Facilitating Consistency Check between Specification and Implementation with MapReduce Framework

Shigeru KUSAKABE, Yoichi OMORI, and Keijiro ARAKI

Grad. School of Information Science and Electrical Engineering, Kyushu University
744, Motooka, Nishi-ku, Fukuoka city, 819-0395, Japan

Abstract. We often need well-formed specifications in order to properly maintain or extend a system by members who were not in charge of the original development. In contrast to our expectation, formal specifications and related documents may not be maintained, or not developed in real projects. We are trying to build a framework to develop specifications from a working implementation. Testability of specifications is important in our framework, and we develop executable, or testable, formal specifications in model-oriented formal specification languages such as VDM-SL. We figure out a formal specification, check it with the corresponding implementation by testing, and modify it if necessary. While the specific level of rigor depends on the aim of the project, millions of tests may be performed in developing highly reliable specifications. In this paper, we discuss our approach to reducing the cost of specification test. We use Hadoop, which is an implementation of the MapReduce framework, so that we can expect the scalability in testing specifications. We can automatically distribute the generation of test cases from a property, the interpretation of the executable specification and the execution of its corresponding implementation code for each test data using Hadoop. While straightforward sequential execution for large data set is expensive, we observed scalability in the performance in our approaches.

1 Introduction

While we are supposed to have adequate documents in ideally disciplined projects, we may not have such documents in many actual projects. In spite of potential effectiveness of formal methods, formal specifications and related documents may not be maintained, or not developed. Nonetheless, we often need well-formed specifications in order to properly maintain or extend a system by members who were not familiar with the details of current implementation.

We are trying to build a framework to develop specifications for maintenance from the actual code of working implementation plus ill-maintained specifications for development if they exist. Fig. 1 shows our concept. We expect we can figure out some specifications by using techniques of software engineering while we assume documents used in the development may be unreliable and members familiar with the detail of the development may be unavailable. However, we will rely on testing to check the consistency between the implementation and the specifications in the process of making the specifications close to ideal ones.

We can develop executable, or testable, formal specifications in model-oriented formal specification languages such as VDM-SL. By using the interpretor of VDMTools,

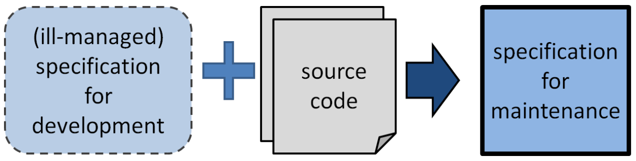
(ill-managed) specification for development
source code
specification for maintenance

Fig. 1. Developing specifications for maintaining or extending a system from source code plus specifications that are not well managed during the development.

we can test executable specifications in VDM languages to increase our confidence in the specifications. In our framework, we expect executable specifications to play an important role. We develop a formal specification from the running implementation, check it with the result or behavior of the corresponding implementation by testing, and modify and retest it if necessary. In contrast to the usual software development, we modify the specifications to be consistent with the corresponding running code.

In this paper, we discuss our approach to reducing the cost of testing specifications. In the testing phase, millions of tests may be performed in developing highly reliable specifications, while the specific level of rigor depends on the background of the project. Our approach is a brute-force one which adopts recent cloud computing technologies. We use Hadoop, which is an implementation of the MapReduce framework, so that we can expect the scalability in testing specifications. We can automatically distribute the generation of test data from a property, the interpretation of the executable specification and the execution of its corresponding implementation code for each test data using Hadoop. While straightforward sequential execution for large volume of test data is expensive, our preliminary evaluation indicates that we can expect scalability of our approach.

The rest of this paper is organized as follows. We explain our approach to reduce the cost of testing for large data set by using emerging cloud technology in section 2. We outline our framework in section 3. We evaluate our testing framework in section 4. Finally we conclude in section 5.

2 Approach with Cloud Technology

2.1 Elastic platform

Our approach shares the issues of testing with that of usual software development. As the size and complexity of software increase, its test suite becomes larger and its execution time becomes a problem in software development.

Large software projects may have large test suites. There are industry reports showing that a complete regression test session of thousands lines of software could take weeks of continuous execution [5]. If each test is independent with each other, high level of parallelism provided by a computational grid can be used to speed up the test execution [4]. Distributing tests over a set of machines aims at speeding up the test stage by executing tests in parallel [7].

We consider an approach to leveraging the power of testing by using elastic cloud platforms to perform large scale testing. Increasing the number of tests can be effective in obtaining higher confidence, and increasing the number of machines can be effective in reducing the testing time.

The cloud computing paradigm seems to bring a lot of changes to many fields. We believe it also has impact on the field of software engineering and consider an approach to leveraging light-weight formal methods by using cloud computing which has the following aspects [1]:

1. The illusion of infinite computing resources available on demand, thereby eliminating the need for cloud computing users to plan far ahead for provisioning;
2. The elimination of an up-front commitment by cloud users, thereby allowing organizations to start small and increase hardware resources only when there is an increase in their needs; and
3. The ability to pay for use of computing resources on a short-term basis as needed and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

We can prepare a platform of arbitrary number of machines and desired configuration depending on the needs of the project.

2.2 MapReduce

While we can prepare a platform of an arbitrary number of computing nodes and prepare an arbitrary number of test cases, we need to reduce the cost of managing and administrating of the platform and runtime environment.

The MapReduce programming model was created in order to generate and process large data sets on a cluster of machines [3]. Programs are written in a functional style, in which we specify mapper functions and reducer functions, as in `map` and `reduce` (or `fold`) in functional programming language. Fig. 2 shows the concept of `map` and `reduce`. For example, when we calculate square-sum of the elements in a sequence (list in typical functional programming languages), we specify a square function as the function f_M, an add as the function f_R, and 0 as the initial value init. In `map`, the function f_M, square, is applied to every element in the sequence, and in `reduce`, squared values are reduced to a single value by using f_R and init. In MapReduce programming framework, input data set is split into independent elements, and each mapper task processes each independent element in a parallel manner. Data elements are typically data chunks when processing a huge volume of data. The outputs of the mappers are sorted and sent to the reducer tasks as their inputs. The combination of map/reduce phase has flexibility, thus, for example, we can align multiple map phases in front of a reduce phase.

MapReduce programs are automatically parallelized and executed on a large cluster of machines. The runtime system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. Its implementation allows programmers to easily utilize the resources of a large distributed system without expert skills in parallel and distributed systems.

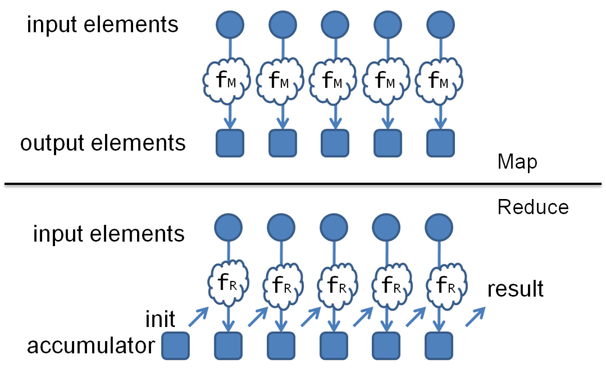

Fig. 2. Concept of the MapReduce programming model.

When using this MapReduce framework, input elements are test data, `f` can be an executable specification in VDM or actual code fragment under test, and output elements are test results.

3 Testing Framework

In this section, we discuss our testing framework, which uses Hadoop to reduce the cost of testing with a large data set. Hadoop is an open source software framework implementing MapReduce programming model [6] written in Java. While our framework can be adjusted to testing in a typical software development, we focus on testing specifications, which are figured out based on a corresponding implementation.

3.1 Property-based testing

Fig. 3 shows the outline of our testing framework. First, we generate test data according to the specified property. We use a property-based testing tool, QuickCheck [2], which supports a high-level approach to testing Haskell programs by automatically generating random input data. QuickCheck defines a formal specification language to state properties, and we can customize test case generation of QuickCheck including the number of test cases. We modify QuickCheck to fit to our approach for testing formal specifications with Hadoop. We try to automatically distribute the generation of test data for a formal specification in addition to the execution of the formal specification. We store the generated data in a file on the Hadoop file system. In the evaluation phase, we pass the test data to mappers, each mapper execute the executable specification and its corresponding implementation code for each test data, and then outputs the comparison result. Reducers receive and aggregate the comparison results.

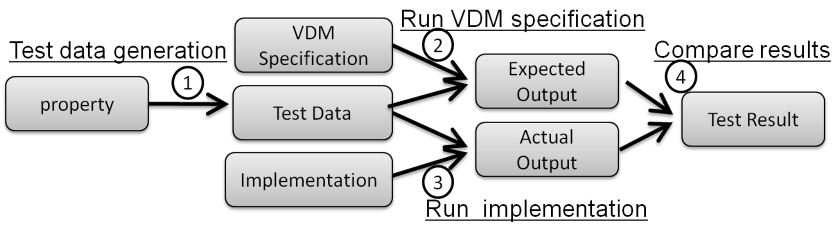

Fig. 3. Outline of our approach to property-based testing.

Table 1. Configuration of the platform

	NameNode	JobTracker	Slave
CPU	Xeon E5420	Xeon E5420	Xeon X3320
	2.50GHz 4core	2.50GHz 4core	2.50GHz 4core
Memory	3.2GB	8.0GB	3.2GB
Disk	2TB	1TB	140GB

3.2 Hadoop Streaming

In the Hadoop framework, we write mapper and reducer functions in Java by default. However, the Hadoop distribution contains a utility, Hadoop Streaming, which allows us to create and run jobs with any executable or script that uses standard input/output as the mapper and/or the reducer. The utility will create a mapper/reducer job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes. When an executable is specified for mappers, each mapper task will launch the executable as a separate process when the mapper is initialized. When an executable is specified for reducers, each reducer task will launch the executable as a separate process then the reducer is initialized. This Hadoop Streaming is useful in implementing our testing framework. We execute specifications in VDM with the combination of the command-line interface of VDMtools and the mechanism of Hadoop Streaming.

4 Performance evaluation

4.1 Evaluation of a formal specification for a large data set

In order to examine the effectiveness of our testing framework using Hadoop, we measured performance in testing a specification of the Enigma machine given in [8] for a large data set. The Enigma cipher machine is basically a typewriter composed of three parts: the keyboard to enter the plain text, the encryption device and a display to show the cipher text. Both the keyboard and the display consist of 26 elements, one for each letter in the alphabet.

The configuration of the platform is shown in Table 1. We show the result of elapsed time in Fig. 4. As we can see from the results, the elapsed time of the Hadoop version

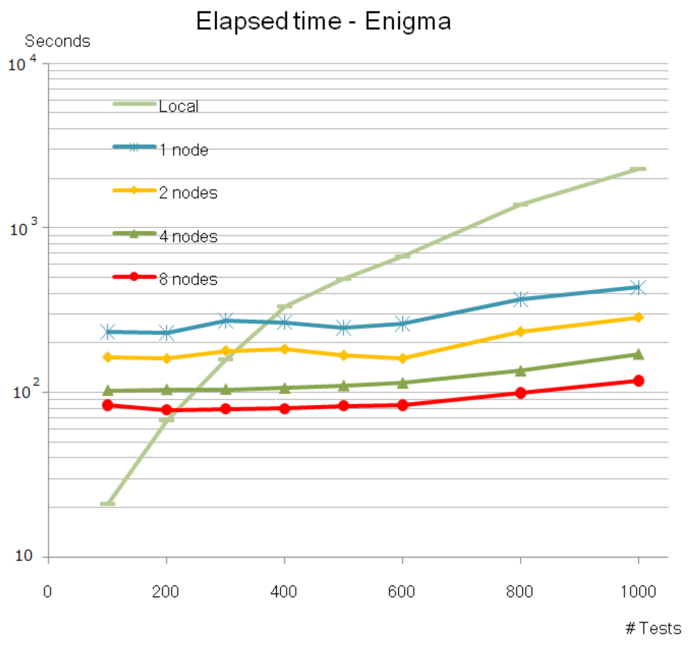

Fig. 4. Elapsed time for VDM enigma specification in increasing the number of tests on the various number of nodes.

reduced when the number of tests was over four hundreds. Since the Hadoop framework is designed for large scale data processing, we see no advantage in elapsed time for small set of test data. Each computation node has four processor cores, and we can achieve speedup even on a single node as Hadoop can exploit thread-level parallelism on multi-core platforms.

4.2 Evaluation of our testing framework for large data set

In order to examine the effectiveness of our approach, we measured elapsed time in testing with an implementation of a small address book system and the corresponding specification in VDM on Hadoop, for a variable number of test cases. The property we used is idempotency, which is an invariant to check if the sorting operation obeys the basic rule: applying sort twice has the same result as applying it only once.

We show the result in Fig. 5. As we see in Fig. 5, until the number of tests exceeds about 300, the total elapsed time of the Hadoop version is more than that of non-Hadoop version while the former uses eight nodes and the latter uses only a single local node. The results of Hadoop version include some overheads such as distributing test data and collecting evaluation results over the network. After that point, the gap between the two

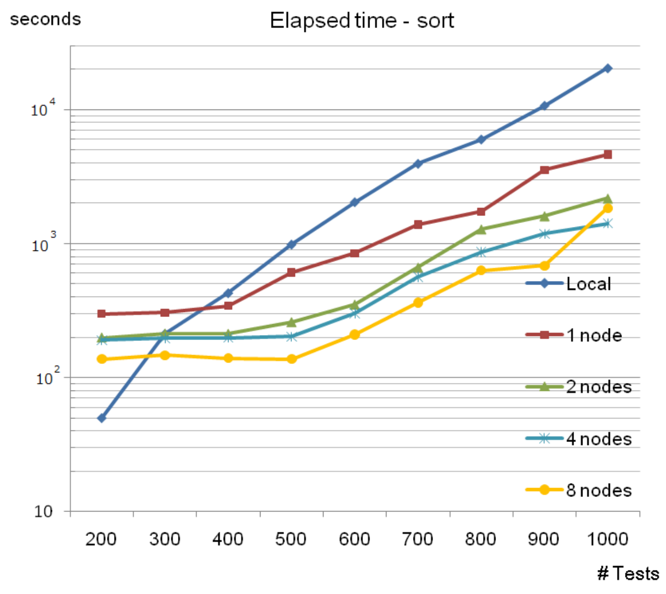

Fig. 5. Elapsed time in increasing the number of test data for idempotency on the various number of nodes.

version becomes wider (i.e. the Hadoop version becomes faster) as the number of test data increases. Thus, our approach is suitable for a large scale test data set.

We show the result on speedup in Fig. 6. The speedup ratio is calculated as *(time for N Hadoop nodes) / (time for local single node)*. As we see in Fig. 6, the increase of the number of slave machines is generally effective in reducing testing time. However, the speedup ratio against the number of slaves does not seem ideal. There are some troughs in the graph. While one of the reasons is overhead of using Hadoop, we will investigate further to achieve more efficient environment.

5 Concluding Remarks

In this paper, we presented preliminary evaluation results of our approach to reducing the cost of testing executable specifications for a large data set using Hadoop. Testability of a specification helps us increase our confidence in the specification. We are trying to develop executable formal specifications while we assume documents used in the development may be unreliable and members familiar with the detail of the development may be unavailable. We rely on testing to check the consistency between the implementation and the specifications. In order to increase our confidence in the specification, we can increase the number of test cases on elastic computing platforms at reasonable cost.

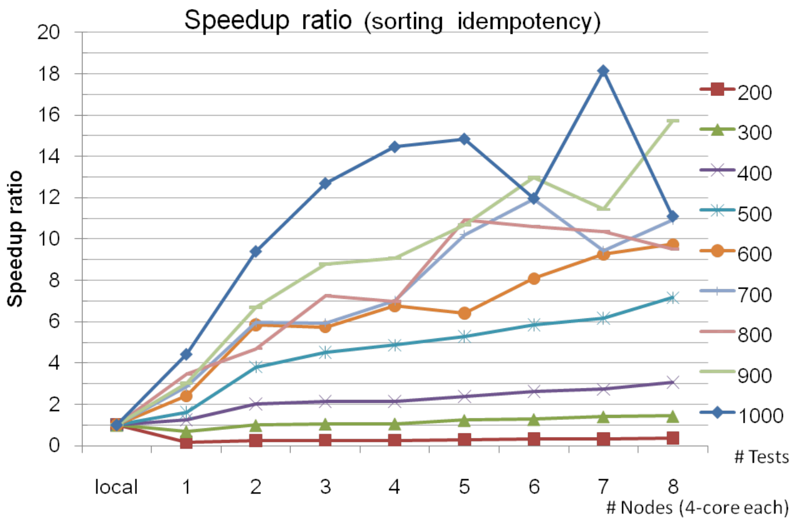

Fig. 6. Speedup in increasing the number of test data for idempotency on the various number of nodes.

We are able to automatically distribute the interpretation of the executable specification and the execution of its corresponding implementation code for each test data by using Hadoop. While straightforward sequential execution for large data set is expensive, we observed scalability in the performance in our approaches.

As one avenue of future works, we will investigate a more detailed performance breakdown to achieve more efficient environment. We will also try to extend usability of our framework. VDMTools and other programming language systems include test coverage statistics tools. We will extend our framework to exploit these tools in a parallel and distributed way to examine the impact of our approach on increasing test coverage.

References

1. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, UCB/EECS-2009-28, Reliable Adaptive Distributed Systems Laboratory, February 2009.
2. Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. *ACM SIGPLAN Notices*, 35(9):268–279, 2000.
3. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
4. A. DUARTE, W. CIRNE FILHO, F. V. BRASILEIRO, and P. D. L. MACHADO. Gridunit: Software testing on the grid. In *Proceedings of the 28th ACM/IEEE International Conference on Software Engineering*, volume 28, pages 779 – 782. ACM, 2006.

5. Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Prioritizing test cases for regression testing. In *In Proceedings of the International Symposium on Software Testing and Analysis*, pages 102–112. ACM Press, 2000.
6. Hadoop. As of Jan.1, 2011. http://hadoop.apache.org/.
7. G. M. Kapfhammer. Automatically and transparently distributing the execution of regression test suites. In *Proceedings of the 18th International Conference on Testing Computer Software*, 2001.
8. Peter Gorm Larsen, Paul Mukherjee, Nico Plat, Marcel Verhoef, and John Fitzgerald. *Validated Designs For Object-oriented Systems*. Springer Verlag, 1998.