Boa: Analyzing Ultra-Large-Scale Code Corpus

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

Abstract

Software repositories contain an enormous amount of information such as revisions and bugs. Analyzing this data requires knowledge in mining software repositories and a large amount of infrastructure. We present our infrastructure *Boa* to ease such analyses. Our results show writing analyses with our framework is simpler and executes faster.

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

General Terms Experimentation, Languages

Keywords MapReduce, software repository mining

1. Introduction

There are many very large software repositories, such as SourceForge (350k+ projects), GitHub (250k+ projects), and Google Code (250k+ projects). These repositories contain massive amounts of interesting data with potentially billions of lines of source code just waiting to be mined. Many researchers and engineers would like to ask questions about this data however doing so requires a large amount of previous expertise in software mining and a large amount of existing infrastructure to perform the mining task(s).

Consider for example a simple question such as "how many revisions exist on SourceForge for all Java projects that also use Subversion?" Answering such a question would require knowledge of (at a minimum): how to read/scrape the project metadata from the repository, how to mine the code repository locations, how to access those code repositories, additional filtering code, controller logic, etc. This assumes that the query is performed locally and runs on a single machine, which might take an extremely long time to complete depending on the task. Writing such a program in Java for example, would take upwards of 100 lines of code and require knowledge of at least 2 complex libraries.

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

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

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

Instead, consider the example code written in *Boa* and shown in Figure 1. These 5 lines of code not only answer the question of interest, but run on a distributed cluster potentially saving hours of execution time. Note that writing this small program required no intimate knowledge of how to find/access the project metadata, how to access the repository information, or any mention of parallelization. All of these concepts are abstracted from the user providing simple primitives such as the `Project` type, which contains attributes related to software projects such as the name, programming languages used, repository locations, etc.

2. Related Work

There are a number of languages that provide efficient means for computing highly data-parallel tasks. These languages abstract as much of the parallelization details as possible from the user and scale to large numbers of machines.

Dean and Ghemawat describe a MapReduce system [**?**] where users write procedural code to take key-value paired input, filter it (via *mappers*), and then aggregate the results (via *reducers*). Languages like Sawzall [**?**] hide some of the map-reduce framework details from users, by requiring users to only write the map functions and then select from a set of pre-determined aggregators. Other languages like Dryad [**?**] and Pig Latin [**?**] provide SQL-like syntax and allow for more general computations. None of these languages however provide the software mining benefits of *Boa*.

3. Approach

Our language's syntax is inspired by the Sawzall programming language [**?**]. We also add additional domain-specific types for software mining. Our compiler is based on the Sizzle [**?**] open source implementation of Sawzall, which runs on the Hadoop [**?**] MapReduce [**?**] framework.

Boa abstracts the notion of map-reduce programs from the user. Users write only the map portion of a program and then select from a previously defined set of aggregation functions. Output in *Boa* is defined as tables, such as the table *total_revisions* (line 1). This table can have integers emitted to it (line 5) and proceeds to use the *sum* function to aggregate these integers into a final result, which is the sum of all values emitted to the table.

Input in *Boa* (line 2) is a domain-specific type provided by the language, the type `Project`. An overview of the provided types is given in Figure 2. These types provide attributes which can be used to filter the data. For example, this program uses the attributes to select only Java projects (line 3) that use Subversion (line 4) by using quantifiers and *when* statements.

Type	Attributes
Project	id, name, created_date, code_repositories, ...
Repository	url, repository_type, revisions
Revision	id, log, committer, commit_date, files
Person	username, real_name, email

Figure 2. Domain-specific Types Available in *Boa*

In order to provide the capability to reproduce results and speed up execution of queries in our language, we cache the repository data locally on our cluster. First we download the data in its raw format and then proceed to translate it into a custom data format. This custom data format is described using Protocol Buffers, which is a data description format developed by Google with the goals of making it compact in binary form as well as extremely fast to parse.

4. Early Results

Early results are quite promising, as shown in Figure 3. This table lists 3 different mining tasks. The first counts the number of revisions for Java projects using Subversion (the code shown in Figure 1). The second counts the number of committers for Java projects using SVN. The third determines how many projects use more than one programming language. All tasks were implemented in both Java and in *Boa*.

	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 3. 3 mining tasks implemented in Java and *Boa*. Results for lines of code (LOC) and execution time given.

The results are given in terms of lines of code required to implement the mining task as well as the running time on an input size of over 620k projects. The Java versions all required over 8 times as many lines of code and made use of several libraries (for SVN and JSON parsing). The *Boa* versions however only required around 5 lines of code per task. The execution times also show impressive results for *Boa*, taking less than one minute to run each task whereas the Java versions ran anywhere from 10 times slower to taking over a day to answer one single task.

We believe these early results show that our approach provides several benefits. First, programs are much smaller and simpler to write. They also require less knowledge about how you mine software. Second, program execution scales and thus even when querying extremely large datasets (such as all of SourceForge) the queries only take one minute.

5. Conclusions and Future Work

Despite having such a large wealth of information available in code repositories, mining that software is a difficult task requiring significant expertise not only in the mining technique used, but many additional libraries and approaches. There is a significant cognitive and infrastructure burden to overcome for anyone wishing to perform research in this area. Our language *Boa* and the associated framework helps users overcome this burden.

In the future we plan to extend our approach to additional repositories (GitHub, Google Code, etc) and additional version control systems (Git, CVS, Mercurial, etc). We also plan to investigate additional methods of abstraction to ease writing mining tasks and additional language constructs and data structures to provide further optimizations.

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.