Expert Developer Recommendation Using Very Large Datasets

Tilemachos Pechlivanoglou, Saim Mehmood
Electrical Engineering and Computer Science
York University
saim@eecs.yorku.ca
tipech@eecs.yorku.ca

Abstract - Global and distributed software development increases the need to find developers with relevant expertise. At the same time, online collaborative development tools such as GitHub have become extremely popular and are publicly providing considerably large datasets on projects and developers. Currently, a variety of algorithms, methodologies and techniques exists on using collaborative data to recommend expert developers based on knowledge and other criteria. We worked to adapt as well as extend several of these methods so as to better match the large size of available datasets. Specifically, we created a tool that can be used as a search engine for expert developers, based on their skills and experience, which are extracted based on their actual code contributions.

Index Terms - expert recommendation, big data, code analysis.

I. Introduction

Companies and academia often need to hire expert developers with very specific backgrounds and programming expertise. However, traditional methods for these institutions to locate and obtain these experts have many shortcomings. Job postings and research of individual developers can be time-consuming, while the actual development experience and fulfillment of requirements may be hard to estimate beforehand - employers and academics need to rely heavily on interviews or test projects to judge the strengths and skills of the potential expert [1].

To counteract this issue, some methods were proposed and tools developed to work as search engines for developers [2,3]. These utilities have access to development and/or collaboration data and use them to determine the expertise of programmers on various fields such as programming languages and frameworks. We go into more depth about these tools in the "Related Work" section below.

A common factor in past works, however, is the size of the data used in the analysis. Due to the lack of past data and processing requirements, most projects rely on medium-sized code and developer datasets. In recent years, the code collaboration service GitHub has increased significantly in popularity and has aggregated a very large number of projects, most of them public and open-source. For a large number of these open projects, they have made available all development data. By leveraging this fact, along with advances in Big Data processing, we implemented existing expert developer recommendation algorithms, as long as adapted and extended them for use in these very large scales.

The goal was the creation of a tool that can function as a search engine for developers based on a number of search parameters and criteria. The tool should be capable of answering queries such as "*I need a developer who has 5 years of experience in C++ development and who has contributed to an Apache project*", or "*I need a developer who has recently worked on Python, is familiar with the Flask framework and his contributions are frequent*". Furthermore, the system should provide the user with options for ranking the resulting recommendations for display, as long as allowing for partial or "close-enough" matches to be returned, if there aren't sufficient results.

The background dataset where the search is applied was constructed by processing and condensing actual code commits and contributions to the GitHub platform of more than 20 million developers over a period of 12 years. Secondary goals for the product were making sure query reply times were reasonably small and the tool could support extensions, such as more search or sorting criteria, if a more complete background dataset is available in the future.

In Section II we go over previous related work in the field and in Section III we outline the steps taken to process the large-scale data and condense it. Moreover, in Section IV we cover the tools and methodologies used to construct the search engine and finally we provide some conclusions and thoughts for future work in Section V.

TABLE 1
GITHUB DATASET PROPERTIES

Property	Size
number of repositories	2.9 million
number of commits	215 million
number of file paths	2.3 billion
number of file contents	163 million
size in disk	3+ TB

II. RELATED WORK

Researchers in the area of recommender systems have investigated how to recommend experts based on a mined history of development. Some approaches use both *implementation expertise* (i.e. through the act of changing a file, the developer is considered to have gained expertise for the said file) [4] and *usage expertise* (i.e. simply by calling a method, a developer demonstrates that they at the very least know what the method does, without knowing implementation details) [5,6]. The underlying assumption in such systems is that the commited change set is evidence that the commit author had the expertise to do so.

Teyton *et al.* tries to automate the identification of experts within Open Source Software communities. Their idea is that a developer is an expert of a library if he/she introduces source code that uses this library [7]. Their contribution comes with a tool, named LIBTIC, that is intended to be used by project leaders to evaluate library knowledge among developers and find experts of various libraries.

In one other study D Surian *et al.* proposed an approach using random walk with restart procedure to recommend people from developer's collaboration network extracted from a super-repository of Source-Forge.net containing numerous developers, projects, and project properties [8]. They essentially extract a Developer-Project-Property (*DPP*) graph to recommend developers based on compatibility metrics.

Our contribution mainly relies on techniques involving implementation expertise, as the more in-depth analysis methods and data structures required during usage expertise analysis on large scale data are too costly in terms of processing time. Furthermore, we extracted additional statistical information for every developer such as frequency of commits or size of commit messages, a notion that hasn't been explored in previous work.

III. DATA REDUCTION

A. Dataset

The source data for this project is the dataset provided by GitHub itself, readily available in the Google BigQuery project resources. This is maintained and regularly updated by the GitHub organization itself on a regular basis, and it contains contributions from more than 20 million developers. More details about the size of the dataset can be seen in Table 1.

The particularly large size of the dataset has several benefits. The data is characterized by greater variety, which means that the results can be more refined, as well as cover most potential search queries (rarely used languages etc.). Furthermore, since the pool of potential experts is much larger, the chances of many potential matching experts are greater.

On the other hand, dealing with data in such a large scale can be problematic. Arrangements had to be made to make sure search query completion time isn't prohibitively high, as well as keeping memory requirements within reason. Moreover, it is almost prohibitive to perform more in-depth analysis of the actual code, such as mining commit messages and code to detect developers who tend to introduce or fix a large number of bugs, etc.

The data is formatted in 4 different tables:

- the table "commits" holds an entry for every code contribution, along with any commit messages, information on the repository the contribution is to and the username of the contributing developer
- the table "contents" holds the filenames and paths of all the files altered; unfortunately, although the table is also supposed to contain the actual code changes of the commit, that information isn't present in the table
- the table "files" contains a filename and path entry for every file in every GitHub repository; similarly to the previous table, although a file contents column exists, it is empty
- the table "languages" contains general statistical information on the programming language content of the 400.000 most watched repositories

There are also 4 additional tables, identical in structure to the first ones but 500 times smaller, that contain a much smaller sample of the dataset. During the initial data reduction, this sample dataset was used when frequent testing of large-scale processing was necessary. Finally, there is another table containing licence information for all repositories, which wasn't used in this project.

B. Tools and Goals for Large-Scale Data Reduction

For this first processing step, we used Google's BigQuery large-scale data processing tool. BigQuery is a web service tool that enables interactive analysis of massively large datasets working in conjunction with Google Storage and the Google Cloud processing platform. It allows for the application of traditional or extended and specialised SQL queries in an efficient and distributed way.

Our aim was to condense and compact the contribution information in the dataset to a single table, which would hold an entry for every single developer. This way, all the past experience and skill-set information of each developer would be readily available and searchable in the same source. Of course, the notion of development experience can be ambiguous and relative - if a developer only wrote ReadMe or configuration files for a web-app project involving HTML/CSS, PHP and Javascript, are they experienced in one, all or none of these languages and tools?

Therefore, to specify our past experience and skills criteria under the previously mentioned *implementation expertise* standards, we considered these factors:

- the programming languages or frameworks a developer has experience in; for example, a person who made many contributions in the Java language can be considered experienced in Java
- the specific projects a developer has contributed to; for example, a person who has made a significant number of contributions to the Node.js framework is considered experienced in it and in Javascript
- the size, time and frequency of contributions of each developer; for example, a person who has altered a single Python file in a repository can't easily be considered an expert in Python

Because the programming languages statistics table available is repository-wide and not specific to each file, commit or contributor, we associated file extensions with their respective language or framework with the use of an extra table based on the way GitHub itself detects programming languages [9].

As code commit data include the username, email and relevant information on actual developers, along with the ids of the files they modified in that commit, it is possible to match developers with file types and therefore languages they have used. Also, timestamp information is available for all those events, which means time-related queries can be formed (e.g. Java developer for 5 years). Any possible extensions to the final product can make use of the actual code content within commits or file contents, if that becomes available in the future.


```
{
  "developer": "Aaron Koskinen",
  "email": "2c839f485b6c7a4b3551406434eb065903220af6@nokia.com",
  "repos":
  [
    "apache/apache-web-server",
    "linux/kernel"
  ],
  "languages":
  [
    {
      "name": "C",
      "first_language_commit": 1522437797,
      "last_language_commit": 1522438797,
      "duration": 94545421,
      "commit_count": 80,
      "files": 100
    },
    {
      "name": "C++",
      "first_language_commit": 1522337797,
      "last_language_commit": 1522478797,
      "duration": 94545521,
      "commit_count": 81,
      "files": 110
    }
  ],
  "average_message_length": 495.7,
  "total_commits": 150,
  "first_commit": 1522437797,
  "last_commit": 1522467797,
  "total_duration": 94545421
}
```


Fig. 1 JSON Representation of a single developer entry

C. Data Reduction Steps

The actual process of reducing the initial large dataset into a single, smaller table was composed of three steps:

Discarding unnecessary data Only information on the actual developer commits was needed, so the rest of the details in the commits, contents and files tables (such as commit and file hashes, encoding information etc.) was discarded. Also, specific entries, such as image files or commits that only consisted of actions such as folder renaming, etc. were also removed. This way, the dataset was reduced in size from the initial 3TB to just 100GB.

Associating file extensions and languages The ids of the files altered in every commit are matched to the respective filename and the file extensions are extracted using regular expressions. Consequently, the extensions are associated with the programming language or framework they correspond to and therefore each individual commit is now linked with both a developer and a set of skills. The processed file data is no longer needed, as only the expertise information is required further. This results in a size reduction from 100GB to 51GB.

Condensing time data Instead of holding a record for every individual commit, we use statistical methods to combine the various entries to a few meaningful metrics. Specifically, during this aggregation we store these metrics, separately for each skill, for each repository and overall:

a. the date of first and last commit
b. the number of commits and files altered
c. the average time between commits
d. the average length of commit messages

Using these metrics we can construct individual profiles for each developer and associate them with specific and quantifiable skills and experience. The final format of a single developer entry can be seen in Fig. 1. This results in a significant reduction of data size, from 51GB down to just 1.1GB.

Small optimizations We also performed a number of small actions that reduced the overall size of the data. The most significant ones were:

- reducing forked repository names (e.g. torvalds/linux → linux)
- changing the structure from single entry to nested (CSV to JSON representations)

With these improvements the size of the final dataset was brought down to 600MB, just 0.02% of the source. Although not trivial, this reduced size means that quick and complete searches are now possible.

Finally, the resulting data table was downloaded and stored in JSON format, which was later loaded to the search engine database. Any intermediate operations, such as indexing or date reformatting, were performed either with the use of a text editor or small scripts run locally.

IV. Expert Search

A. Tools and Goals for Search

Now that we have a table of manageable size with individual developer profile entries, our next goal is to make it searchable. To achieve this, 3 components are required:

- a rapid-access database to store the data
- a fast search engine, able to handle both text queries (e.g. contributed to "Apache" projects) and logical comparisons (e.g. between 2 and 5 years)
- a user interface to submit queries, display results

For both the database and search engine functionalities, the tool Elasticsearch by Elastic was used, as it can successfully perform both functions. Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is used as the underlying engine/technology that powers applications with complex search features. It can be used to search all kinds of documents and supports multi tenancy (refers to a software architecture in which a single instance of software runs on a server and serves multiple tenants). Furthermore, it is distributed, which means that indices can be divided into shards (a horizontal partition of data in a database) and each shard can have zero or more replicas.

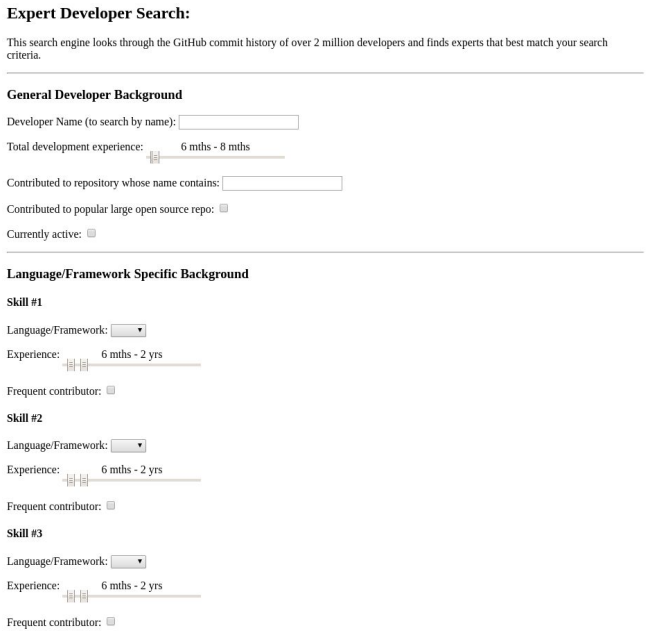

Fig. 2 Search entry form

For the purposes of user interface, we used simple web front-end technologies, mainly HTML/CSS and Javascript to display the search criteria forms and results (as seen in Fig. 2). To communicate between front-end and the Elasticsearch engine, we used Bottle.py, a fast and light Python back-end framework. When the user selects and submits the criteria or terms for the search, the properties entered in the web-browser forms are sent to the Bottle server, which constructs and sends the query to the local Elasticsearch instance. After waiting for the results, the Bottle back-end sorts them by the ranking criteria the user requested and sends them to the browser for display back to the user.

V. Conclusions

Developing a tool capable of recommending experts with specific programming skills based on their actual contributions isn't a straightforward process. There are two different approaches to the problem: performing in-depth analysis of code and contributions to come up with highly accurate representation of developer experience; or conduct a more shallow analysis on a much larger dataset. Trying to combine the two options can often result in tremendous amounts of processing requirements.

We hope that we have outlined the methods and techniques that can be used to achieve the best of both

worlds. Specifically, we show that by starting from very large datasets and using statistical techniques the data required for searching can be significantly condensed. Moreover, as it is improved and extended, we hope that the tool we developed can be used to assist or streamline the discovery process for experts in the programming field.

There are many short or long-term steps that can be taken to improve or extend the functionalities of the developed tool. For example, it is possible to create and maintain a separate table of repository or project "profiles", combining the extracted information of developer skills. This would allow for search queries on projects and repositories that require specific experience or skill-sets, making the tool useful for developers themselves looking for potential environments to apply their knowledge.

As part of more long-term work on this project, it should be possible to incorporate more in-depth code analysis into the search engine. For example, code quality metrics such as bugs fixed are introduced, and even large-scale features such as measuring the originality or repetition of code by developers could be introduced. Furthermore, other notions from the expert recommendation field, such as recommendation of a group of experts simultaneously with regards to complementing experience and group cohesion, for example, could be explored. In any case, it is our hope that this tool will prove its usefulness and further understanding in the field

VI. References

[1] D Tesch, MG Sobol, G Klein, JJ Jiang, User and developer common knowledge: Effect on the success of information system development projects, International Journal of Project Management, Volume 27, Issue 7, October 2009, Pages 657-664

[2] H Kagdi, M Hammad, JI Maletic, Who can help me with this source code change?, 2008 IEEE International Conference on Software Maintenance

[3] J. G. Bae, J. D. Yang, M. Y. Lee, "Design and Implementation of Expert Recommending System with Extended Object-Based Thesauri on Social Network Services", Information Science and Applications (ICISA) 2011 International Conference.

[4] David Ma, David Schuler, Thomas Zimmerman and Jonathan Sillito, "Expert recommendation with usage expertise," Software Maintenance, 2009. ICSM 2009.J Anvik, GC Murphy, Determining Implementation Expertise from Bug Reports, MSR '07 Proceedings of the Fourth International Workshop on Mining Software Repositories

[5] D Schuler, T Zimmermann, Mining usage expertise from version archives, MSR '08 Proceedings of the 2008 international working conference on Mining software repositories

[6] D Ma, D Schuler, T Zimmermann, J. Sillito, Expert recommendation with usage expertise, 2009 IEEE International Conference on Software Maintenance

[7] C. Teyton, J. Falleri, F. Morandat & X. Blanc, "Find your library experts," IEEE: Reverse Engineering, 2013.

[8] D Surian, N Liu, D Lo, H Tong, EP Lim, C. Faloutsos, "Recommending people in developers' collaboration network", 2011 18th Working Conference on Reverse Engineering

[9] Linguist GitHub project repository [Online]. Available: https://github.com/github/linguist/blob/master/lib/linguist/languages.yml