Software Heritage

Source Code Archival and Analysis at the Scale of the World

Stefano Zacchiroli

Univ. Paris Diderot & Inria — zack@upsilon.cc, @zacchiro

14 November 2019
University of Zurich — Zurich, Switzerland

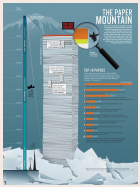
Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

Software Source code: pillar of Open Science

Software is everywhere in modern research



[…] software […] essential in their fields.

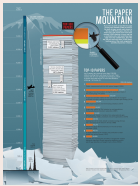
Top 100 papers (Nature, 2014)

Sometimes, if you dont have the software, you dont have the data

Christine Borgman, Paris, 2018

Software Source code: pillar of Open Science

Software is everywhere in modern research



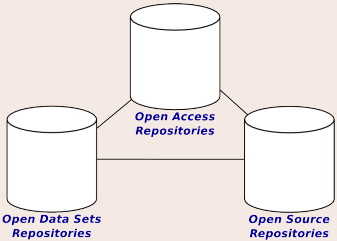
[…] software […] essential in their fields.

Top 100 papers (Nature, 2014)

Sometimes, if you dont have the software, you dont have the data

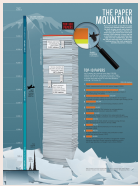
Christine Borgman, Paris, 2018

Open Science: three pillars



Software Source code: pillar of Open Science

Software is everywhere in modern research



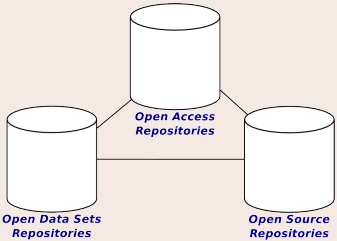
[…] software […] essential in their fields.

Top 100 papers (Nature, 2014)

Sometimes, if you dont have the software, you dont have the data

Christine Borgman, Paris, 2018

Open Science: three pillars



Nota bene

The links in the picture are essential

Source code is *special*

Harold Abelson, Structure and Interpretation of Computer Programs (1st ed.) 1985

"Programs must be written for people to read, and only incidentally for machines to execute."

Source code is *special*

Harold Abelson, Structure and Interpretation of Computer Programs (1st ed.) 1985

"Programs must be written for people to read, and only incidentally for machines to execute."

Apollo 11 source code (excerpt)

```
P63SPOT3        CA      BIT6            # IS THE LR ANTENNA IN POSITION 1 YET
                EXTEND
                RAND    CHAN33
                EXTEND
                BZF     P63SPOT4        # BRANCH IF ANTENNA ALREADY IN POSITION 1

                CAF     CODE500         # ASTRONAUT:    PLEASE CRANK THE
                TC      BANKCALL        #               SILLY THING AROUND
                CADR    GOPERF1
                TCF     GOTOPOOH        # TERMINATE
                TCF     P63SPOT3        # PROCEED       SEE IF HE'S LYING

P63SPOT4        TC      BANKCALL        # ENTER         INITIALIZE LANDING RADAR
                CADR    SETPOS1

                TC      POSTJUMP        # OFF TO SEE THE WIZARD ...
                CADR    BURNBABY
```

Source code is *special*

Harold Abelson, Structure and Interpretation of Computer Programs (1st ed.) 1985

"Programs must be written for people to read, and only incidentally for machines to execute."

Apollo 11 source code (excerpt)

```
P63SPOT3        CA      BIT6            # IS THE LR ANTENNA IN POSITION 1 YET
                EXTEND
                RAND    CHAN33
                EXTEND
                BZF     P63SPOT4        # BRANCH IF ANTENNA ALREADY IN POSITION 1

                CAF     CODE500         # ASTRONAUT:    PLEASE CRANK THE
                TC      BANKCALL        #               SILLY THING AROUND
                CADR    GOPERF1
                TCF     GOTOPOOH        # TERMINATE
                TCF     P63SPOT3        # PROCEED       SEE IF HE'S LYING

P63SPOT4        TC      BANKCALL        # ENTER         INITIALIZE LANDING RADAR
                CADR    SETPOS1

                TC      POSTJUMP        # OFF TO SEE THE WIZARD ...
                CADR    BURNBABY
```

Quake III source code (excerpt)

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y; // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
// y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this
can be removed

    return y;
}
```

Source code is *special*

Harold Abelson, Structure and Interpretation of Computer Programs (1st ed.) 1985

"Programs must be written for people to read, and only incidentally for machines to execute."

Apollo 11 source code (excerpt)

```
P63SPOT3        CA      BIT6            # IS THE LR ANTENNA IN POSITION 1 YET
                EXTEND
                RAND    CHAN33
                EXTEND
                BZF     P63SPOT4        # BRANCH IF ANTENNA ALREADY IN POSITION 1

                CAF     CODE500         # ASTRONAUT:    PLEASE CRANK THE
                TC      BANKCALL        #               SILLY THING AROUND
                CADR    GOPERF1
                TCF     GOTOPOOH        # TERMINATE
                TCF     P63SPOT3        # PROCEED       SEE IF HE'S LYING

P63SPOT4        TC      BANKCALL        # ENTER         INITIALIZE LANDING RADAR
                CADR    SETPOS1

                TC      POSTJUMP        # OFF TO SEE THE WIZARD ...
                CADR    BURNBABY
```

Quake III source code (excerpt)

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y; // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
// y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this
can be removed

    return y;
}
```

Len Shustek, Computer History Museum

"Source code provides a view into the mind of the designer."

The state of the art is not ideal

Analysis of 613 papers

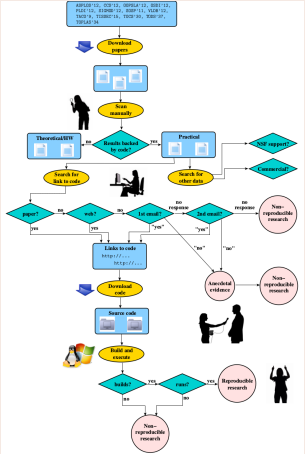
- 8 ACM conferences: ASPLOS'12, CCS'12, OOPSLA'12, OSDI'12, PLDI'12, SIGMOD'12, SOSP'11, VLDB'12
- 5 journals: TACO'9, TISSEC'15, TOCS'30, TODS'37, TOPLAS'34

all very practical oriented

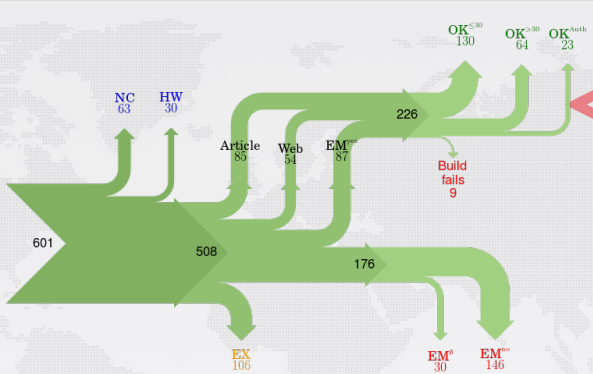
The basic question

can we get the code to build and run?

The workflow



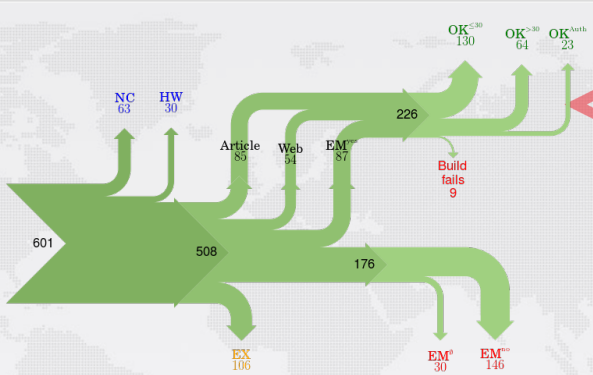
The state of the art is not ideal… (cont.)



…

… that's a whopping 40% of non reproducible works!

The state of the art is not ideal… (cont.)



…

… that's a whopping 40% of non reproducible works!

The main reason

source code (*or the right version of it*) cannot be found

Software is fragile

Like all digital information, FOSS is fragile

- inconsiderate and/or malicious code loss (e.g., Code Spaces)
- business-driven code loss (e.g., Gitorious, Google Code)
- for obsolete code: physical media decay (data rot)

Software is fragile

Like all digital information, FOSS is fragile

- inconsiderate and/or malicious code loss (e.g., Code Spaces)
- business-driven code loss (e.g., Gitorious, Google Code)
- for obsolete code: physical media decay (data rot)

Where is the archive…

where do we go if (a repository on) GitHub or GitLab.com goes away?

Software is spread all around

Fashion victims

- many disparate development platforms
- a myriad places where distribution may happen
- projects tend to migrate from one place to another over time

Software is spread all around

Fashion victims

- many disparate development platforms
- a myriad places where distribution may happen
- projects tend to migrate from one place to another over time

Where is the place …

where we can find, track and search *all* source code?

Software lacks its own research infrastructure

A wealth of software research on crucial issues…

- safety, security, test, verification, proof
- software engineering, software evolution
- big data, machine learning, empirical studies

Software lacks its own research infrastructure

A wealth of software research on crucial issues…

- safety, security, test, verification, proof
- software engineering, software evolution
- big data, machine learning, empirical studies

If you study the stars, you go to Atacama…

… where is the *very large telescope* of source code?

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

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.

Our principles

Our principles

Open approach

- open source
- transparency

In for the long haul

- non profit
- replication & mirrors

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

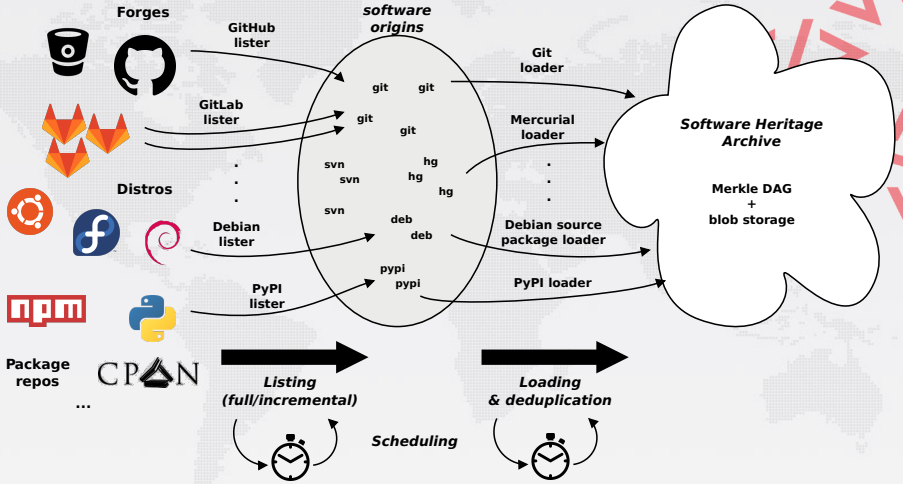
… in a VCS-/archive-agnostic canonical data model

We DON'T archive

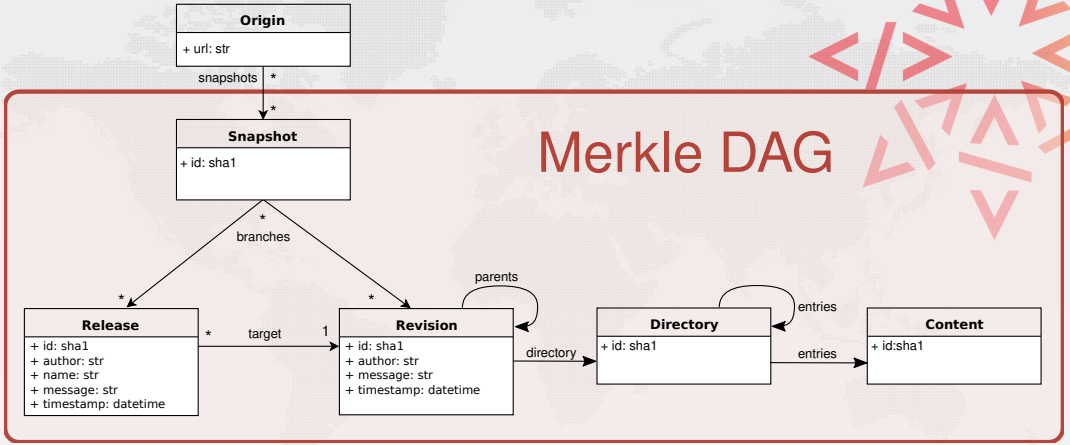
- homepages, wikis
- BTS/issues/code reviews/etc.
- mailing lists

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

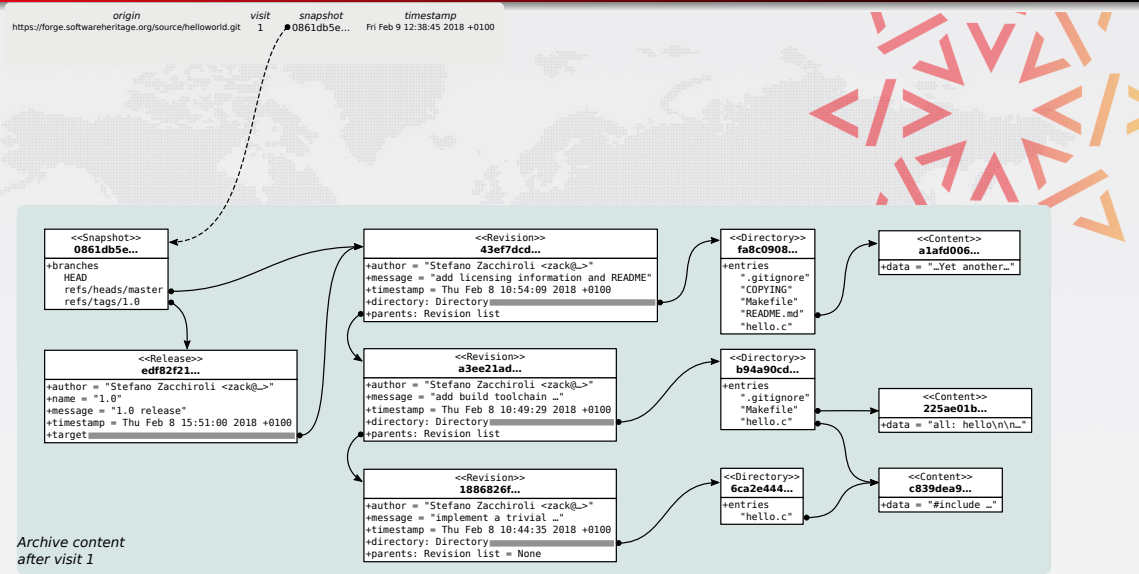
Data flow


Forges
GitHub lister
GitLab lister
Distros
Debian lister
PyPI lister
Package repos
CPAN
...
software origins
git
git
git
git
svn
svn
svn
hg
hg
hg
deb
deb
pypi
pypi
Git loader
Mercurial loader
Debian source package loader
PyPI loader
Software Heritage Archive
Merkle DAG
+
blob storage
Listing (full/incremental)
Scheduling
Loading & deduplication

The archive: a (giant) Merkle DAG



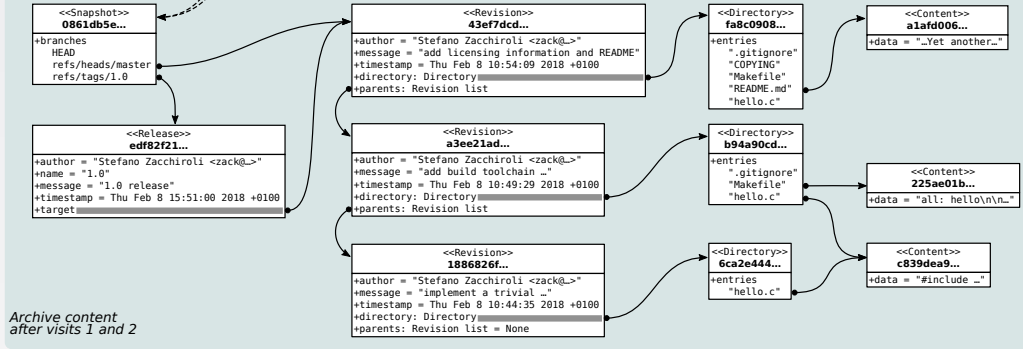
The archive: a (giant) Merkle DAG



Archive content after visit 1

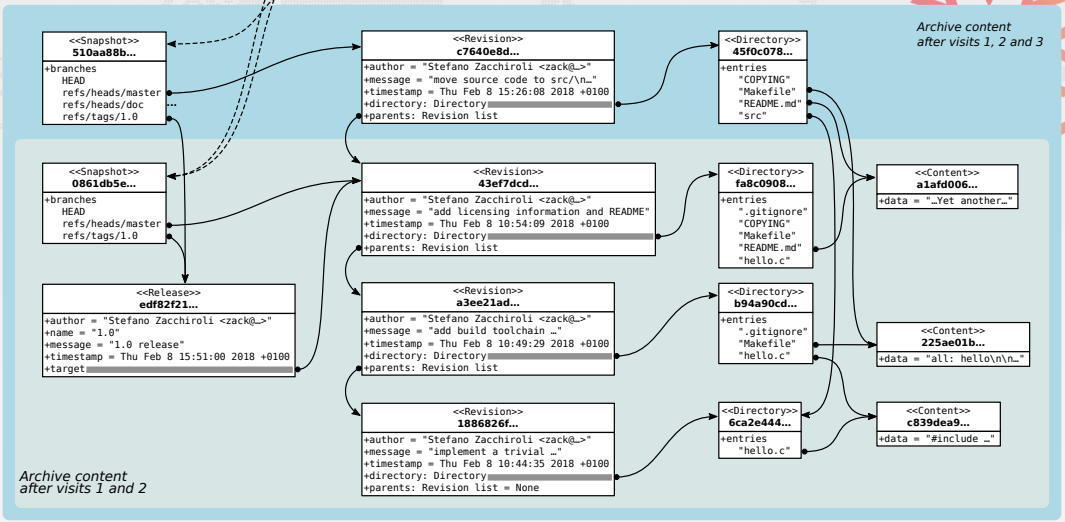
The archive: a (giant) Merkle DAG

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

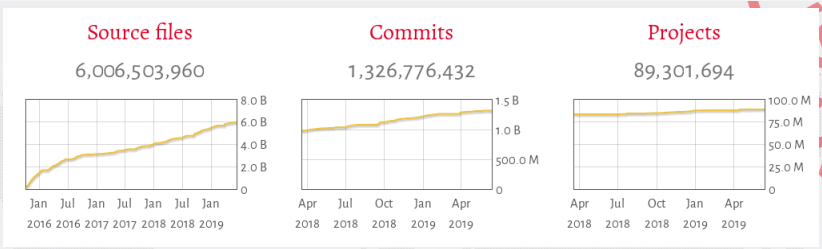


The archive: a (giant) Merkle DAG

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



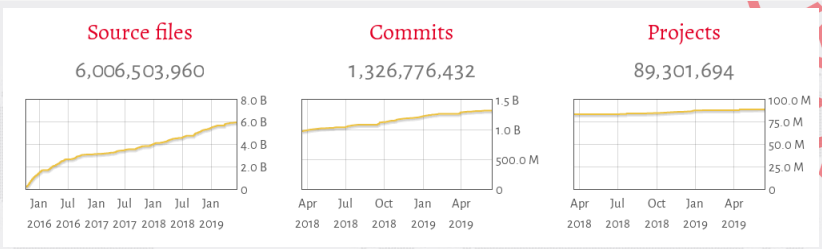
Archive coverage — archive.softwareheritage.org



GitHub debian GitLab npm Google code {P}

GITORIOUS GNU HAL archives-ouvertes.fr Inria inventeurs du monde numérique python Package Index

Archive coverage — archive.softwareheritage.org



GitHub · debian · GitLab · npm · Google code · {P} · python Package Index · GITORIOUS · GNU · HAL archives-ouvertes.fr · Inria inventeurs du monde numérique

- ~400 TB (uncompressed) blobs, ~20 B nodes, ~280 B edges
- The *richest* public source code archive, … and growing daily!

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

Realizing the "large telescope of source code" in practice

Requirements

- Availability: Software Heritage mirror, relatively up-to-date
- Efficiency: massive computing resources with fast access to the mirror
- Sustainability: pay-per-use or bring-your-own-computing

Challenges

- mirroring
- compression
- efficient processing
- experiments description language
- big code analysis (i.e., ML on source code)

… at the scale of the world!

Challenge #1: Mirroring

Thomas Jefferson, February 18, 1791

Let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident.

Challenge #1: Mirroring

Thomas Jefferson, February 18, 1791

Let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident.

Mirroring: the good

- big, but not *that* big
- append-only archive (in theory), easy to journal and incrementally update

Challenge #1: Mirroring (cont.)

Mirroring: the bad — Merkle DAGs with "holes"

- the world sucks: corrupted repositories, takedown notices, data losses
  - nodes can go missing at archival time or disappear later on
- top-level hash(es) no longer capture the full state of the archive

Open questions

- how do you capture such full state then?
  - by extension: how do you *timestamp* the archive?
- how do you efficiently check if something is to be re-archived?
- ultimately, what's your notion of having "fully archived" something?

Challenge #2: Compression — file contents

Storage figures

- file contents: ~400 TB (raw), ~200 TB compressed (1-by-1 content compression)
- median compressed size: 3 KB ⟶ a lot (~6 B) of very small files

Practical problem: scale-out object storages are not designed for this workload.

Challenge #2: Compression — file contents

Storage figures

- file contents: ~400 TB (raw), ~200 TB compressed (1-by-1 content compression)
- median compressed size: 3 KB ⟶ a lot (~6 B) of very small files

Practical problem: scale-out object storages are not designed for this workload.

Content compression

- low compression ratio (2x) with 1-by-1 compression
- typical Git/VCS packing heuristics do not work here, because contents occur in many different contexts
- early experiences with Rabin-style compression & co. were unsatisfactory
  - increased deduplication granularity (e.g., SLOCs) will likely suffer of the same problem
- research lead: object packing, using heuristics that maximize the chances of compressability (e.g., having *a* file name in common)

Challenge #2: Compression — graph

Storage figures

- Merkle DAG: ~20 B nodes, ~280 B edges
  - breakdown by node type: ~40% contents, ~40% directories, ~10% commits

Still outside the limits of what (cheaply & trivially) fits in RAM.

Challenge #2: Compression — graph

Storage figures

- Merkle DAG: ~20 B nodes, ~280 B edges
  - breakdown by node type: ~40% contents, ~40% directories, ~10% commits

Still outside the limits of what (cheaply & trivially) fits in RAM.

Graph compression

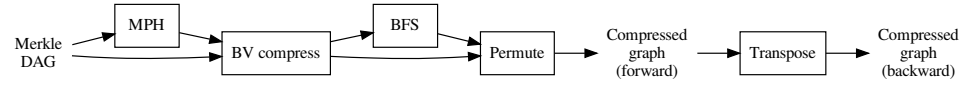
- related: Web graph compression techniques in the style of, e.g.:

  Paolo Boldi, Sebastiano Vigna
  The WebGraph framework I: compression techniques
  WWW 2004

- challenges: no canonical identifiers with good locality properties, different node types/subgraphs, live updates
- research lead: graph topology characterization

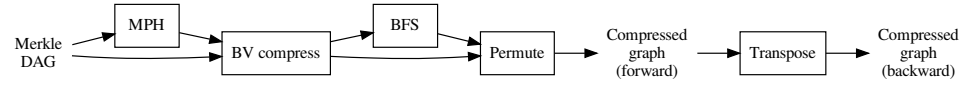
Graph compression — preliminary results



Dataset

- archive snapshot 2018-09-25
- size: 12 B nodes, 165 B edges

Graph compression — preliminary results



Dataset

- archive snapshot 2018-09-25
- size: 12 B nodes, 165 B edges

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%

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

Graph compression — preliminary results (cont.)

Analysis efficiency (time) — 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 µs/node)

Analysis efficiency (time) — 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).

Challenge #3: Efficient processing — graph

Big graphs calls for scale-out processing, e.g.:

Malewicz, Grzegorz, et al.
Pregel: a system for large-scale graph processing
ACM SIGMOD 2010

Roy, Amitabha, et al.
Chaos: Scale-out graph processing from secondary storage
ACM SOSP 2015

support very large graphs, exploiting topological characteristics (e.g., small world)

Software Heritage graph characteristics

- scale-free, not small world (?)
- connected components size distribution unclear yet

Approach: topological characterization

- if amenable to scale-out ⟶ distribution
- if not ⟶ scale-up using graph compression

Challenge #3: Efficient processing — file contents

Code search: a natural need

Find code snippets for reuse, find reverse dependencies for maintenance, impact analysis, etc.

Approaches vary with their level of code "understanding"

- full-text search: treat source code as text
- symbol extraction (e.g., ctags)
- AST search (language-specific)

Challenge #3: Efficient processing — file contents (cont.)

Russ Cox

Regular Expression Matching with a Trigram Index or How Google Code Search Worked

2012, https://swtch.com/ rsc/regexp/regexp4.html

Use regexs, Luke!

- build an inverted index of trigrams for all source code files
- compile regex to trigrams, find *potential* matches
- `grep` each potential match using map reduce

Application

- Google Code Search (now gone)
- Debsources (`https://sources.debian.org`)
  - scale up to 1 B SLOCs (Debian development branch)

Challenge

Scale it up to Software Heritage, several orders of magnitude later

Challenge #4: Experiment definition

Problem

How would researchers define their experiments on the dataset?

- corpus selection (code? history? sampling? etc.)
- how will they run their tools in a fully-deduplicated world?
- job scheduling (well-known HPC problem)

Related work (at a smaller scale)

Robert Dyer et al.
Boa: Ultra-large-scale software repository and source-code mining
ACM TOSEM 25.1 (2015): 7

Approach

formal languages, DSLs, tool adapters/porting

Challenge #5: Machine Learning on code

Big Code = Big Data + Source Code

Popular research trends: NLP on code / SBSE using ML, e.g.:

Miltiadis Allamanis et al.
A survey of machine learning for big code and naturalness
ACM Computing Surveys

Xiaodong Gu, Hongyu Zhang, Sunghun Kim
Deep Code Search
ICSE 2018

Common assumptions: AST availability, "small" datasets (e.g., random GitHub samples)

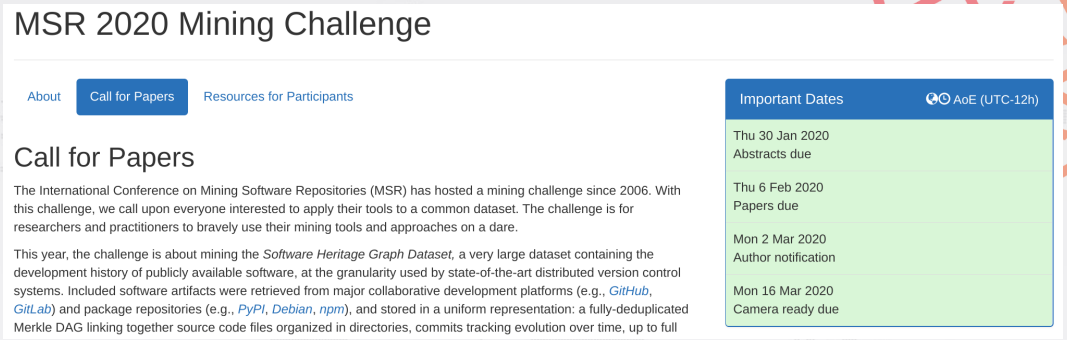
Example (Universal programming language detection)

- programming language detection at the scale of Software Heritage
- no AST, only bytes (not characters!)
- supervised or unsupervised (would be best for language evolution)

Outline

1. Source code in science
2. Software Heritage
3. Research challenges & roadmap
4. Conclusion

MSR 2020 Mining Challenge = Software Heritage Graph Dataset



MSR 2020 Mining Challenge

About | Call for Papers | Resources for Participants

Call for Papers

The International Conference on Mining Software Repositories (MSR) has hosted a mining challenge since 2006. With this challenge, we call upon everyone interested to apply their tools to a common dataset. The challenge is for researchers and practitioners to bravely use their mining tools and approaches on a dare.

This year, the challenge is about mining the *Software Heritage Graph Dataset,* a very large dataset containing the development history of publicly available software, at the granularity used by state-of-the-art distributed version control systems. Included software artifacts were retrieved from major collaborative development platforms (e.g., *GitHub*, *GitLab*) and package repositories (e.g., *PyPI*, *Debian*, *npm*), and stored in a uniform representation: a fully-deduplicated Merkle DAG linking together source code files organized in directories, commits tracking evolution over time, up to full

Important Dates — AoE (UTC-12h)

Thu 30 Jan 2020
Abstracts due

Thu 6 Feb 2020
Papers due

Mon 2 Mar 2020
Author notification

Mon 16 Mar 2020
Camera ready due

https://2020.msrconf.org/track/msr-2020-mining-challenge

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

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://www.softwareheritage.org/jobs/`
- ask me! zack@upsilon.cc

Wrapping up

- Software Heritage archives all software source code with its development history.
- It is a major endeavor that benefits society, science, and industry.
- For computer scientists, it is a gold mine of research opportunities. Wanna join?

References

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

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

Roberto Di Cosmo, Stefano Zacchiroli
Software Heritage: Why and How to Preserve Software Source Code
iPRES 2017: Intl. Conf. on Digital Preservation

Contacts

Stefano Zacchiroli / zack@upsilon.cc / @zacchiro

Software Heritage Graph dataset

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

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)

References and sample queries

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

Graph dataset — sample queries

Most frequent first commit words

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

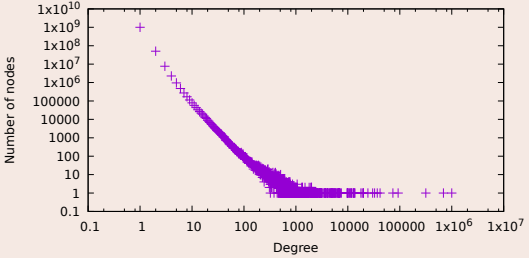
Count	Word
71'338'310	update
64'980'346	merge
56'854'372	add
44'971'954	added
33'222'056	fix

Graph dataset — sample queries

Fork arity

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

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



Merge arity

i.e., how large are merges?

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