Fast traversal of the SWH graph

Overview of available APIs

Valentin Lorentz (@vlorentz)

Software Heritage
Fondation INRIA

2024-10-04 - SWHSec Plenary

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Outline

The APIs I am *not* going to talk about

The HTTP API

- https://archive.softwareheritage.org/api/1/
- Simple to use, but for small amounts of data
- Only downward

The GraphQL API

- https://archive.softwareheritage.org/graphql/
- For small amounts of data
- Supports limited recursion, only downward

ORC exports

- https://docs.softwareheritage.org/devel/swh-dataset/graph/dataset.html
- Impractical to use for most workflows

swh-graph

Compressed graph representation

- Uses an ORC export as input
- Takes a few weeks to compress
- Fits the graph structure in 2 × 150GB
- And the rest in 12TB
- Does *not* include files' contents

Implementation

- Based on webgraph (https://webgraph.di.unimi.it/)
- Written in Rust (formerly Java) and a bit of Python

The APIs I *am* going to talk about

The graph HTTP API

- https://docs.softwareheritage.org/devel/swh-graph/api.html
- Simple to use

The gRPC API

- https://docs.softwareheritage.org/devel/swh-graph/grpc-api.html
- More flexible

The Rust API

- https://docs.rs/swh-graph/
- Most efficient and flexible, but requires writing Rust

Outline

HTTP API: How to access it

Hosted by SWH

- https://archive.softwareheritage.org/api/1/graph/
- Requires authorization from SWH
- No availability guarantees

Host it yourself

- Data available at s3://softwareheritage/graph/2024-05-16/compressed
- Requires 12TB of disk and at least 500GB of RAM for the full dataset

HTTP API: Capabilities

`/graph/neighbors/:src`

- Of a single node

`/graph/leaves/:src`

- Set a single source node
- Filter by node type, configurable resource limits
- Traverses the induced subgraph, and streams leaves

`/graph/visit/nodes/:src` and `/graph/visit/edges/:src`

- Ditto, but streams all the nodes/edges in the induced subgraph

HTTP API: Examples

Origins that contained a particular commit

```
/graph/leaves/swh:1:rev:82ecc538193b380a21622aea02.../
?direction=backward&resolve_origins=true
```

HTTP API: Examples

Origins that contained a particular commit

```
/graph/leaves/swh:1:rev:82ecc538193b380a21622aea02.../
?direction=backward&resolve_origins=true
```

Every SWHID in https://git.tukaani.org/xz.git

```
$ echo -n 'https://git.tukaani.org/xz.git'\
    | sha1sum
a1811f6ee1bc5aa0f8d06f1971ac288f19aa07bf

/graph/visit/nodes/
swh:1:ori:a1811f6ee1bc5aa0f8d06f1971ac288f19aa07bf/
/graph/visit/edges/
swh:1:ori:a1811f6ee1bc5aa0f8d06f1971ac288f19aa07bff/
```

Outline

gRPC API: How to access it

Not hosted by SWH

- (It may be some day, but there are no plans for now)

Host it yourself

- Data available at s3://softwareheritage/graph/2024-05-16/compressed
- Requires 12TB of disk and at least 500GB of RAM for the full dataset

gRPC

A framework for fast RPC

- Built on top of HTTP/2
- Very efficient, based on binary protocols
- Requires a schema, but supports introspection
- Support streaming in both directions
- Built-in support for servers to omit fields clients are not interested in

Multi-language

- Available in Rust, Python, Java, …
- Has a couple of CLI tools too

SWH's Graph Traversal gRPC API

Underlies the HTTP API

For example, `/graph/leaves/:src` is implemented as:

```
with grpc.insecure_channel(HOSTNAME) as channel:
    stub=swhgraph_grpc.TraversalServiceStub(channel)
    traversal_request = TraversalRequest(
        src=[src],
        mask=FieldMask(paths=["swhid"]),
        return_nodes=NodeFilter(
            max_traversal_successors=0
        )
    )
    async for node in stub.Traverse(
            traversal_request):
        yield stream_line(node)
```

SWH's Graph Traversal gRPC API

Documentation

- https://docs.softwareheritage.org/devel/swh-graph/grpc-api.html
- https://gitlab.softwareheritage.org/swh/devel/swh-graph/-/blob/master/proto/swhgraph.proto

Example: Getting a content node

CLI

```
$ grpc_cli call localhost:50091 \
 swh.graph.TraversalService.GetNode \
 'swhid: "swh:1:cnt:94a9ed024d3859793618152e..."'
```

Python

```
swhid="swh:1:cnt:94a9ed024d3859793618152ea55..."
print(stub.GetNode(swhgraph.GetNodeRequest(
    swhid=swhid)))
```

Result

```
swhid: "swh:1:cnt:94a9ed024d3859793618152ea5..."
cnt {
  length: 35147, is_skipped: false
}
num_successors: 0
```

gRPC API Example: Getting a revision node

CLI

```
$ grpc_cli call localhost:50091 \
 swh.graph.TraversalService.GetNode 'swhid:
  "swh:1:rev:e83c5163316f89bfbde7d9ab23ca2e2..."'
```

Result

```
swhid: "swh:1:rev:e83c5163316f89bfbde7d9ab23..."
rev {
  author: 40606653, committer: 40606653
  author_date: 1112911993, author_date_offset: -420
  committer_date: 1112911993
  committer_date_offset: -420
  message: "Initial revision of \"git\", the [...]"
}
successor { swhid: "swh:1:dir:2b5bfdf7798569..." }
num_successors: 1
```

gRPC API Example: Querying root commits

CLI

```
$ grpc_cli call localhost:50091 \
    swh.graph.TraversalService.Traverse \
    'src: ["swh:1:rev:3a75f674017fdf86d8b74..."],
     edges: "rev:rev",
     return_nodes: { max_traversal_successors: 0},
     mask: {paths: ["swhid", "rev.message"]}'
```

Result

(next slide, please)

gRPC API Example: Querying root commits

Result

```
swhid: "swh:1:rev:1db95b00a2d2a001fd91cd860a71..."
rev {
  message: "Add initial version of gitk to the
    CVS repository\n"
}
swhid: "swh:1:rev:2744b2344dc42fa2a1ddf17f4818..."
rev {
  message: "Start of early patch applicator ..."
}
swhid: "swh:1:rev:e83c5163316f89bfbde7d9ab23ca..."
rev {
  message: "Initial revision of \"git\", the
    information manager from hell\n"
}
```

gRPC API: And more…

Traversal

- from multiple nodes
- in either direction
- depth limit
- node/edge limit

Path-finding

- from a set of nodes to any node of a given type, or
- from a set of nodes to another set of node
- in either direction
- shortest path
- example: can find the a directory containing two other directories

Outline

The Rust API

Host it yourself

- Data available at s3://softwareheritage/graph/2024-05-16/compressed
- Requires between 1.5 and 12TB of disk and at least 500GB of RAM for the full dataset

Documentation

- Reference: https://docs.rs/swh-graph/
- Tutorial: https://docs.rs/swh-graph/latest/swh_graph/_tutorial/
- Crash course for users of the legacy Java API: https://docs.rs/swh-graph/latest/swh_graph/_crash_course/

Rust API: Quickstart

Loading the graph

```
use swh_graph::{graph::*, mph::DynMphf};
let basename = PathBuf::from("./example/graph");
let graph = load_full::<DynMphf>(basename)
    .expect("Could not load graph");
```

Example: statistics

```
let average_outdegree = graph.num_arcs() as f64
    / graph.num_nodes() as f64;

let mut distribution = HashMap::<u64, u64>::new();
for node in 0..graph.num_nodes() {
    *distribution.entry(
        graph.outdegree(node)
    ).or_insert(0) += 1;
}
```

Rust API: Operations

SWHID <-> NodeId

```
let props = graph.properties();

let node: NodeId = props.node_id("swh:1:rev:...")
  .expect("Unknown SWHID");

let swhid: SWHID = props.swhid(node);
```

Neighbors

```
for succ in graph.successors(node) {
    println!("{} -> {}", swhid, props.swhid(succ));
}
```

BFS example: https://gitlab.softwareheritage.org/swh/devel/swh-graph/-/blob/master/rust/examples/bfs.rs

Neighbors with edge labels

```
for (succ, labels) in graph.labeled_successors(node) {
    println!("{} -> {}", swhid, props.swhid(succ));
    for label in labels { match label {
        EdgeLabel::Visit(label) => {
            println!("visit timestamp={}", label.timestamp());
            println!("visit status={:?}", label.status());
        }
        EdgeLabel::Branch(label) => {
            let name = props.label_name(label.filename_id());
            println!("name={}", String::from_utf8_lossy(&name));
        }
        EdgeLabel::DirEntry(label) => {
            let name = props.label_name(label.filename_id());
            println!("name={}", String::from_utf8_lossy(&name));
            println!("perm={:?}", label.permission().unwrap());
    }}}
}
```

BFS example: https://gitlab.softwareheritage.org/swh/devel/swh-graph/-/blob/master/rust/examples/bfs_labels.rs

Rust API: Operations

Revision/release properties

- `props.author_id(node)`, `props.author_timestamp(node)`, `props.author_timestamp_offset(node)`
- `props.committer_id(node)`, `props.committer_timestamp(node)`, `props.committer_timestamp_offset(node)`
- `props.message(node)`, `props.tag_name(node)`

Content properties

- `props.content_length(node)`, `props.is_skipped_content(node)`

Origin properties

- `props.message(node)` (soon renamed to `props.url(node)`)

Rust API: Helpers

Collections

- `AdaptiveNodeSet`: optimized `HashSet<NodeId>`
- `PathStack`: optimized `Vec<Vec<FilenameId> >`

Algorithms

- `iter_nodes`

Accessors

- `find_head_rev`, `find_head_rev_by_refs`, `find_root_dir`
- `fs_resolve_name`, `fs_resolve_path`
- ...

Outline

The End

Documentation

- https://docs.softwareheritage.org/devel/swh-graph/
- https://docs.rs/swh-graph/

Data access

- https://docs.softwareheritage.org/devel/swh-dataset/graph/dataset.html