Evaluating Data Integrity in the Cloud using the UPPAAL

Sachi Nishida and Yoshiyuki Shinkawa

1Fujitsu FSAS Inc., 13-2 Nakamaruko, Nakahara-ku, Kawasaki, Kanagawa, Japan
2Graduate School of Science and Technology, Rikukoku University, 1-5 Seta Oe-cho Yokotani, Otsu, Shiga, Japan

Abstract: There are several considerations when implementing a transaction processing system in cloud environments. One of the most critical ones is the data integrity since the cloud provides us with limited capacity for it. Therefore, we need to evaluate the applications and the cloud platform carefully from the data integrity viewpoint. This paper presents a model-based data integrity evaluation method using the UPPAAL model checker in order to verify the data integrity evaluation in the cloud. The data integrity evaluation is performed in two different ways. One is a simulation-based method in which the model is executed by the UPPAAL simulator to obtain the resultant variable values. The other is a verification-based method in which the given integrity constraints are examined by the UPPAAL verifier using full state space search of the model.

Keywords: Cloud Computing, UPPAAL, Transaction Processing, Data Integrity.

1 INTRODUCTION

Data integrity is one of the most critical concerns for distributed and concurrent systems, especially for those in cloud environments. One of the most critical ones is the data integrity since the cloud provides us with limited capacity for it. Therefore, we need to evaluate the applications and the cloud platform carefully from the data integrity viewpoint. This paper presents a model-based data integrity evaluation method using the UPPAAL model checker in order to verify the data integrity evaluation in the cloud. The data integrity evaluation is performed in two different ways. One is a simulation-based method in which the model is executed by the UPPAAL simulator to obtain the resultant variable values. The other is a verification-based method in which the given integrity constraints are examined by the UPPAAL verifier using full state space search of the model.

1. While the ACID restricts the concurrent database accesses within a critical section, the BASE allows arbitrary concurrent database accesses from any transaction.
2. While the ACID postulates the transparent replication of the database, the BASE tolerates non-transparent replication.
3. While the ACID aims at the data integrity at every instant, the BASE tries for achieving the data integrity within some duration.

According to the above differences between these two principles, we evaluate the ACID and BASE in the cloud. Since this evaluation must be performed before the system implementation, we need a precise model that reflects the cloud platform mechanism implementing these principles, namely, ACID and BASE.

This new and different principle is referred to as BASE, which stands for "Basically Available" and "Eventually Consistent." The ACID and BASE are distinguished by their different properties with respect to the data integrity of traditional transaction processing, which is adopted by the cloud.

In order to guarantee the data integrity in transaction processing, the ACID restricts the concurrent database accesses within a critical section, the BASE allows arbitrary concurrent database accesses from any transaction, and the BASE tries for achieving the data integrity within some duration.

ACID stands for Atomicity, Consistency, Isolation, and Durability.
The ASE principle is a newly introduced principle to cope with the conflicting requirements that are specific to the cloud. In order to prove the validity of the ASE principle, it is necessary to understand the BASE principle more rigorously in terms of evaluating data integrity. For maintaining the data integrity in such an environment, a TPM following the BASE principle provides us with version information instead of a log. The relevant transaction aborts if the referred data are invalid. The non-transparent or asynchronous replication allows the referred data to be valid. If so, the referred data are invalid; the relevant transaction aborts the database updates. This mechanism is known as optimistic locking.

For this purpose, we use the UPPAAL model checker. The rest of the paper is organized as follows. In section 2 we introduce the basic concepts of the data integrity in the cloud along with the transaction behavior following the ASE principle. Section 3 shows how transaction processing in the cloud is modeled using the UPPAAL. Section 4 discusses an evaluation and validation method for the data integrity using the UPPAAL.

2 TRANSACTION PROCESSING IN THE CLOUD

Data integrity in transaction processing has been hitherto relying on the ACID principle that is guaranteed by a transaction processing monitor (TPM) under which they are running. One of the ac grounds of the ACID is that the serial and the execution level of the TP isolates and serializes the critical sections of each transaction. The ACID i plicitly presumes the transparent replication or synchronous replication of database updates to real-time consistency property of it.

This approach could cause the reduction of data availability along with the performance degradation of transaction processing. In cloud computing, the ACID principle eco es are a burden too much to guarantee the high availability scalability and stability. Therefore, one of the aspects of the system is the lightweight echanism to attain the data integrity is desired in the cloud.

Fitzgerald et al. (2004) and so on, which are specialized to the functional aspect, system modeling tools like Petri nets, SDL (Thiel, 2001) and so on, which are specialized to the behavioral aspect, and architecture oriented modeling tools like UML class diagrams, and so on, which are specialized to the structural aspect.

On the other hand, it is desirable to press the multiple aspects of a system simultaneously in a single model for accurate evaluation of the data integrity. For this purpose, we use the UPPAAL model checker. David et al. (2015) as a modeling and evaluation tool, since it can express the behavior of a system as a set of timed automata connected through communication channels along with the functional and data structure specifications using a C-like language provided by the tool.

The language (Jensen and Kristensen, 2009) is used differently in various contexts. For example, it deals with database related matters. Therefore, it is necessary to define rigorously the semantics of the data rather than their structure. The concept of data integrity rigorously in order to evaluate it effectively. The term integrity or data integrity is used differently in various contexts. For example, it deals with database related matters. Therefore, it is necessary to define rigorously the semantics of the data rather than their structure. The concept of data integrity rigorously in order to evaluate it effectively.

Consequently, we first need to define rigorously the concept of data integrity at the application level using a unified notation. The data integrity at the application level can be defined as a set of constraints or rules on database occurrences. One of the ways to express these constraints is to use predicate logic formulas. Shin and colleagues (2012) in order to cope with these logic for ulae we rst have to define the language L and the structure S to provide the syntactic and semantic properties of it.
er ones are the functions or predicates defined in a database manipulation language like SQL. On the other hand, the latter ones are those used in a specific application domain and application dependent part where the former part can reuse a single function for each application domain.

Therefore we need to prepare the $L$ as composed of two parts namely the application independent part and application dependent part. On the former part, the latter is referred to the functions and predicates or assigned to variables and constants using the elements of the above defined in the database manipulation language.

The interpretation $I$ maps each symbol in the $L$ to an actual entity within the $D$ structure and the interpretation $I$ consists of the domain of discourse $D$ and the database $D$. All the objects that are referred to from the functions and predicates or assigned to variables and constants are the elements of the above defined in the database manipulation language.

In our case this includes
1. all the database instances $D_i$
2. all the database records $r_j(i)$ in each $D_i$ and
3. all the attribute values $a_k(i,j)$ in each $r_j(i)$

The interpretation $I$ maps each symbol in the $L$ to an actual entity within the $D$. Some of the functions and predicates are predefined in a database manipulation language like SQL. Other symbols in the $L$ are defined during the modeling process discussed in the succeeding sections.

Using the above language $L$ and the structure $S$, each constraint is expressed logically as a standardized logic formula (PCNF).

$$Q_1 \cdot \ldots \cdot Q_n \left( \bigwedge_{j} P_j(t_1^{(i)} \ldots t_m^{(i)}) \right)$$

where $Q_j$ is a variable with the quantifier $\forall$ or $\exists$, $P_j$ is a predicate and $t_k^{(i)}$ is a term composed of variables and constants.

Schoening 2008

There are several kinds of constraints regarding data integrity like restrictions on table values, constraints on values derived from a set of records. However, any kinds of those constraints can be expressed by the above predicate logic as follows:

1. The initial state is set up the database to be used during the simulation. The data as set $e$ pressed as three-dimensional integer arrays. The first dimension represents the replication number, the second represents the record number and the third represents the attributes in the database schema.

2. The scheduling module sends a transaction to one of the Thread instances to process it. A transaction is expressed in the form of an integer array with each element of which represents an argument or parameter to the transaction. These integer arrays compose a two-dimensional integer transaction list.

3. The Thread module performs the functionality of each transaction. The functionality is determined by the transaction type and the specific arguments in the transaction list. The database is updated by the transaction. The transaction is routed to the Data as module through a UPPAAL channel.

4. The Data as module is to be instantiated as any database as replication. Each instance reads and updates each replicated set of records. The transaction is expressed in the form of an integer array.

5. The replication module tries to keep the replicated data as identical in an asynchronous way.

Since the UPPAAL allows only fixed size for arrays in each transaction type could reuse the different number of arguments an individual two-dimensional integer array is defined for each transaction type independently.

3 MODELING THE TRANSACTION PROCESSING WITH THE BASE PRINCIPLE

Once the rules or constraints for data integrity are expressed in the form of predicate logic for database, the next step is to model the transaction processing with the ASE principle which updates the data as in the cloud. For this modeling we use the UPPAAL model checker or the UPPAAL in short. The UPPAAL e presses a system as a set of finite automata with varia les along with the functions that manipulate the AUT.

Each ti ed aut aton consists of states locations in terms of the UPPAAL and arcs edges in terms of the UPPAAL that represent the state transitions. Each state transition is expressed with clock type variables can be used as time constraints which are associated with any a ove stated location or edge. These transitions are defined as parameterizable templates, and each transaction type could require the different number of arguments, an individual two-dimensional array is defined for each transaction type independently.
implementing the Soft state property. This module is instantiated only once and deals with all the databases and their replications.

In addition to the above modules, we have to prepare several functions to make the model executable and verifiable. These functions are written in a C-like UPPAAL unique language. While the model structure is common among application domains, these functions are application unique and must be built for each application domain.

Figure 1 through Figure 5 show an example of the above UPPAAL modules. As stated above, the structure of their five modules can be commonly used among different application domains, including function names and channels associated with edges and locations in the model. However, the implementation of these functions and other supplemental functions are differently built among different application domains.

For example, the function “dbLoad()” in Figure 1 represents a function that initializes all the databases in the system and the name is common for all applications. However, its implementation is usually different depending on the structure and usage of the databases. Figure 6 shows a simple example of the database module for a simplified library application.

In executing the above modules, these databases are instantiated through the system definition as shown in Figure 7. In this example, three concurrent threads and three database replications are used.

These modules operate as follows:

1. Firstly, the dbLoad function of the Initialization module is invoked to prepare all the databases. At this time, only the associated edge is eligible for transition, and other modules are waiting for signals through the UPPAAL channels.
2. After the completion of the dbLoad function, the Scheduler module is activated through the “initS” channel.
3. The scheduler module sends a signal to the Thread module through the channel “S2T”.
4. The Thread module selects a transaction from the predefined transaction list by the “selectTran” function and sends a signal to the Database module through the channel “T2D”.
5. The Data module accesses and updates the databases.

Evaluating Data Integrity in the Cloud using the UPPAAL
4 DATA INTEGRITY EVALUATION USING THE UPPAAL

The UPPAAL model checker provides us with three a or functionalities. The rst is a graphical ode editor with progr aility that we have used in the previous section. The second is a odel siulator that e ceues the odel we ulid to show an instance of its ehavior. The third is a odel veri er that e a ines all the possi le ehavior whether the odel sati es the given properties written in the form of CTL Co putational Logic Tree for ulae.

Therefore two alternative ways are availa le to evaluate the data integrity of transaction processing. The rst is to ecue the odel to a in the values of the varia les for the data a se records at each state transition. As discussed in the previous section the data integrity is e pressed as a set of pred icate logic for ulae in the form of PC F. In the UPPAAL odel these logic for ulae refer to the vari a les associated with the data a se records and at tri utes. Therefore we can deter ine whether the data integrity is a ined in the transaction processing y e a ining the a ove varia les using a function i ple enting each constraint logic for ulae. Since this method can evaluate only one instance of the system behavior selected by the simulation we have to perform the siulation for every possi le ehavior. However this possible behavior could e uncount ed. Therefore this method would e a plying a ed evaluation.

On the other hand the UPPAAL veri er provides us with a ca pability of full state space search against a set of CTL for ulae. In order to evaluate the data integrity in this way we have to tran for a set of predicate logic for ulae into a set of CTL for ulae. Unli e the predicate logic for ulae can include the path operator A and E which deal with state transition paths of a system and the poral operator and which de ne the validation points of the for ulae. In addition there are no ners and in CTL. Therefore several considerations should e a en into account in the a ove tran for ation to predicate logic for ulae into CTL for ulae. These considerations include:

1. If a property P ust always holds in a predicate logic for ulae the CTL for ulae is .
2. If a property P always holds a property Q then the CTL for ulae is .
3. If a property P eventually holds a property Q the CTL for ulae is .
4. If a property $P$ must hold at a specific point, we introduce a boolean variable to express the point, and set it to $true$ at the point in the model. In this case we need to modify the model.

5. If the original predicate logic formula includes the quantifiers $\forall$ and $\exists$, we introduce a boolean function into the model to examine whether all of or some of the variables in the model satisfy the formula. A model modification is required in this case again.

After the above transformation is completed, we can evaluate the data integrity by running the verifier that the UPPAAL provides. This CTL based evaluation seems simpler than the simulator based one, however it performs full state space search and consumes huge computing resources. As a result, it takes long time to obtain the result. In such cases, we need to reduce the model, by decreasing the number of variables or values to be assigned.

5 CONCLUSIONS

In cloud environments, the behavior of transaction processing is considerably different from the traditional ones. One of the main reasons is that the cloud introduces a new principle for data integrity called "BASE", instead of the traditional "ACID". In order to make the transaction processing stable in the cloud, we need to clearly reveal the behavior of it and evaluate the data integrity rigorously.

This paper proposed a model-based data integrity evaluation using the UPPAAL model checker. In order to make the model easily understandable and reusable, we composed it using five functional modules, namely, "Initialization", "Scheduling", "Thread", "Database", and "Replication", following the BASE principle. While the model structure can be reused among different application domains, we need to build application unique functions for the model.

The UPPAAL provides us with two different ways to evaluate the data integrity. One is a simulation-based evaluation that examines only one instance of the behavior of transaction processing. The other is a verifier-based evaluation that examines full state space search to determine whether the given constraints are satisfied. While the latter way can evaluate the integrity more precisely, we need to transform the original predicate logic formulae into the CTL formulae. In addition, it consumes huge computing resources for full state space search and takes long time to obtain the evaluation results.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number 25330094.

REFERENCES


Fitgerald Larsen P u her ee P Plat and erhoef 2004 Validated Designs for Object-oriented Systems Springer.

ray and enter A 1993 Transaction Processing: Concepts and Techniques organ auf ann.


Pritchett D 2008 ASE An ACID alternative In ACM QUEUE Volume 6 Issue 3 pages 48–55 AC.


Sanderson D 2009 Programming Google App Engine Oreilly Associates Inc.

Schoening U 2008 Logic for Computer Scientists (Modern Birkhaeuser Classics) ir hausers oston.


van liet and Paganeli F 2011 Programming Amazon EC2 Oreilly Associates Inc.