Demonstrating Programming Language Feature Mining using Boa

Robert Dyer
Bowling Green State University
Bowling Green, OH, USA
rdyer@bgsu.edu

Hridesh Rajan Tien N. Nguyen
Hoan Anh Nguyen
Iowa State University
Ames, IA, USA
{hridesh,tien,hoan}@iastate.edu

Abstract

Programming language researchers often study real-world projects to see how language features have been adopted and are being used. Typically researchers choose a small number of projects to study, due to the immense challenges associated with finding, downloading, storing, processing, and querying large amounts of data. The *Boa* programming language and infrastructure was designed to solve these challenges and allow researchers to focus on simply asking the right questions. *Boa* provides a domain-specific language to abstract details of how to mine hundreds of thousands of projects and also abstracts how to efficiently query that data.

We have previously used this platform to perform a large study of the adoption of Java's language features over time. In this demonstration, we will show you how we used *Boa* to quickly analyze billions of AST nodes and study the adoption of Java's language features.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features

Keywords mining; software repositories; language features

1. Background

Programming language researchers often study real-world software projects to see how language features are being (mis)used. Typically researchers select a handful of "representative" projects to analyze, which can limit the generalizability of the research. Ideally researchers would study thousands (or more) such projects, but this comes at a great cost in terms of effort and time. For example, analyzing all the projects on SourceForge poses several key problems for researchers: 1) researchers would need substantial knowledge about how to acquire and manage such a large amount of data; 2) processing the data to make it amenable to mining requires substantial domain knowledge; and 3) querying such a large amount of data in a reasonable time requires additional knowledge of distributed computing and makes implementations more complex.

Even a relatively simple question such as "how have variable-arity method arguments been used over time?" is extremely difficult to answer for most researchers. Answering such a question requires, at a minimum: finding and downloading a large number of open-source projects, mining the repository data to find source files, parsing the source code, analyzing the source code, and accumulating the results. Solving this task would require knowledge of many libraries (e.g. for accessing the repositories, parsing source code) and would take substantial time unless the researcher also writes the queries as distributed programs, which would add additional time and complexity to solving the task.

```
1  Varargs: output collection[string][time] of int;
2
3  fileName: string;
4  commitDate: time;
5
6  visit(input, visitor {
7    before node: ChangedFile -> fileName = node.name;
8    before node: Revision -> commitDate = node.commit_date;
9    before node: Method ->
10     if (len(node.arguments) > 0) {
11       lastArg := node.arguments[len(node.arguments) - 1];
12       if (strfind("...", lastArg.variable_type.name) > -1)
13         Varargs[input.project_url+fileName][commitDate] << 1;
14     }
15 });
```

Figure 1. Analyzing use of varargs (T... id) over time.

2. Boa

The *Boa* [2, 3, 5] language and infrastructure was designed to solve these problems and provide researchers with an easy

SPLASH Companion'15, October 25–30, 2015, Pittsburgh, PA, USA

http://dx.doi.org/10.1145/2814189.2814192

to use domain-specific language for writing queries that analyze hundreds of thousands of open-source projects. While *Boa* queries look sequential, the server automatically parallelizes the query for efficient execution. Users also do not need to learn libraries for accessing version control systems, parsing source code, etc. since *Boa* has already processed the data and made it available via a small number of domain-specific types.

As an example, consider the *Boa* query shown in Figure 1. This query finds (line 12) uses of a specific language feature available in many languages, variable-arity arguments for methods. It then outputs (line 13) the location of the use and the commit time. This allows tracking how the feature was adopted over time.

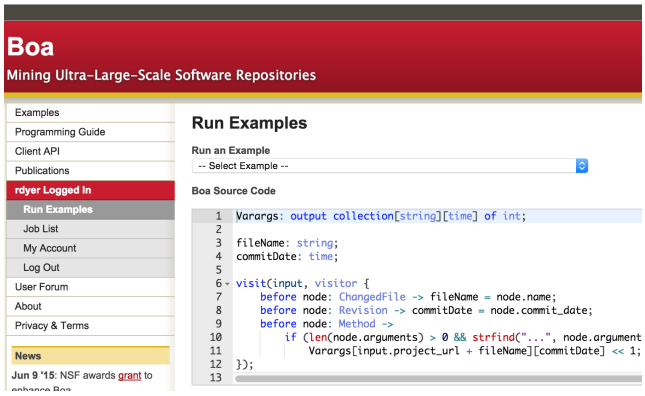

Figure 2. *Boa*'s web-based interface [5] for submitting, executing, and viewing results of queries.

Such a query can be submitted to *Boa*'s website, as shown in Figure 2. The *Boa* servers then automatically transform the query into a distributed Hadoop [1] program and run it on a cluster. When finished, the results are then available via the website or can be downloaded for further processing.

3. Benefits of Boa

Boa provides programming language researchers with a lower barrier to entry for studying language feature use in real-world software. It allows researchers to easily and efficiently mine hundreds of thousands of projects. In summary, *Boa* has the following key benefits for researchers:

- Relatively small and simple queries,
- no libraries needed to access version control or parse source code,
- queries automatically scale and parallelize, running in a fraction of the time of standard sequential repository mining approaches, and
- provides several very large datasets of open-source projects for analysis.

4. Demonstration Overview

This demonstration shows how researchers can utilize *Boa* to mine programming language feature usage in thousands of open-source projects.

- background and introduction to *Boa*,
- background on our previous study [4],
- example queries from the study, including how to construct them and showing how to run them,
- demonstrating what to do with the results of queries, and
- pointers on where to find help with using *Boa*.

At the end of the demonstration, researchers should be capable of utilizing *Boa* in their own research.

5. Presenter Biographies

All of the presenters were involved in designing and implementing the *Boa* project. They have also successfully given previous demonstrations at SPLASH, ECOOP, ICSE, and FSE. Both Robert Dyer and Hridesh Rajan have extensive experience designing and implementing new programming languages. Tien Nguyen and Hoan Nguyen have extensive expertise in mining software repositories and software evolution. They are also experts in version control systems.

Acknowledgments

This work was supported in part by the US National Science Foundation under grants CCF-15-18897, CCF-15-18776, CNS-15-13263, CNS-15-12947, CCF-14-23370, CCF-13-49153, CCF-13-20578, TWC-12-23828, CCF-11-17937, CCF-10-17334, and CCF-10-18600.

References

[1] Apache Software Foundation. Hadoop: open source implementation of MapReduce. `http://hadoop.apache.org/`, 2014.

[2] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: a language and infrastructure for analyzing ultra-large-scale software repositories. In *ICSE'13*, pages 422–431, 2013.

[3] R. Dyer, H. Rajan, and T. N. Nguyen. Declarative visitors to ease fine-grained source code mining with full history on billions of AST nodes. In *GPCE'13*, pages 23–32, 2013.

[4] R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen. Mining billions of AST nodes to study actual and potential usage of Java language features. In *ICSE'14*, pages 779–790, 2014.

[5] H. Rajan, T. N. Nguyen, R. Dyer, and H. A. Nguyen. Boa website. `http://boa.cs.iastate.edu/`, 2015.