

Software Heritage

Analyzing the Global Graph of Public Software Development

Stefano Zacchioli

Université de Paris & Inria – zack@upsilon.cc, [@zacchiro](https://twitter.com/zacchiro)

7 February 2020

LIRIS – Lyon, France



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Conclusion

Software is spread all around

Debian CPAN
Sourceforge Gitorious
Maven Inria
Bitbucket
Git GitHub
BerliOs CTAN
GoogleCode GitLab Adullact CRAN



A word cloud of terms related to software fragility, set against a background of a world map. The words are arranged in various sizes and orientations. The largest words are 'damage', 'disaster', 'malicious', 'deletion', and 'attack'. Other prominent words include 'obsolete', 'format', 'encryption', 'corruption', 'dangling', 'wear', 'dependencies', 'reference', 'storage', 'media', 'aging', and 'tear'.

damage
disaster
malicious
deletion
obsolete
format
encryption
corruption
dangling
wear
dependencies
reference
storage
media
aging
tear
attack



Software lacks its own research infrastructure



Photo: ALMA(ESO/NAOJ/NRAO), R. Hills





Software Heritage

THE GREAT LIBRARY OF SOURCE CODE



Our mission

Collect, **preserve** and **share** the *source code* of *all the software* that is publicly available.

Past, present and future

Preserving the past, enhancing the present, preparing the future.

Archiving goals

Targets: VCS repositories & source code releases (e.g., tarballs)

We DO archive

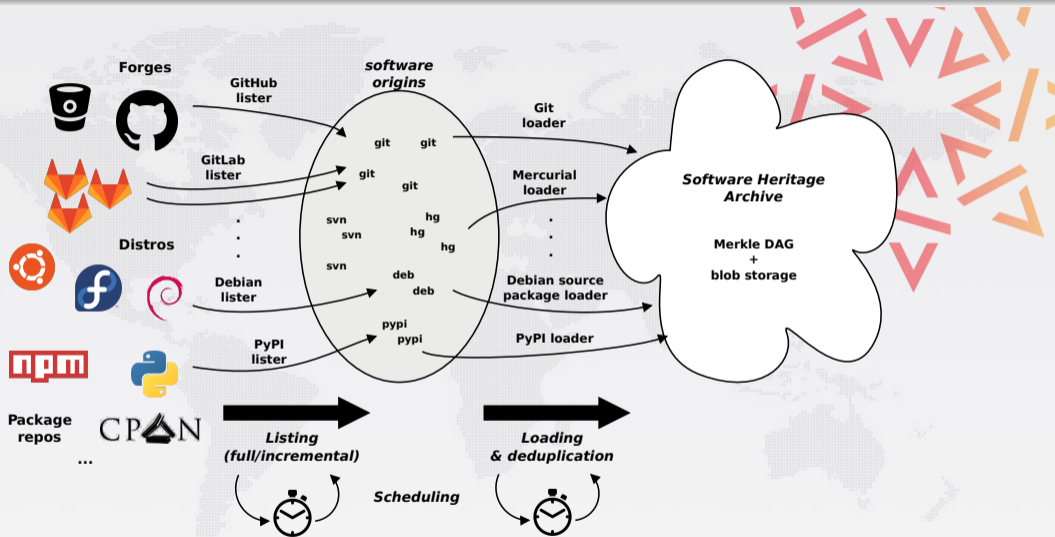
- file **content** (= blobs)
- **revisions** (= commits), with full metadata
- **releases** (= tags), ditto
- where (**origin**) & when (**visit**) we found any of the above

... in a VCS-/archive-agnostic **canonical data model**

We DON'T archive

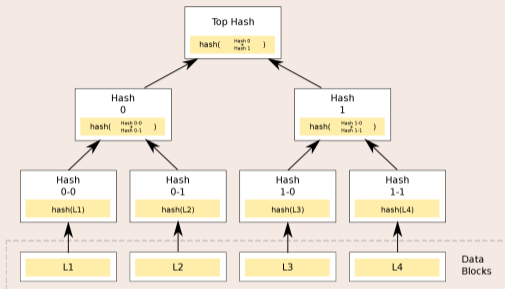
- homepages, wikis
- BTS/issues/code reviews/etc.
- mailing lists

Long term vision: play our part in a *"semantic wikipedia of software"*



Merkle trees

Merkle tree (R. C. Merkle, CRYPTO 1987)

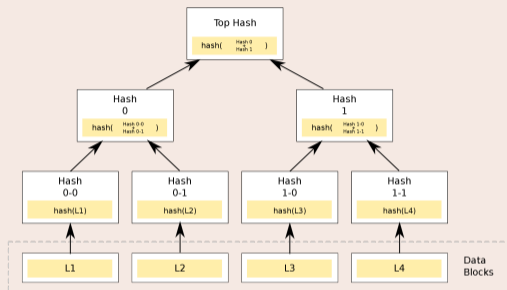


Combination of

- tree
- hash function

Merkle trees

Merkle tree (R. C. Merkle, CRYPTO 1987)



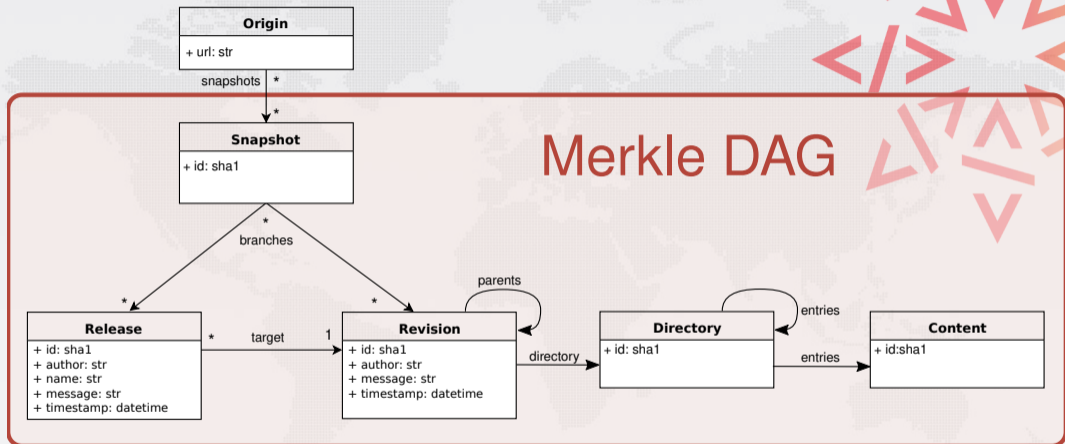
Combination of

- tree
- hash function

Classical cryptographic construction

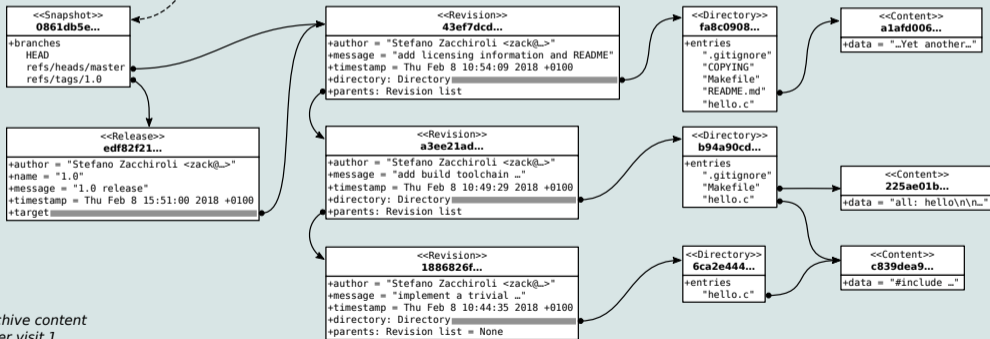
- fast, parallel signature of large data structures
- widely used (e.g., Git, blockchains, IPFS, ...)
- built-in deduplication

The archive: a (giant) Merkle DAG



The archive: a (giant) Merkle DAG

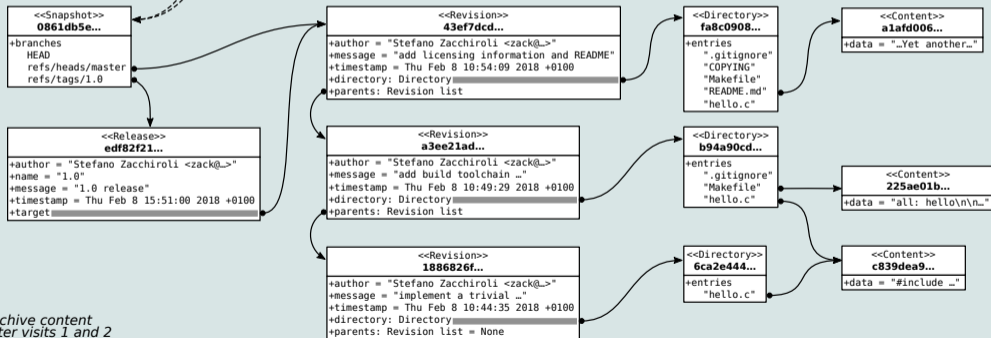
origin https://forge.softwareheritage.org/source/helloworld.git
visit 1
snapshot 0861db5e...
timestamp Fri Feb 9 12:38:45 2018 +0100



Archive content
after visit 1

The archive: a (giant) Merkle DAG

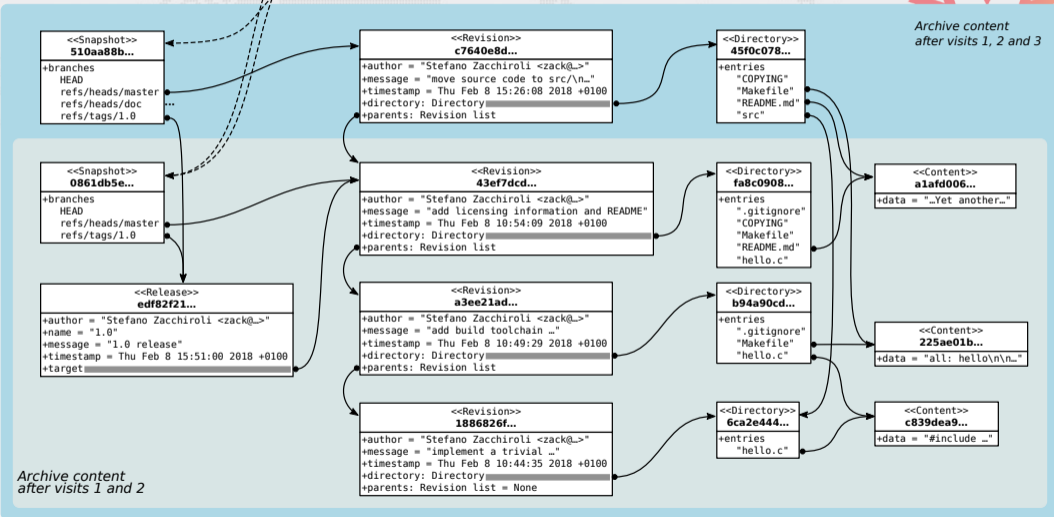
origin	visit	snapshot	timestamp
https://forge.softwareheritage.org/source/helloworld.git	1	0861db5e...	Fri Feb 9 12:38:45 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	2	0861db5e...	Fri Feb 9 13:29:00 2018 +0100

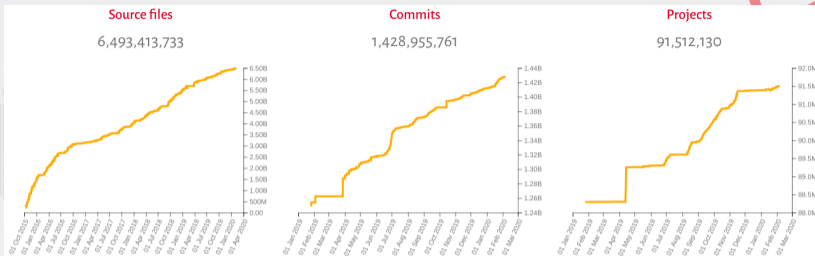


Archive content
after visits 1 and 2

The archive: a (giant) Merkle DAG

origin	visit	snapshot	timestamp
https://forge.softwareheritage.org/source/helloworld.git	1	0861db5e...	Fri Feb 9 12:38:45 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	2	0861db5e...	Fri Feb 9 13:29:00 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	3	510aa88b...	Fri Feb 9 15:52:50 2018 +0100





GitHub

debian



GitLab



Google code



GITORIOUS

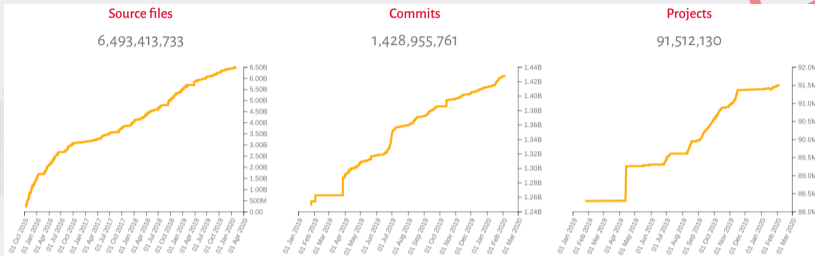


GNU

HAL
archives-ouvertes.fr

Inria
inventeurs du monde numérique

python™
Package Index



GitHub

debian



GitLab



Google code



GITORIOUS



GNU

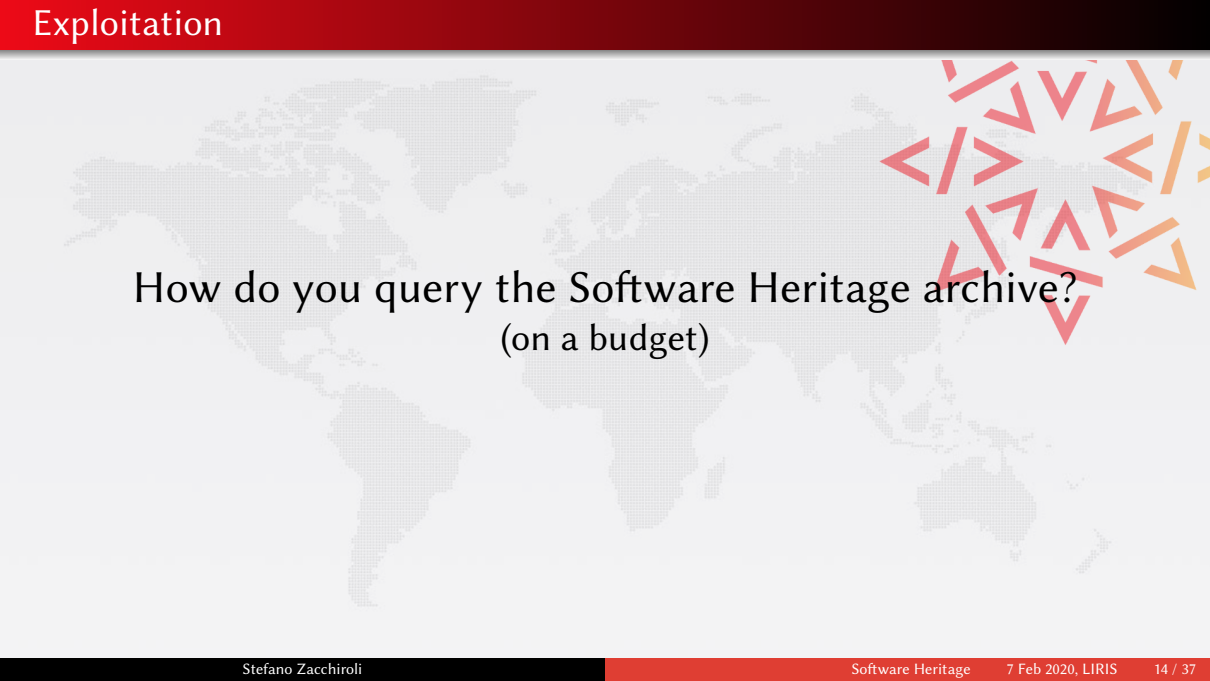
HAL
archives-ouvertes.fr

Inria
inventeurs du monde numérique

python™
Package Index

- ~400 TB (uncompressed) blobs, ~20 B nodes, ~300 B edges
- The *richest* public source code archive, ... and growing daily!

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Conclusion



How do you query the Software Heritage archive?
(on a budget)

e.g., for <https://archive.softwareheritage.org>

Browsing

- `ls`
- `git log` (Linux kernel: 800K+ commits)

Wayback machine

- `tarball`
- `git bundle` (Linux kernel: 7M+ nodes)

Provenance tracking

- commit provenance (one/all contexts)
 - requires backtracking
- origin provenance (one/all contexts)

For the sake of it


- local graph topology
- connected component size
 - enabling question to identify the best approach (e.g., scale-up v. scale-out) to conduct large-scale analyses
- any other emerging property

Software Engineering topics

- software provenance analysis at this scale is pretty much unexplored yet
- industry frontier: increase granularity down to the individual line of code
- replicate at this scale (famous) studies that have generally been conducted on (much) smaller version control system samples to confirm/refute their findings
- ...

Software Heritage Graph dataset

Use case: large scale analyses of the most comprehensive corpus on the development history of free/open source software.

 Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli
The Software Heritage Graph Dataset: Public software development under one roof
MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE
preprint: <http://deb.li/swhmsr19>

Dataset

- Relational representation of the full graph as a set of tables
- Available as open data: <https://doi.org/10.5281/zenodo.2583978>

Formats

- Local use: PostgreSQL dumps, or Apache Parquet files (~1 TiB each)
- Live usage: Amazon Athena (SQL-queriable), Azure Data Lake (soon)

Sample query — most frequent first commit words

```
1 SELECT COUNT(*) AS c, word FROM (  
2   SELECT LOWER(REGEXP_EXTRACT(FROM_UTF8(  
3     message), '^\\w+')) AS word FROM revision)  
4 WHERE word != ''  
5 GROUP BY word ORDER BY COUNT(*) DESC LIMIT 5;
```

Sample query — most frequent first commit words

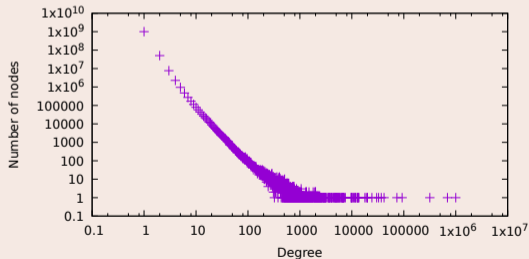
```
1 SELECT COUNT(*) AS c, word FROM (  
2   SELECT LOWER(REGEXP_EXTRACT(FROM_UTF8(  
3     message), '^\\w+')) AS word FROM revision)  
4 WHERE word != ''  
5 GROUP BY word ORDER BY COUNT(*) DESC LIMIT 5;
```

Count	Word
71 338 310	update
64 980 346	merge
56 854 372	add
44 971 954	added
33 222 056	fix

Fork arity

i.e., how often is a commit based upon?

```
1 SELECT fork_deg, count(*) FROM (  
2   SELECT id, count(*) AS fork_deg  
3   FROM revision_history GROUP BY id) t  
4 GROUP BY fork_deg ORDER BY fork_deg;
```

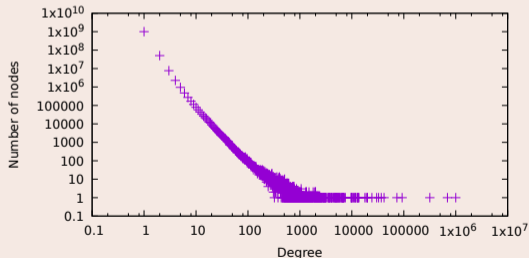


Sample query — fork and merge arities

Fork arity

i.e., how often is a commit based upon?

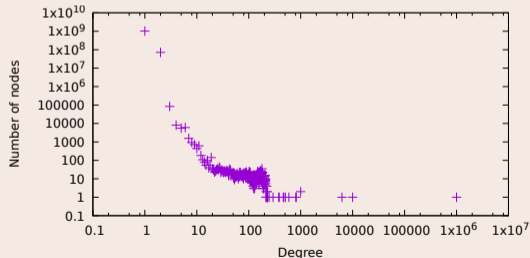
```
1 SELECT fork_deg, count(*) FROM (  
2   SELECT id, count(*) AS fork_deg  
3   FROM revision_history GROUP BY id) t  
4 GROUP BY fork_deg ORDER BY fork_deg;
```



Merge arity

i.e., how large are merges?

```
1 SELECT merge_deg, COUNT(*) FROM (  
2   SELECT parent_id, COUNT(*) AS merge_deg  
3   FROM revision_history GROUP BY parent_id  
4 GROUP BY deg ORDER BY deg;
```



Sample query – ratio of commits performed during weekends

```
1 WITH revision_date AS
2   (SELECT FROM_UNIXTIME(date / 1000000) AS date
3    FROM revision)
4 SELECT yearly_rev.year AS year,
5        CAST(yearly_weekend_rev.number AS DOUBLE)
6        / yearly_rev.number * 100.0 AS weekend_pc
7 FROM
8   (SELECT YEAR(date) AS year, COUNT(*) AS number
9    FROM revision_date
10   WHERE YEAR(date) BETWEEN 1971 AND 2018
11   GROUP BY YEAR(date) ) AS yearly_rev
12 JOIN
13   (SELECT YEAR(date) AS year, COUNT(*) AS number
14   FROM revision_date
15   WHERE DAY_OF_WEEK(date) >= 6
16         AND YEAR(date) BETWEEN 1971 AND 2018
17   GROUP BY YEAR(date) ) AS yearly_weekend_rev
18 ON yearly_rev.year = yearly_weekend_rev.year
19 ORDER BY year DESC;
```



Sample query — ratio of commits performed during weekends (cont.)


Year	Weekend	Total	Weekend percentage
2018	15130065	78539158	19.26
2017	33776451	168074276	20.09
2016	43890325	209442130	20.95
2015	35781159	166884920	21.44
2014	24591048	122341275	20.10
2013	17792778	88524430	20.09
2012	12794430	64516008	19.83
2011	9765190	48479321	20.14
2010	7766348	38561515	20.14
2009	6352253	31053219	20.45
2008	4568373	22474882	20.32
2007	3318881	16289632	20.37
2006	2597142	12224905	21.24
2005	2086697	9603804	21.72
2004	1752400	7948104	22.04

Sample query — average size of the most popular file types

```
1 SELECT suffix,
2     ROUND(COUNT(*) * 100 / 1e6) AS Million_files,
3     ROUND(AVG(length) / 1024) AS Average_k_length
4 FROM
5     (SELECT length, suffix
6      FROM -- File length in joinable form
7           (SELECT TO_BASE64(sha1_git) AS sha1_git64, length
8            FROM content ) AS content_length
9      JOIN -- Sample of files with popular suffixes
10          (SELECT target64, file_suffix_sample.suffix AS suffix
11           FROM -- Popular suffixes
12                (SELECT suffix FROM (
13                   SELECT REGEXP_EXTRACT(FROM_UTF8(name),
14                     '\.[^.]+$') AS suffix
15                   FROM directory_entry_file) AS file_suffix
16                  GROUP BY suffix
17                  ORDER BY COUNT(*) DESC LIMIT 20 ) AS pop_suffix
18          JOIN -- Sample of files and suffixes
19              (SELECT TO_BASE64(target) AS target64,
20               REGEXP_EXTRACT(FROM_UTF8(name),
21                 '\.[^.]+$') AS suffix
22              FROM directory_entry_file TABLESAMPLE BERNOULLI(1))
23          AS file_suffix_sample
```



- one *can* query such a corpus SQL-style
- but relational representation shows its limits at this scale
- ...at least as deployed on commercial SQL offerings such as Athena
- note: (naive) sharding is ineffective, due to the pseudo-random distribution of node identifiers
- experiments with Google BigQuery are ongoing
 - (we broke it at the first import attempt..., due to very large arrays in directory entry tables)

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Conclusion

 Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchiroli
Ultra-Large-Scale Repository Analysis via Graph Compression
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering.
IEEE

Research question

Is it possible to efficiently perform software development history analyses at ultra large scale (= the scale of Software Heritage archive or more), on a single, relatively cheap machine?

Idea

Apply state-of-the-art graph compression techniques from the field of Web graph / social network analysis.

Borrowing (great!) slides from:

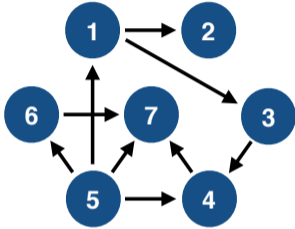


Giulio Ermanno Pibiri

Effective Web Graph Representations, 2018

http://pages.di.unipi.it/pibiri/slides/webgraphs_compression.pdf

Context - Web Graphs



Conceptual graph

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	1
1	0	0	1	0	1	1
0	0	0	0	0	0	1
0	0	0	0	0	0	0

Adjacency **matrix**

1: 2,3
2: -
3: 4
4: 7
5: 1,4,6,7
6: 7
7: -

Adjacency **lists**

Many results are known for
compressing
integer sequences.

The WebGraph Framework

Java/C++ framework consisting in algorithms and compression codes for managing large Web Graphs.

<http://webgraph.di.unimi.it/>

The WebGraph Framework I: Compression Techniques, Boldi-Vigna, WWW 2004

Locality - pages links to pages whose URL is lexicographically similar. URLs share long common prefixes.

Use *d-gap* compression.

Similarity - pages that are close together in lexicographic order, tend to have many common successors.

Use *reference* compression.

The WebGraph Framework

Exploiting **locality**.

If we have: $x: [y_1, \dots, y_k]$, then we represent
 $[y_1 - x, y_2 - y_1 - 1, y_3 - y_2 - 1, \dots, y_k - y_{k-1} - 1]$

First gap $d = y_1 - x$ is represented as $2d$ if $d \geq 0$ or $2|d|-1$ if $d < 0$

Node	Outdegree	Successors
...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...

Adjacency lists

Node	Outdegree	Successors
...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...

***d*-gapped** adjacency lists

The WebGraph Framework

Exploiting **similarity**.

Idea: use reference compression, i.e., represent a list with respect to another one called its **reference list**.

	Node	Outdegree	Successors

1	15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
2	16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
3	17	0	
4	18	5	13, 15, 16, 17, 50

Adjacency lists

Node	Outd.	Ref.	Copy list	Extra nodes
...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...

Copy lists

Nodes

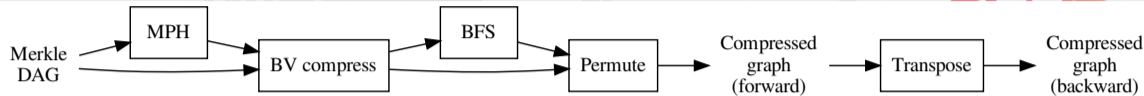
Node type	N. of nodes
origins	88 M
snapshots	57 M
releases	9.9 M
revisions	1.1 B
directories	4.9 B
contents	5.5 B
Total nodes	12 B

Edges

Edge type	N. of edges
origin → snapshot	195 M
snapshot → revision	616 M
snapshot → release	215 M
release → revision	9.9 M
revision → revision	1.2 B
revision → directory	1.1 B
directory → directory	48 B
directory → revision	482 M
directory → content	112 B
Total edges	165 B

Archive snapshot 2018-09-25, from the Software Heritage graph dataset.
Growth rate: exponential, doubling every 22-30 months.

Graph compression pipeline



- **MPH**: minimal perfect hash, mapping Merkle IDs to 0..N-1 integers
- **BV compress**: Boldi-Vigna compression (based on MPH order)
- **BFS**: breadth-first visit to renumber
- **Permute**: update BV compression according to BFS order

(Re)establishing locality

- key for good compression is a node ordering that ensures locality and similarity
- which is very much *not* the case with Merkle IDs...
- ...but is the case *again* after BFS

Step	Wall time (hours)
MPH	2
BV Compress	84
BFS	19
Permute	18
Transpose	15
Total	138 (6 days)

- server equipped with 24 CPUs and 750 GB of RAM
- RAM mostly used as I/O cache for the BFS step
- *minimum* memory requirements are close to the RAM needed to load the final compressed graph in memory

Compression efficiency (space)

Forward graph

total size	91 GiB
bits per edge	4.91
compression ratio	15.8%

Backward graph

total size	83 GiB
bits per edge	4.49
compression ratio	14.4%

Operation cost

The structure of a full bidirectional archive graph fits in less than 200 GiB of RAM, for a hardware cost of ~300 USD.

Compression efficiency (time)

Benchmark — Full BFS visit

Forward graph

wall time	1h48m
throughput	1.81 M nodes/s (553 ns/node)

Backward graph

wall time	3h17m
throughput	988 M nodes/s (1.01 μ s/node)

Benchmark — Edge lookup

random sample: 1 B nodes (8.3% of entire graph)

Forward graph

visited edges	13.6 B
throughput	12.0 M edges/s (83 ns/edge)

Backward graph

visited edges	13.6 B
throughput	9.45 M edges/s (106 ns/edge)

Note how edge lookup time is close to DRAM random access time (50-60 ns).

Incrementality

compression is **not incremental**, due to the use of contiguous integer ranges

- but the graph is append-only, so...
- ...based on expected graph growth rate it should be possible to pre-allocate enough free space in the integer ranges to support **amortized incrementality** (future work)

Incrementality


compression is **not incremental**, due to the use of contiguous integer ranges

- but the graph is append-only, so...
- ...based on expected graph growth rate it should be possible to pre-allocate enough free space in the integer ranges to support **amortized incrementality** (future work)

In-memory v. on-disk

the compressed in-memory graph structure has **no attributes**

- usual design is to exploit the 0..N-1 integer ranges to **memory map node attributes** to disk for efficient access
- works well for queries that does graph traversal first and "join" node attributes last; ping-pong between the two is expensive
- edge attributes are more problematic

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Conclusion

We're hiring! (a postdoc)

Paris-based postdoc on software provenance

- large-scale, big data **graph analysis**
- tracking the **provenance of source code** artifacts
- ... at the **scale of the world** (what else?)
- in the context of **industrial partnerships** on open source license compliance
- supervision: Stefano Zacchiroli, Roberto Di Cosmo

Learn more and apply

- <https://softwareheritage.org/jobs/>
- ask me! zack@upsilon.cc

Wrapping up

- Software Heritage archives all public source code as a huge Merkle DAG
- Querying and analyzing it pose scaling challenges (20/300 B nodes/edges)
- It is a gold mine of research leads for graph/database scholars. Wanna join?

References



Jean-François Abramatic, Roberto Di Cosmo, Stefano Zacchiroli
Building the Universal Archive of Source Code
Communications of the ACM, October 2018



Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli
The Software Heritage graph dataset: public software development under one roof
MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE



Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchiroli
Ultra-Large-Scale Repository Analysis via Graph Compression
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering. IEEE

Contacts

Stefano Zacchiroli / zack@upsilon.cc / @zacchiro