Analyzing Ultra-Large-Scale Code Corpus with *Boa*

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

Abstract

Analyzing the wealth of information contained in software repositories requires significant expertise in mining techniques as well as a large infrastructure. In order to make this information more reachable for non-experts, we present the *Boa* language and infrastructure. Using *Boa*, these mining tasks are much simpler to write as the details are abstracted away. *Boa* programs also run on a distributed cluster to automatically provide massive parallelization to users and return results in minutes instead of potentially days.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Distributed programming

General Terms Experimentation, Languages

Keywords MapReduce, software repository mining

1. Background

Repositories such as SourceForge, GitHub, and Google Code contain over 250,000 projects each. Together, these represent an enormous amount of software, as well as related project and bug information, just waiting to be analyzed. Such a task however is much easier said than done. There are at least three problems to mining such repositories: 1) researchers must have significant knowledge about how to access and mine such data, including all of the libraries needed to implement the required infrastructure; 2) analyzing such a large amount of data using traditional methods takes a significant amount of time, thus also potentially requiring knowledge and implementation complexity of parallelizing the analysis; and 3) reproducing another research result is almost impossible, due to the burdens imposed by the first two problems.

Consider a relatively simple example that wishes to answer the question "how many revisions are there for all Java projects using Subversion in SourceForge?" A typical approach to this 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 and use Subversion, accesses the Subversion repository to obtain the revision count, and accumulates the results into a final answer. Such a solution would require using several libraries (to parse the metadata and access Subversion) and contain upwards of 60 lines of code. This analysis would also take a significant amount of time, as it runs sequentially and accesses hundreds of thousands of remote repositories.

2. *Boa*

To solve these issues, we present the *Boa* language and supporting infrastructure. *Boa* is inspired by the Sawzall language [**?**] but adds several domain-specific types to ease software mining tasks. These types represent the mined data from the repository and abstract the details of how to mine that data from the user. *Boa* also abstracts away the details of the MapReduce framework [**?**], which allows the programs to run in a distributed environment without requiring users to explicitly mark parallelism in their code.

An example program is shown in Figure 1 which answers the previous question of how many revisions exist for Java programs using Subversion. Note how simple this code is - it is only 5 lines of code! There is also no notion of mining the software or parallelizing the code, as these are completely abstracted from the user.

```
1 total_revisions : output sum of int;
2 p: Project = input;
3 when (i: some int; match('^java$', p.programming_languages[i]))
4   when (j: each int; p.code_repositories[j].repository_type ==
          RepositoryType.SVN))
5     total_revisions << len(p.code_repositories[j].revisions);
```

Figure 1. Program in *Boa* answering "How many revisions in Java projects using Subversion?"

To run such programs, our infrastructure builds on the Sizzle compiler [**?**], which generates programs that run on the Hadoop MapReduce framework [**?**]. We add support for our domain-specific types as well as several language features not previously implemented, such as quantifiers and

SPLASH'12, October 19–26, 2012, Tucson, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

`when` statements. An example use of these features is shown in Figure 1 on lines 3 and 4. These statements allow easily filtering the data, which is then sent as output to a table (line 5). 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 very large scale repositories.

	LOC		Time	
Task	**Java**	***Boa***	**Java**	***Boa***
Counting revisions	60	4	331m	<1m
Counting committers	69	6	1,596m	<1m
# Multi-lingual projects	32	4	10m	<1m

Figure 2. Three mining tasks implemented in Java and *Boa*.

Some early results are shown in Figure 2, where we implemented three software mining tasks in both Java and *Boa*. These results show significant improvements in lines of code, requiring only 5 lines of code on average for *Boa* programs. They also show considerable speedup in execution time on an input size of 620k+ projects. The *Boa* versions finish in under one minute, whereas the Java versions take up to 27 hours (a speedup of over 1,500x)! Even in the simplest case *Boa* shows speedups of 10x.

In summary, *Boa* provides the following key benefits:

- simple to write programs (usually around 5 lines),
- details of repository mining abstracted away from users,
- no need to learn libraries to perform repository mining,
- extremely efficient - runs in a fraction of the time of standard approaches, and
- scalable to ultra-large code repositories.

4. Demonstration Overview

This demonstration showcases the benefits of the *Boa* language and infrastructure via several realistic examples. Software engineering questions are proposed and answered using *Boa* programs. These example programs are then run on the *Boa* web-based infrastructure (see Figure 3), demonstrating how researchers can successfully answer research hypotheses using *Boa*.

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

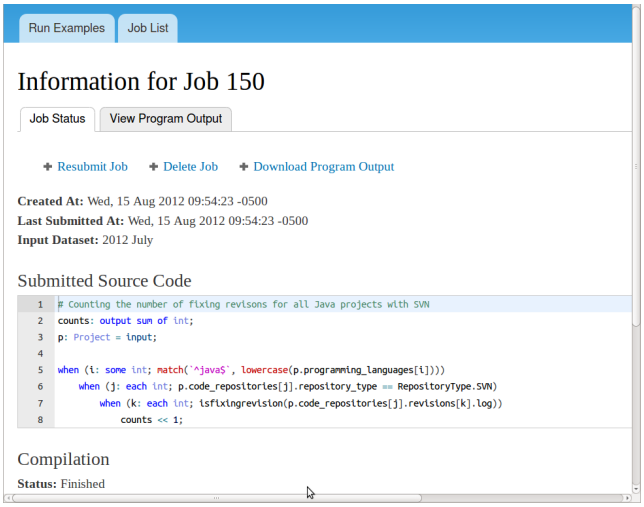

Figure 3. *Boa*'s web-based interface

and evaluation of the Ptolemy language. They have successfully given previous demonstrations at AOSD'10, FSE'10, ECOOP'11, and SPLASH'11.

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 frontend compiler, backend caching mechanisms, and additional supporting infrastructures.

Acknowledgments

Dyer and Rajan are funded in part by NSF grant CCF-10-17334. Tien Nguyen and Hoan Nguyen are funded in part by NSF grant CCF-1018600.