
- customizable / extensible plugin architecture
  - pluggable protocols (document access)
  - URL filters + normalizers
  - parsing: document formats + meta data extraction
  - indexing back-ends
- mostly used to feed a search index …
- …but also data mining
Nutch Community

- mature Apache project
- 6 active committers
  - maintain two branches (1.x and 2.x)
- “friends” — (Apache) projects Nutch delegates work to
  - Hadoop: scalability, job execution, data serialization (1.x)
  - Tika: detecting and parsing multiple document formats
  - Solr, ElasticSearch: make crawled content searchable
  - Gora (and HBase, Cassandra, ...): data storage (2.x)
  - crawler-commons: robots.txt parsing
- steady user base, majority of users
  - uses 1.x
  - runs small scale crawls (< 1M pages)
  - uses Solr for indexing and search
Crawler Workflow

0. initialize CrawlDb, **inject** seed URLs

repeat generate-fetch-update cycle \( n \) times:

1. **generate** fetch list: select URLs from CrawlDb for fetching
2. **fetch\** URLs from fetch list
3. **parse** documents: extract content, metadata and links
4. **update** CrawlDb status, score and signature, add new URLs

inlined or at the end of one crawler run (once for multiple cycles):

5. **invert links**: map anchor texts to documents the links point to
6. (calculate **link rank on web graph**, update CrawlDb scores)
7. **deduplicate** documents by signature
8. **index** document content, meta data, and anchor texts
Crawler Workflow

- seed URLs
- inject
- dedup
- CrawlDb
- fetch list
- generate
- content
- parse
- fetched data
- update
- index
- invert links
- LinkDb
Workflow Execution

- every step is implemented as one (or more) MapReduce job
- shell script to run the workflow (**bin/crawl**)
- **bin/nutch** to run individual tools
  - inject, generate, fetch, parse, updatedb, invertlinks, index, ...
  - many analysis and debugging tools
- local mode
  - works out-of-the-box (bin package)
  - useful for testing and debugging
- (pseudo-)distributed mode
  - parallelization, monitor crawls with MapReduce web UI
  - recompile and deploy job file with configuration changes
Features of selected Tools

Fetcher
- multi-threaded, high throughput
- but always be polite
- respect robots rules (robots.txt and robots directives)
- limit load on crawled servers, guaranteed delays between access to same host, IP, or domain

WebGraph
- iterative link analysis
- on the level of documents, sites, or domains

... most other tools
- are either quite trivial
- and/or rely heavily on plugins
Extensible via Plugins

Plugins basics
▶ each Plugin implements one (or more) extension points
▶ plugins are activated on demand (property plugin.includes)
▶ multiple active plugins per ext. point
  ▶ “chained”: sequentially applied (filters and normalizers)
  ▶ automatically selected (protocols and parser)

Extension points and available plugins
▶ URL filter
  ▶ include/exclude URLs from crawling
  ▶ plugins: regex, prefix, suffix, domain
▶ URL normalizer
  ▶ canonicalize URL
  ▶ regex, domain, querystring
Plugins (continued)

▶ protocols
  ▶ http, https, ftp, file
  ▶ protocol plugins take care for robots.txt rules

▶ parser
  ▶ need to parse various document types
  ▶ now mostly delegated to Tika (plugin parse-tika)
  ▶ legacy: parse-html, feed, zip

▶ parse filter
  ▶ extract additional meta data
  ▶ change extracted plain text
  ▶ plugins: headings, metatags

▶ indexing filter
  ▶ add/fill indexed fields
  ▶ url, title, content, anchor, boost, digest, type, host, ...
  ▶ often in combination with parse filter to extract field content
  ▶ plugins: basic, more, anchor, metadata, ...
Plugins (continued)

- index writer
  - connect to indexing back-ends
  - send/write additions / updates / deletions
  - plugins: Solr, ElasticSearch

- scoring filter
  - pass score to outlinks (in 1.x: quite a few hops)
  - change scores in CrawlDb via inlinks
  - can do magic: limit crawl by depth, focused crawling, ...
  - OPIC (On-line Page Importance Computation, [1])
    - online: good strategy to crawl relevant content first
    - scores also ok for ranking
    - but seeds (and docs “close” to them) are favored
    - scores get out of control for long-running continuous crawls

- LinkRank
  - link rank calculation done separately on WebGraph
  - scores are fed back to CrawlDb

Complete list of plugins:
http://nutch.apache.org/apidocs/apidocs-1.9/
Pluggable classes

Extensible interfaces (pluggable class implementations)

- fetch schedule
  - decide whether to add URL to fetch list
  - set next fetch time and re-fetch intervals
  - default implementation: fixed re-fetch interval
  - available: interval adaptive to document change frequency

- signature calculation
  - used for deduplication and not-modified detection
  - MD5 sum of binary document or plain text
  - TextProfile based on filtered word frequency list
Data Structures and Storage in Nutch 1.x

All data is stored as map \langle url, someobject \rangle in Hadoop map or sequence files.

Data structures (directories) and stored values:

- **CrawlDb**: all information of URL/document to run and schedule crawling
  - current status (injected, linked, fetched, gone, ...)
  - score, next fetch time, last modified time
  - signature (for deduplication)
  - meta data (container for arbitrary data)

- **LinkDb**: incoming links (URLs and anchor texts)

- **WebGraph**: map files to hold outlinks, inlinks, node scores
Data Structures (1.x): Segments

Segments store all data related to fetch and parse of a single batch of URLs (one generate-fetch-update cycle)

crawl_generate: list of URLs to be fetched with politeness requirements (host-level blocking):

▶ all URLs of one host (or IP) must be in one partition to implement host-level blocking in one single JVM
▶ inside one partition URLs are shuffled: URLs of one host are spread over whole partition to minimize blocking during fetch

crawl_fetch: status from fetch (e.g., success, failed, robots denied)
content: fetched binary content and meta data (HTTP header)
parse_data: extracted meta data, outlinks and anchor texts
parse_text: plain-text content
crawl_parse: outlinks, scores, signatures, meta data, used to update CrawlDb
Storage and Data Flow in Nutch 1.x

- seed URLs
  - dedup
  - inject
  - CrawlDb
    - generate
    - crawl_fetch
    - content
    - parse
      - parse_data
      - parse_text
    - update
    - segment 1
      - crawl_generate (fetch list)
      - fetch
      - parse
      - index
    - segment 2
      - index
    - segment n
      - index
  - score updater
  - WebgraphDb
    - outlinks
    - nodes
    - inlinks
  - LinkDb
    - invert links

- map file
- sequence file
Storage in Nutch 2.x

One big table (WebTable)

- all information about one URL/document in a single row: status, metadata, binary content, extracted plain-text, inlink anchors, ...

- inspired by BigTable paper [3]

- …and rise of NoSQL data stores

Apache Gora used for storage layer abstraction

- OTD (object-to-datastore) for NoSQL data stores

- HBase, Cassandra, DynamoDb, Avro, Accumulo, ...

- data serialization with Avro

- back-end specific object-datastore mappings
Benefits of Nutch 2.x

Benefits

- less steps in crawler workflow
- simplify Nutch code base
- easy to access data from other applications
  - access storage directly
  - or via Gora and schema
- plugins and MapReduce processing shared with 1.x
Performance and Efficiency 2.x?

Objective: should be more performant (in terms of latency)
- no need to rewrite whole CrawlDb for each smaller update
- adaptable to more instant workflows (cf. [5])

...however
- benchmark by Julien Nioche [4] shows that Nutch 2.x is slower than 1.x by a factor of ≥2.5
- not a problem of HBase and Cassandra but due to Gora layer
- will hopefully be improved with Nutch 2.3 and recent Gora version which supports filtered scans
- still fast at scale
...and Drawbacks?

Drawbacks

- need to install and maintain datastore
- higher hardware requirements

Impact on Nutch as software project

- 2.x planned as replacement, however
  - still not as stable as 1.x
  - 1.x has more users

- need to maintain two branches
- keep in sync, transfer patches
News

Common Crawl moves to Nutch (2013)
http://blog.commoncrawl.org/2014/02/common-crawl-move-to-nutch/

- public data on Amazon S3
- billions of web pages

Generic deduplication (1.x)

- based on CrawlDb only
- no need to pull document signatures and scores from index
- no special implementations for indexing back-ends (Solr, etc.)

Nutch participated in GSoC 2014 ...
Nutch Web App

GSoC 2014 (Fjodor Vershinin):
“Create a Wicket-based Web Application for Nutch”

- Nutch Server and REST API
- Web App client to run and schedule crawls
- only Nutch 2.x (for now)
- needs completion: change configuration, analytics (logs), ...
What’s coming next?

Nutch 1.x is still alive!

...but we hope to release 2.3 soon
  ▶ with performance improvements from recent Gora release

Fix bugs, improve usability
  ▶ our bottleneck: review patches, get them committed
  ▶ try hard to keep 1.x and 2.x in sync

Pending new features
  ▶ sitemap protocol
  ▶ canonical links
What’s coming next? (mid- and long-term)

Move to Hadoop 2.x and new MapReduce API

... complete Rest API and Web App
... port it “back” to 1.x

... use graph library (Giraph) for link rank computation

... switch from batch processing to streaming
  ▶ Storm / Spark / ?
  ▶ parts of the workflow
Thanks and acknowledgments ...

... to Julien Nioche, Lewis John McGibbney, Andrzej Białecki, Chris Mattmann, Doug Cutting, Mike Cafarella – for inspirations and material from previous Nutch presentations

... and to all Nutch committers for review and comments on early versions of theses slides.
References


Questions?