Mining Source Code Repositories with Boa

Robert Dyer Hoan Anh Nguyen Hridesh Rajan Tien N. Nguyen
Iowa State University
{rdyer,hoan,hridesh,tien}@iastate.edu

Abstract

Mining source code has become a common task for researchers and yielded significant benefits for the software engineering community. Mining source code however is a very difficult and time consuming task. The *Boa* language and infrastructure was designed to ease mining of project and revision metadata. Recently *Boa* was extended to support mining source code and currently contains source code for over 23k Java projects, including full revision histories.

In this demonstration we pose source code mining tasks and give solutions using *Boa*. We then execute these programs via our web-based infrastructure and show how to easily make the results available for future researchers.

Categories and Subject Descriptors D.3.3 [*PROGRAMMING LANGUAGES*]: Language Constructs and Features

Keywords MapReduce; software repository mining

1. Background

Source code repositories such as SourceForge, GitHub, and Google Code contain a vast wealth of information. Researchers are interested in mining this source code to gain insights into problems and test hypotheses. Mining this source code is a difficult task requiring, at a minimum: 1) substantial knowledge about how to access the source code data; 2) knowledge about how to mine the source code data, including parsing; and 3) analyzing a large quantity of data, typically requiring additional complexity and knowledge of how to parallelize the mining task.

Consider a relatively simple example that wishes to answer the question "how many null checks are there in Java programs?" A typical approach to solve this task would write a program that does (at a minimum) the following: downloads/scrapes project metadata from the repository, parses this metadata, determines which projects are Java projects, accesses the source code repository to download the source code, parses the source code, mines the parsed code for null checks, and accumulates the results into a final answer. Such a solution would require using several libraries (e.g. to parse the metadata, access the repository, parse the code). This analysis would also take a significant amount of time as it runs sequentially and accesses thousands of remote repositories. Minimizing this time would require additional complexity and knowledge of distributed programming.

2. Boa

To solve these issues, we present the *Boa* language and supporting infrastructure [3, 4]. *Boa* provides several domain-specific types to ease software mining tasks. These types abstract the details of how to mine repositories and represent the data from the repository. *Boa* also abstracts away the details of its underlying MapReduce framework [2], allowing *Boa* programs to run efficiently in a distributed environment without requiring users to explicitly define parallelism in their code.

```
1  p: Project = input;
2  NullChecks: output sum of int;

3  nullVisitor := visitor {
4    before node: Expression ->
5      if ((node.kind == ExpressionKind.EQ ||
           node.kind == ExpressionKind.NEQ)
6          && (isliteral(node.expressions[0], "null") ||
              isliteral(node.expressions[1], "null")))
7        NullChecks << 1;
8  };

9  ifVisitor := visitor {
10   before node: Statement ->
11     if (node.kind == StatementKind.IF)
12       visit(node.expression, nullVisitor);
13 };

14 exists (i: int; p.programming_languages[i] == "Java")
15   visit(p, ifVisitor);
```

Figure 1. Program in *Boa* answering "How many null checks are there in Java programs?"

An example program is shown in Figure 1 which answers the previous question of how many null checks are there in Java programs. Note how simple this code is - it is only 15 lines of code! There is also no notion of mining the software

SPLASH '13, October 26–31, 2013, Indianapolis, Indiana, USA.

ACM 978-1-4503-1995-9/13/10.
http://dx.doi.org/10.1145/2508075.2514570

repository or parallelizing the code, as these are completely abstracted from the user.

To run such programs, our infrastructure builds on the Sizzle compiler [5], which generates programs that run on the Hadoop MapReduce framework [1]. We add support for our domain-specific types as well as several language features not previously implemented, such as quantifiers. These statements allow easily filtering, e.g. Figure 1 on line 14 for selecting only Java projects.

The language also provides syntax based on the object-oriented visitor pattern to ease source code mining tasks. This allows easily querying for `if` statements (lines 9–13) that contain a comparison to `null` (lines 3–8).

Output is then sent to a table (line 7). The table provides an aggregation function (several are built into the language, such as *sum, mean, min/max*, etc.) to collect the results and reduce them to a final answer.

3. Benefits of Boa

Boa aims to lower the barrier to entry for researchers wishing to perform software mining tasks. It also aims to provide efficient support for performing these tasks on a very large scale. In summary, *Boa* provides the following key benefits:

- Simple programs,
- details of repository mining abstracted away,
- no libraries needed to perform repository mining,
- extremely efficient and scalable - automatically runs in a fraction of the time of standard approaches, and
- queries a very large set of data (project and revision metadata for all projects on SourceForge and all Java source code with full histories).

4. Demonstration Overview

This demonstration gives several simple mining tasks and uses the *Boa* language to solve those tasks. These *Boa* programs are then submitted to the web-based infrastructure [4] (see Figure 2) for execution and the query output downloaded. Additionally, we demonstrate how researchers can use *Boa* to answer their hypotheses and then publish their results to allow easy reproduction by other researchers.

5. Presenter Biographies

Robert Dyer and Hridesh Rajan have prior experience in developing new programming languages. Rajan developed the Ptolemy event-based language as well as the aspect-oriented language Eos. Dyer worked on the implementations and evaluation of the Ptolemy language. They have successfully given previous demonstrations at AOSD'10, FSE'10, ECOOP'11, SPLASH'11, and SPLASH'12.

Hoan Nguyen and Tien Nguyen are experts in software evolution and mining software repositories. Their work includes mining research in clone and API usage evolution, bug prediction and localization, and traceability link recovery. They are also experts in version control systems with work on novel infrastructures for semantics-based version control and configuration management.

All four authors worked on the design of the *Boa* language and infrastructure. Robert Dyer and Hoan Nguyen also developed the supporting infrastructure. Robert Dyer has previously given a demo of an early version of *Boa* at SPLASH'12 and a demo during the ICSE'13 presentation.

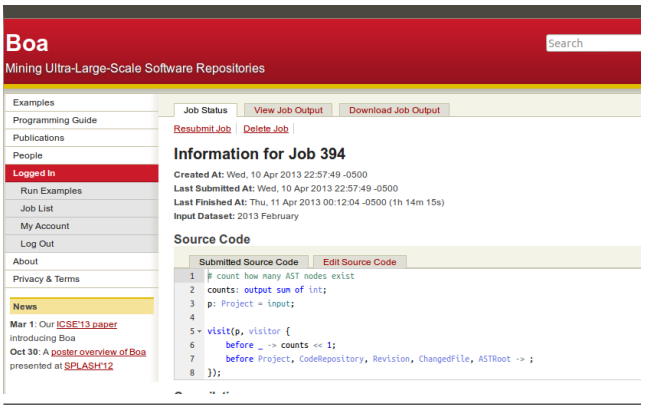

Figure 2. *Boa*'s web-based interface [4] for submitting and executing programs and retrieving their results

Acknowledgments

Dyer and Rajan are funded in part by NSF grants CCF-10-17334, CCF-11-17937, and CCF-08-46059. Tien Nguyen and Hoan Nguyen are funded in part by NSF grants CCF-10-18600 and CNS-12-23828.

References

[1] Apache Software Foundation. Hadoop: Open source implementation of MapReduce. `http://hadoop.apache.org/`.

[2] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, 2004.

[3] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Proceedings of the International Conference on Software Engineering*, ICSE'13, pages 422–431, 2013.

[4] H. Rajan, T. N. Nguyen, R. Dyer, and H. A. Nguyen. Boa website. `http://boa.cs.iastate.edu/`, 2012.

[5] A. Urso. Sizzle: A compiler and runtime for Sawzall, optimized for Hadoop. `https://github.com/anthonyu/Sizzle`.