LEVERAGING LIGHT-WEIGHT FORMAL METHODS WITH FUNCTIONAL PROGRAMMING APPROACH ON CLOUD

Shigeru Kusakabe, Yoichi Ohmori and Keijiro Araki
Graduate School of Information Science and Electrical Engineering, Kyushu University
744, Motooka, Nishi-ku, Fukuoka city, 819-0395, Japan

Keywords: Light-weight formal methods, Testing, Cloud computing, Parallel processing, Functional programming, MapReduce programming.

Abstract: We discuss the features of functional programming related to formal methods and an emerging paradigm, Cloud Computing. Formal methods are useful in developing highly reliable mission-critical software. However, in light-weight formal methods, we do not rely on very rigorous means, such as theorem proofs. Instead, we use adequately less rigorous means, such as evaluation of pre/post conditions and testing specifications, to increase confidence in our specifications. Millions of tests may be conducted in developing highly reliable mission-critical software in a light-weight formal approach. We consider an approach to leveraging light-weight formal methods by using "Cloud." Given a formal specification language which has the features of functional programming, such as referential transparency, we can expect advantages of parallel processing. One of the basic foundations of VDM specification languages is Set Theory. The pre/post conditions and proof-obligations may be expressed in terms of set expressions. We can evaluate this kind of expression in a data-parallel style by using MapReduce framework for a huge set of test cases over cloud computing environments. Thus, we expect we can greatly reduce the cost of testing specifications in light-weight formal methods.

1 INTRODUCTION

Functional programming has many advantages including features useful to create short, fast, and safe software(Hughes, 1989). In this paper, we focus on the features serving as a glue between light-weight formal methods and a new programming model MapReduce on emerging Cloud Computing paradigm.

While formal methods are useful in developing highly reliable mission-critical software, we do not rely on very rigorous means, such as theorem proofs, in light-weight formal methods. Instead, in order to increase confidence in our specifications, we use adequately less rigorous means, such as evaluation of pre/post conditions and testing specifications. While the specific level of rigor depends on the aim of the project, millions of tests may be conducted in developing highly reliable mission-critical software in a light-weight formal approach. For example, in an industrial project using VDM++, a model-oriented formal specification language(Larsen et al., 1998), they developed formal specifications of about 100,000 steps including test cases (about 60,000 steps) and comments written in the natural language(Kurita et al., 2008). They carried out about 7,000 black-box tests and 100 million random tests. We believe Cloud Computing is useful in performing this kind of activity.

The Cloud Computing paradigm seems to bring a lot of changes in many IT 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. Given a formal specification language, such as VDM-SL, which supports functional programming style(Fitzgerald and Larsen, 1998), we can exploit the features suitable for parallel processing such as referential transparency. One of the basic foundations of VDM specification languages is Set Theory. The pre/post conditions and proof-obligations may be expressed in terms of set expressions. We can evaluate this kind of expression in a data-parallel style by using MapReduce framework for a huge set of test cases over cloud computing environments(Dean and Ghemawat, 2008). By using MapReduce framework, we expect we can greatly re-

Kusakabe S., Ohmori Y. and Araki K. (2009).
LEVERAGING LIGHT-WEIGHT FORMAL METHODS WITH FUNCTIONAL PROGRAMMING APPROACH ON CLOUD.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 263-268
DOI: 10.5220/0002281802630268

duce the cost of testing specifications in light-weight formal methods. MapReduce framework provides an abstract view of computing platforms, and is suitable for Cloud Computing which has the following aspects(Armbrust et al., 2009):

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.

2 LIGHT-WEIGHT FORMAL METHODS

Formal methods are useful in developing highly reliable mission-critical software, and can be used at different levels:

Level 0: In this light-weight formal methods level, we develop a formal specification and then a program from the specification informally. This may be the most cost-effective approach in many cases.

Level 1: We may adopt formal development and formal verification in a more formal manner to produce software. For example, proofs of properties or refinement from the specification to an implementation may be conducted. This may be most appropriate in high-integrity systems involving safety or security.

Level 2: Theorem provers may be used to perform fully formal machine-checked proofs. This kind of activity may be very expensive and is only practically worthwhile if the cost of defects is extremely expensive.

In light-weight formal methods, we do not rely on very rigorous means, such as theorem proofs. Instead, we use adequately less rigorous means, such as evaluation of pre/post conditions and testing specifications, to increase confidence in specifications, while the specific level of rigor depends on the goal of the project.

One of the formal methods is to use a model-oriented formal specification language, such as VDM-SL and VDM++, to write, verify, and validate specifications of the products. VDM languages have a tool with functionalities of syntax check, type check, interpreter, and generation of proof-obligations. We can use the interpreter to evaluate pre/post conditions in the specifications and test the specifications. In addition to explicit-style executable specifications, implicit-style non-executable specifications can be checked through the evaluation of pre/post conditions, and proof-obligations by using interpreter.

One of the basic foundations of the VDM specification language is Set Theory. The pre/post conditions and proof-obligations are expressed in terms of set expressions, so that we expect these expressions can be evaluated by using MapReduce framework. In addition to evaluating different test scenarios in parallel, we can exploit data-parallelism in fine-grained boolean expressions over huge sets of values. We generate test cases and easily unfold their evaluation over cloud computing environments.

3 LIGHT-WEIGHT FORMAL APPROACH WITH VDM ON CLOUD

Figure 1 shows the outline of our light-weight formal approach. In our light-weight formal approach, we first develop a formal model in VDM languages. Executable specification in VDM languages can be evaluated with interpreter. We can also translate a large part of VDM models into models in a functional programming language as VDM languages share several features with functional programming languages. There have been works focusing on their relations(Borba and Meira, 1993)(Visser et al., 2005). Both types of model may be evaluated using MapReduce framework on Cloud, while functional programming languages seem to have more integrated evaluation environment.

3.1 Modelling in VDM

We have following components in formal specification models in VDM.

- Data types are built from basic types (`int`, `real`, `char`, `bool` etc.), and built by using type constructors (`sets`, `sequences`, `mappings`, `records`). Newly constructed types can be named and used throughout the model.
- A data type invariant is a Boolean expression used to restrict a data type to contain only those values that satisfy the expression.
- Functions define the functionality of the system. Functions are referentially transparent with no

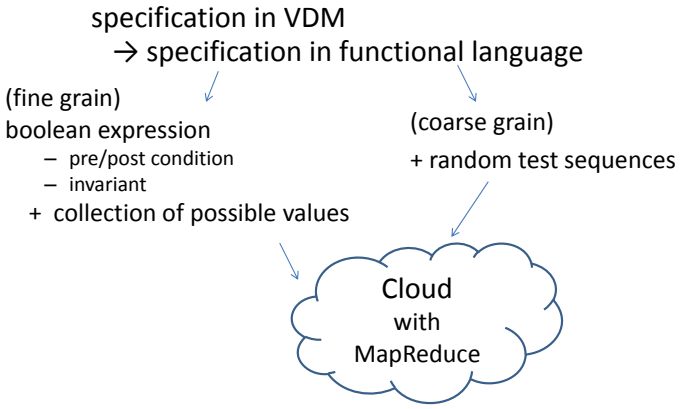

Figure 1: Outline of our approach using Cloud.

side-effects and no global variables. A different modelling style, operational style, is used in cases where it is intuitive to have states.

- A pre-condition is a Boolean expression over the input variables. This expression is used to record restrictions assumed to hold on the inputs.
- A post-condition is a Boolean expressions relating inputs and outputs. Function abstraction is provided by implicit specification when appropriate. Post-conditions are used when we do not wish to explicitly define which output is to be returned, or where the explicit definition would be too concrete at the time of modelling.

As an approach of validation to check whether the model of the target system behaves as desired, we can test the specification model. In order to efficiently check the specification on cloud, we discuss inherent parallelism in evaluating VDM specifications.

3.2 Specification over Collection

There are type constructors to define a finite collection of values. Conditions using values of such types may be evaluated over a collection of values in a data-parallel manner.

```
set of _     Finite sets
seq of _     Finite sequences
map _ to _   Finite mappings
```

We discuss set as an example of source of parallelism in this section. The `set of` is a finite set type constructor and there are three ways of defining sets: enumeration, subrange, and comprehension. Comprehension is a powerful way to define a set. The form of a set comprehension is:

```
{value-expression | binding & predicate}
```

The `bindings` binds one or more variables to a type or set. The `predicate` is a logical expression using the bound variables. While the `value-expression` is an expression using the bound variables, the `value-expression` defines elements of the set being constructed.

Evaluation of a set comprehension generates all the values of the expression for each possible assignment of range values to the bound variables for which the predicate is true. We often use sets in the specification model in VDM languages. We may have to handle a huge number of values even in a single evaluation of a specification.

We can gain confidence through tests on the formal model. Systematic testing is possible, in which we define a collection of test cases, execute each test case on the formal model, and compare with expectation. Test cases can be generated by hand or automatically. Automatic generation can produce a vast number of individual test cases.

VDM languages share many features with functional programming languages. We expect we can exploit MapReduce framework in evaluating VDM specifications. Techniques for test generation in functional programs can carry over to formal models especially when written in the functional style(Claessen and Hughes, 2000).

3.3 Using MapReduce

MapReduce programming model is useful in processing and generating large data sets on a cluster of machines. Programs are written in a functional style, in which we specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Its implementation allows programmers to easily utilize the resources of a large distributed system

without expert skills for parallel and distributed systems. MapReduce programs are automatically parallelized and executed on a large cluster of machines. The run-time 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.

3.4 Hadoop Streaming

Hadoop, open source software written in Java, is a software framework implementing MapReduce programming model(Hadoop,). While we write mapper and reducer functions in Java by default in this Hadoop framework, Hadoop distribution contains a utility, Hadoop streaming. The utility allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. The utility will create a Map/Reduce 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 invoking interpreter of VDM languages and executing functional programs with MapReduce framework.

4 PRELIMINARY EVALUATION

We consider the impact of Cloud with MapReduce framework on leveraging light-weight formal methods. We focus on the productivity in preparing the platform and the scalability of the platform in increasing its size.

4.1 Platform

According to (Armbrust et al., 2009), Cloud Computing is the sum of SaaS and Utility Computing, but does not normally include Private Clouds, which is the term to refer to internal datacenters of a business or other organization that are not made available to the public. However, we use a private cloud computing platform, which is a small version of IBM Blue Cloud, for our preliminary evaluation. Figure 2 shows the outline of our cloud. In our Cloud platform, we can dynamically add or delete servers which compose the Cloud if the machines are x86 architecture and able to run Xen. We can add a new server as the resource of Cloud by automatically installing and setting up Domain 0 (Dom0) through network boot. When a user requests a computing platform from the Web page of Cloud portal, the user can specify the virtual OS image (Domain U (DomU) of Xen in our platform) and applications from registered ones as well as the number of virtual CPUs (VCPUs), the amount of memory and storage within the resource of Cloud. In our Cloud, the number of VCPUs is limited within the number of physical CPUs to guarantee the performance of DomU. When the request is admitted, the requested computing platform is automatically provided.

Our Cloud supports an automatic set up of Hadoop programming environment in provisioning requested platforms. We need the following steps to set up Hadoop environment:

1. Installing base machines into nodes
2. Installing Java
3. Mapping IP address and hostname of each machine
4. Permitting non-password login from the master machine to all the slave machines
5. Configuring Hadoop on the master machine
6. Copying the configured Hadoop environment to all slave machines from the master machine

Setting up Hadoop Platform (Step 2 - 6). Our Cloud is able to perform the step 2, 3, 5, and 6 automatically to set up the Hadoop environment by selecting Hadoop as the application in provisioning the platform.

Addition of Base Machine. For the step 1, we only have to set the machine network bootable in the BIOS configuration when adding the machine to our Cloud.

4.2 Scalability in Cloud Platform

We evaluate the scalability of our approach when increasing the number of the slave machine nodes of in running Hadoop streaming on our Cloud. We measure the elapsed time of simple program with the data of 1GB and 5GB. We change the number of the slave machines from 2 to 9. All the slave machines have 1GB of memory and 20GB of storage. Users cannot control the allocation of the master and the slave machines on the physical machines in our Cloud.

We show the result in Figure 3. As we see in Figure 3, the increase of the number of nodes does not always reduce the elapsed time in both data size. For example, the elapsed time increases when we increase the number of DomUs from 2 to 3 with the data of 1GB and from 4 to 5 with the data of 5GB.

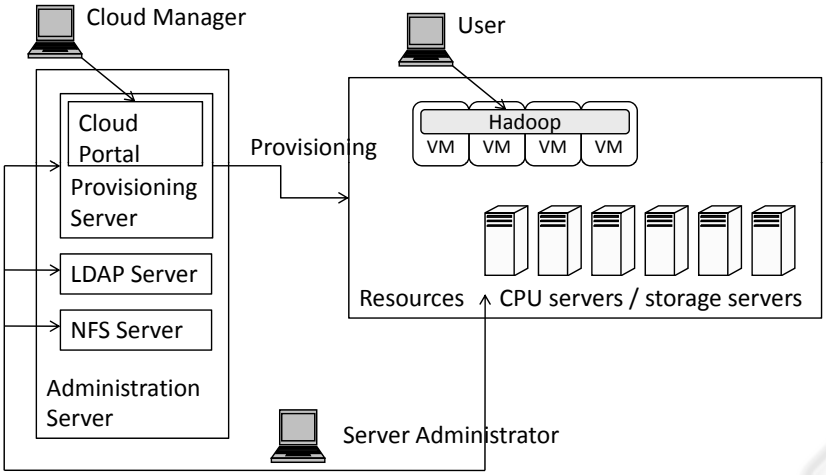

Figure 2: Overview of our private Cloud.

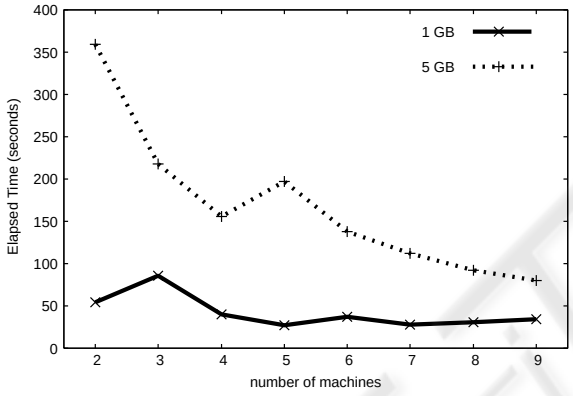

Figure 3: Scalability of a small program in increasing the number of nodes of Hadoop.

Thus, the performance of Hadoop did not always scale according to the number of the slave machines provisioned in our Cloud. Users are not aware of the hierarchy of the locality between each physical machine such as machines, chassis, and racks. The group of nodes provisioned over different hierarchies can degrade the performance caused by the cost of communication. However, we can expect some scalability, while it is easy to set up platform and scale up/down the size of platform.

5 CONCLUDING REMARKS

We discussed our idea to leverage light-weight formal methods, which use specification languages with functional style, such as a model oriented forma specification language VDM-SL, and a functional programming languages, such as Haskell, on Cloud. By applying MapReduce framework, we can expect scalable speed-up without caring the details of parallel and distributed processing. Although preliminary evaluation shows some problems and we have a lot of feature works, we believe our approach is promising.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, UCB/EECS-2009-28, Reliable Adaptive Distributed Systems Laboratory.

Borba, P. and Meira, S. (1993). From vdm specifications to functional prototypes. *J. Syst. Softw.*, 21(3):267–278.

Claessen, K. and Hughes, J. (2000). Quickcheck: a lightweight tool for random testing of haskell programs. *ACM SIGPLAN Notices*, 35(9):268–279.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Fitzgerald, J. and Larsen, P. G. (1998). *Modelling Systems: Practical Tools and Techniques in Software Development*. Cambridge University Press.

Hadoop. As of Jun.1, 09. http://hadoop.apache.org/core/.

Hughes, J. (1989). Why functional programming matters. *Computer Journal*, 32(2):98–107.

Kurita, T., Chiba, M., and Nakatsugawa, Y. (2008). Application of a formal specification language in the development of the ”mobile felica” ic chip firmware for embedding in mobile phone. In *FM*, pages 425–429.

Larsen, P. G., Mukherjee, P., Plat, N., Verhoef, M., and Fitzgerald, J. (1998). *Validated Designs For Object-oriented Systems*. Springer Verlag.

Visser, J., Oliveira, J. N., Barbosa, L. S., Ferreira, J. a. F., and Mendes, A. S. (2005). Camila revival: Vdm meets haskell. In *First Overture Workshop*.