Mining GitHub for fun and profit

Georgios Gousios // @gousiosg
TU Delft

- Project forge (Git, Issues)
- Project wiki (Git based)
- Social network (star repository, follow developer)
- Forks and Pull requests
- Comments on commits, issues and pull requests

GitHub is cool because

- api.github.com
- process and product data
- interconnection

Mining the GitHub API:

- Only current state
- Missing data schema
- 5k reqs/hr

GHTorrent: a scalable, queriable, offline mirror of GitHub data

api.github.com

Entities

- static view
- interlinked
- current state

Events

- dynamic view
- generated by user actions
- affect current entity state
- can be browsing roots
Relational database for querying

MongoDB as query-able cache
**Operation modes**

- **Normal operation**: Follow the event timeline, apply dependency-based retrieval
- **Bi-monthly updates**: Refresh the state of all repos/users (cater for deleted repos, changed user locations etc)
- **Full retrievals**: Get all info for a repo/user, in case stuff is missing

**Periodic dumps of DBs**

- MySQL database dumps
  - We are distributing CSV files (one per table) instead of .sql files, with a directory including a restore script and instructions on how to do the restore. See more information here.
  - You can also query MySQL. It is always backed with the latest dump.
  - Table: `tabela`
    - `id` (int)
    - `name` (varchar)
    - `country` (varchar)
    - `language` (varchar)

- MongoDB database dumps
  - All from 2015, buyer collection, from August 2015 up to new 2015
  - Daily from 2015-12-31:
    - `repo`
    - `user`
    - `lang`
  - If you need more data, you can use `GET repos` to get the full data.

**User geolocation**

<table>
<thead>
<tr>
<th>country_code</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
<td>283217</td>
</tr>
<tr>
<td>gb</td>
<td>57500</td>
</tr>
<tr>
<td>in</td>
<td>46204</td>
</tr>
<tr>
<td>fr</td>
<td>33003</td>
</tr>
<tr>
<td>nl</td>
<td>24441</td>
</tr>
</tbody>
</table>
### Latest commits

```
select c.*
from commits c, project_commits pc, projects p, users u
where u.login = 'rails'
and p.id = pc.project_id
and c.id = pc.commit_id
order by c.created_at desc
limit 20
```

<table>
<thead>
<tr>
<th>id</th>
<th>author</th>
<th>committer</th>
<th>project_id</th>
<th>repo_id</th>
<th>commit_id</th>
<th>created_at</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Ruby on Rails

```
select year(w.created_at) as year, count(*) as num_starts
from watchers w
where w.repo_id = 1334
and p.owner_id = u1.login
and pc.project_id = p.id
and p.name = 'rails'
order by w.created_at desc
```

<table>
<thead>
<tr>
<th>year</th>
<th>num_starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>3779</td>
</tr>
<tr>
<td>2009</td>
<td>3125</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Django

```
select year(w.created_at) as year, count(*) as num_starts
from watchers w
where w.repo_id = 1126
and p.owner_id = u1.login
and pc.project_id = p.id
and p.name = 'rails'
order by w.created_at desc
```

<table>
<thead>
<tr>
<th>year</th>
<th>num_starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>9844</td>
</tr>
<tr>
<td>2013</td>
<td>1343</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Core team members last 3 months

```
select distinct(u.login) as login
from commits c, users u, project_commits pc, users u1, projects p
where u.id = pc.commitor_id
and u.fave is false
and pc.project_id = p.id
and p.owner_id = u1.id
and p.name = 'rails'
and c.created_at = (SELECT MAX(w.created_at) FROM watchers w
where w.repo_id = pc.project_id
and w.created_at < DATE_SUB(NOW(), INTERVAL 3 MONTH)
GROUP BY w.repo_id)
union
select distinct(u.login) as login
from pull_requests pr, projects p, users u, users u1, pull_request_history prh
where prh.id = prh.request_id
and prh.source = 'merged'
and u1.id = prh.requestor_id
and prh.action = 'merged'
and prh.action = 'merged'
and prh.pull_request_id = pr.id
and prh_base_repo_id = p.id
and prh.created_at < DATE_SUB(NOW(), INTERVAL 3 MONTH)
and prh.action = 'merged'
and u1.login = 'rails'
```

<table>
<thead>
<tr>
<th>login</th>
<th>action</th>
<th>merged</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Python developers in Estonia

```
select distinct(u.login), u.location
from users u, commits c,
projects p, project_commits pc
where date(c.created_at) between date('2016-06-01')
and date('2016-07-01')
and pc.commit_id = c.id
and p.id = pc.project_id
and c.author_id = u.id
and u.country_code = 'ee'
and p.language = 'Python';
```

<table>
<thead>
<tr>
<th>login</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>plaes</td>
<td>Estonia</td>
</tr>
<tr>
<td>sim642</td>
<td>Estonia</td>
</tr>
<tr>
<td>rivol</td>
<td>Tallinn, Estonia</td>
</tr>
<tr>
<td>livenson</td>
<td>Estonia</td>
</tr>
<tr>
<td>mittya</td>
<td>Moon</td>
</tr>
<tr>
<td>ribozz</td>
<td>Tallinn</td>
</tr>
<tr>
<td>emddom</td>
<td>Tallinn</td>
</tr>
<tr>
<td>argoroots</td>
<td>Estonia</td>
</tr>
<tr>
<td>NewbiZ</td>
<td>Tallinn, Estonia</td>
</tr>
<tr>
<td>mitselek</td>
<td>Tallinn, Estonia</td>
</tr>
<tr>
<td>asolntsev</td>
<td>Tallinn, Estonia</td>
</tr>
<tr>
<td>Jyrno42</td>
<td>Estonia, Tallinn</td>
</tr>
<tr>
<td>hudolejev</td>
<td>Estonia</td>
</tr>
<tr>
<td>Villane</td>
<td>Estonia</td>
</tr>
<tr>
<td>syko</td>
<td>Estonia</td>
</tr>
<tr>
<td>raidoz</td>
<td>Estonia</td>
</tr>
<tr>
<td>vruusmann</td>
<td>Estonia</td>
</tr>
<tr>
<td>tamPerl</td>
<td>Estonia</td>
</tr>
<tr>
<td>tlsaffre</td>
<td>Estonia</td>
</tr>
<tr>
<td>mikk351</td>
<td>Tallinn</td>
</tr>
<tr>
<td>Sunnepah</td>
<td>Estonia</td>
</tr>
<tr>
<td>ropod7</td>
<td>Estonia</td>
</tr>
<tr>
<td>siims</td>
<td>Estonia</td>
</tr>
<tr>
<td>jaakerisalu</td>
<td>Tallinn, Estonia</td>
</tr>
<tr>
<td>kendas</td>
<td>Estonia</td>
</tr>
<tr>
<td>artizirk</td>
<td>Estonia</td>
</tr>
<tr>
<td>avalikarvamus</td>
<td>Tallinn, Estonia</td>
</tr>
</tbody>
</table>

27 rows in set (0.00 sec)

Query MongoDB programmatically

Demo!

Streaming updates from GHTorrent

Declarating queue

Real time analytics

Openness reports

How open is your project to community contributions?
- 5k projects
- every 15 days

Roll your own dataset

$ gem install sqlite3 bundler
$ git clone https://github.com/gousiosg/github-mirror
$ cd github-mirror
$ bundle install
$ my config.yaml, standalone > config.yaml
$ ruby -Ilib bin/ght-retrieve-repo -t token rails rails
Even simpler with Vagrant

$ apt-get install vagrant
$ git clone https://github.com/ghtorrent/ghtorrent-vagrant.git
$ cd ghtorrent-vagrant
$ vagrant ssh
$ ruby -Ilib bin/ght-retrieve-repo -t token rails

G. Gousios, The Evolution of GHTorrent: Growing an Open Access Dataset

Stats and Impact
Since Feb 2012
12TB in MongoDB
4.5B rows in MySQL
2GB per hour
170k API req/hour
50 user donated API keys

Impact
MSR 2016: 35% of all papers on Github are done with GHTorrent

Valerio Cosentino, Javier Luis, and Jordi Cabot. 2016. Findings from GitHub: methods, datasets and limitations. MSR '16, pp 137-141

G. Gousios and D. Spinellis, “GHTorrent: GitHub’s Data from a Firehose,” in MSR, 2012, 12-21


180+ users, 100+ institutions
100+ papers
3 data mining challenges
3 best paper awards
50 user donated API keys

Growth

<table>
<thead>
<tr>
<th></th>
<th>MongoDB</th>
<th>MySQL</th>
<th>Diff 2016/2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>476</td>
<td>603</td>
<td>11.1x</td>
</tr>
<tr>
<td>Users</td>
<td>6.7</td>
<td>9.2</td>
<td>8.4x</td>
</tr>
<tr>
<td>Repos</td>
<td>28</td>
<td>25.5</td>
<td>21.8x</td>
</tr>
<tr>
<td>Commits</td>
<td>367</td>
<td>362</td>
<td>12.3x</td>
</tr>
<tr>
<td>Issues</td>
<td>24.1</td>
<td>25.3</td>
<td>10.3x</td>
</tr>
<tr>
<td>Pull requests</td>
<td>11.9</td>
<td>11.1</td>
<td>9.7x</td>
</tr>
<tr>
<td>Issue comments</td>
<td>42</td>
<td>43</td>
<td>14.6x</td>
</tr>
<tr>
<td>Watchers</td>
<td>51</td>
<td>37</td>
<td>6.6x</td>
</tr>
</tbody>
</table>

G. Gousios, The Evolution of GHTorrent: Growing an Open Access Dataset

Diff 2016/2013: 10.3x
MSR 2014 challenge

- Pull requests
- Project developer attraction
- Sentiment analysis of commit comments
- Biodiversity of project ecosystems
- Project openness
- Design discussions
- Co-evolution of documentation and popularity

GitHub API rate limit and network latency. The mining program was able to check about one repository ID per second, which was slowed by network latencies, or, once 5000 API calls had been made in one hour, was throttled by GitHub. The apparent solution was to create multiple accounts, each providing 5000 API calls per hour. After contacting GitHub to request help with this issue, they indicated that they do not want users to create multiple accounts for mining projects because it can put a strain on their servers, slowing the service for regular users. An alternative strategy was proposed by GitHub of using ghlmert to find Python projects without using the API. However, at this point in the project, enough data had been acquired to begin analysis and a determination was made to stop development of this mining program and focus on analysis. Future mining efforts are encouraged to obtain repository information from this database instead of crawling through all projects using the GitHub API, like tour.de.source did.


NEW! MSR 2017 DATA CHALLENGE!

From: Gianugo Rabellino Gianugo.Rabellino@microsoft.com
Sent: 9 Sep 2015 18:49
To: Georgios Gousios, g.gousios@cs.ru.nl
Subject: Streaming software analytics

Dear Mr. Gousios,

This is a note to restate my interest and strong support for your research proposal on streaming software. At Microsoft we have been using your GHTorrent project and we can’t thank you enough for the business critical insights your data have and will be able to provide.

Thanks and best of luck with your efforts,

Gianugo Rabellino
Sr. Director, Open Source Programs Office
Microsoft Corporation

Data Lake
Batch, real-time, and interactive analytics made easy.

- Data lake with flexible and extensible schema.
- All data, both structured and unstructured.
- Supports analytics and query using SQL.
- Data lake with built-in data quality and governance.
- Data lake with built-in business intelligence.

Try it now.
Here are my changes
Please fix those issues
Here are my updates
Looks great, thanks!

contributor

integrator

changes examined
changes re-examined
changes re-examined
changes integrated

Large scale collaboration

projects with > 1 integrators
55% shared repository
45% use pull requests

2016
135k repos/month
600k pull-requests/month

Repository

<table>
<thead>
<tr>
<th>Repository</th>
<th>Commits</th>
<th>Pull requests</th>
<th>Code reviews</th>
<th>Issues</th>
<th>Issue comments</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaseweb</td>
<td>100</td>
<td>157</td>
<td>291</td>
<td>2550</td>
<td>3363</td>
<td>6423</td>
</tr>
<tr>
<td>tomkeek</td>
<td>5988</td>
<td>57</td>
<td>178</td>
<td>5</td>
<td>24</td>
<td>6213</td>
</tr>
<tr>
<td>ruby-en/newsy</td>
<td>1021</td>
<td>121</td>
<td>1752</td>
<td>1844</td>
<td>2166</td>
<td>4036</td>
</tr>
<tr>
<td>jekyll/jekyll</td>
<td>212</td>
<td>431</td>
<td>384</td>
<td>2800</td>
<td>3008</td>
<td>5823</td>
</tr>
<tr>
<td>jh/minio</td>
<td>657</td>
<td>830</td>
<td>1127</td>
<td>2394</td>
<td>2803</td>
<td>7796</td>
</tr>
<tr>
<td>Cos茅SProject</td>
<td>2658</td>
<td>2534</td>
<td>1384</td>
<td>515</td>
<td>268</td>
<td>7389</td>
</tr>
<tr>
<td>global/globa</td>
<td>605</td>
<td>871</td>
<td>1142</td>
<td>454</td>
<td>1283</td>
<td>1737</td>
</tr>
<tr>
<td>angular/angular</td>
<td>875</td>
<td>536</td>
<td>1701</td>
<td>2669</td>
<td>2215</td>
<td>4890</td>
</tr>
<tr>
<td>nvm/nvm</td>
<td>2685</td>
<td>3135</td>
<td>4462</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

23,501

http://www.gousios.gr/blog/The-triumph-of-online-collaboration/
Too successful?

"Lack of knowledge of git from contributors: most don't know how to resolve a merge conflict."

"Sifting through the GitHub information flood to find what, if any, I should address."

"Dealing with loud and trigger-happy developers."

How does the PR process look like?

Across 5k repos/ ~1M pull requests
- 85% merged, 70% with merge button
- 80% < 150 lines, < 7 files, 3 commits
- 66% < 1 day to merge
- 80% 4 comments, 3 participants

Mostly rejected due to observability/awareness issues (not technical!)

Which factors affect PR acceptance?

- Do we know the submitter?
- Can we handle the workload?
- What does the PR look like?
- How ready is our project for PRs?

Which factors affect the time to process PRs?

- Can we handle the workload?
- Do we know the submitter?
- How ready is our project for PRs?
- What does the PR look like?
What do integrators actually believe?

Survey of 650 integrators

Generally, few complaints about the process
Points of pain are mostly social (workload, drive-by PRs, explaining rejection)

Needed tools
- Quality analysis
- Impact analysis
- Work prioritization

What do contributors actually believe?

Survey of 640 contributors.
Similar issues, reversed

Awareness
Asynchrony
Responsiveness

It takes 2 to tango

- Quality
- Lack of process
- Workload and responsiveness
- Communication

What are the integrators’ biggest challenges?

- Quality
- Prioritization

Characteristics of PRs

- They change fast (66% are processed within a day)
- Attention is only needed after an event occurred
- They are integrated with other tools
Which are IMPORTANT?

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMPORTANT = about to be active

Age
Discussion Comments
Additions
Track record
CR Comments
Has test code?
Contains fix?
Intra-branch?
Files

Active in next time window

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Precision</th>
<th>Recall</th>
<th>AUC</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forests</td>
<td>0.66</td>
<td>0.63</td>
<td>0.89</td>
<td>0.86</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>0.34</td>
<td>0.79</td>
<td>0.75</td>
<td>0.60</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>0.36</td>
<td>0.84</td>
<td>0.81</td>
<td>0.62</td>
</tr>
</tbody>
</table>

It is easy to get an overview of the state of the pull requests of the project. It is easy to find what pull request to work on next. The filter lacks support for some important fields. Using the prioritizer service causes too much overhead. Some pull requests show incorrect information.

**Prioritizer user evaluation**

I would use prioritizer for my project. I would recommend it to others. I like the prioritizer web interface more than GitHub’s. The performance of prioritizer is adequate. The time delay between GitHub and the service is acceptable.

**Pourquoi**

Based upon...
Promises & Perils

How good is GitHub data for SE research?
- 5 promises
- 13 perils
- Ways to select projects for research
- Examined MSR 2014 papers, all susceptible to perils!

Issue lifetime predictions

Work dual to PRs
Need to train in short term time windows
Difficult to generalise across projects
Contextual features more important than generic ones

Reviewer recommendation

Exploit @mention networks to propose top-3 reviewers for incoming pull requests.
Accuracy ~60% on top-3 recommendation

Automated code review

Examine how "natural" the PR code is WRT the project's code base.
Accepted PRs are significantly similar to the project
More debated PRs are significantly less similar

Not only OSS

“...GitHub's transparency and popular workflow can promote open collaboration, allowing organizations to increase code reuse and promote knowledge sharing across their teams.”

Gender and Tenure diversity

“Our study suggests that, overall, when forming or recruiting a software team, increased gender and tenure diversity are associated with greater productivity.”
**Geographical diversity**

Traces of bias on contributions from certain countries
- Contributors perceive it
- Integrator’s do not

---

**Gender & Contributions**

When gender is identifiable: women rejected more often
When gender is *not* identifiable: women accepted more often

---

**Productivity**

Study how context-switching between repos affects productivity. If
- rate of switching
- number of projects involved in
are high, productivity goes down

---

**The #issue32 incident**

I am not a lawyer!
- Other commenters are no lawyers either
- The law is complicated and open to interpretation
Two important issues

- **Copyright**: Who owns the data?
- **Privacy**: How does GHTorrent protect users from personal data misuse?

---

Copyright — General terms

- For original content, the publisher maintains full copyright by default
- Licenses restrict the effect of copyright
- Events (e.g. the fact that an issue comment was created) are not copyrightable, but their content may be

---

Copyright — GitHub’s POV

- GitHub: *We claim no intellectual property rights over the material you provide to the Service.* (TOS F.1)
- Structure of API responses is GitHub’s IP
- Several fields in API responses may contain copyrighted material

---

Privacy provisions — EU

- **Privacy** is the ability of an individual or group to seclude themselves, or information about themselves, and thereby express themselves selectively.
- **Personal data** identify a person uniquely
- **Facts** are not personal data
- GHTorrent processes personal data, therefore is a controller
- Controllers must
  - get consent for processing (except in the case of legitimate interest)
  - include mechanisms for opting out

---

Copyright situation example

```json
{
  "id": "4141500869",
  "type": "IssueCommentEvent",
  "actor": {},
  "repo": {},
  "payload": {
    "action": "created",
    "issue": {
      "id": 158442053,
      "number": 138,
      "title": "Issue in CopyrightedProjectName",
      "user": {},
      "labels": [],
      "state": "closed",
      "body": "Added data holding classes and a map manager. Will add a system soon",
      "comments": [
        {
          "created": "2016-06-14T05:51:16Z",
          "updated": "2016-06-14T05:51:16Z",
          "body": "continuing in #141"
        }
      ]
    },
    "comment": {
      "created": "2016-06-14T05:51:16Z",
      "updated": "2016-06-14T05:51:16Z",
      "body": "continuing in #141"
    }
  }
}
```
Privacy provisions — USA

- No single law/directive
- Consent only required for specific types of data storage (e.g., social security numbers)
- Offering an opting out mechanism

What did GHTorrent do?

- Stopped distributing user names and emails in MySQL data dumps
- Researchers can “sign” a form to get access to private data
- Created an opt-out process
- In the process of creating Terms of Fair Use

Research ethics

Can we, in the name of science,

- send emails to developers?
- create developer profiles?
- recommend work to developers?
- rank developers based on contributions?
- compare project characteristics?
- characterise community practices?

Where is GHTorrent going?

Challenges

- 3Vs of big data: Volume, Velocity and Variety
- Hardware not enough
- Software not optimal
- Services not always up and running
What can we do?

- Proper storage subsystem
- MongoDB sharding? HDFS? Hyperdex?
- Proper querying infrastructure
  - Spark?
  - DataLake? ← Working with Microsoft
- Distributed operation and eventual consistency?

DataLake: sneak preview

A dataset accessible to anybody who wants to do research on extremely interesting big data

The future: Feedback Driven SE

Feedback-driven development

Data silos
No complex frameworks!

No semantic correctness!

Data streaming

Queries on streams

Observable.flatMap{s => findDev(s) match {
  case Some(s) => Observable.just(s)
  case None => Observable.error(new Exception("Not found"))
}}
.flatMap{case (dev, xs) =>
  xs.map(c => (dev, c))
  .groupBy(x => x)
  .map{case (dev, xs) =>
    (dev, xs.scan(0)((count, _) => count + 1))
  }
}

How does my team test?

Did switching to agile work?
How is my incident response?

Processes:
- Deployment
- Runtime
- System
- Project
- Team
- Developer

Products:
- Ecosystem
- System
- Program
- Logical Entity
- File
- Statement

Software is eating the world

Developers as analysts

I WANT YOU TO HELP GHTorrent

@gousiosg
@ghtorrent