A Comprehensive Evaluation Model for BASE Transaction Processing

Sachi Nishida and Yoshiyuki Shinkawa
Graduate School of Science and Technology, Ryukoku University,
1-5 Seta Oe-cho Yokotani, Otsu, Shiga, Japan

Keywords: Data Integrity, Colored Petri Net, Cloud Computing, Transaction Processing.

Abstract: Transaction processing has been thought to be not suitable for cloud computing. The primary reason is that the data integrity principle is much relaxed from the traditional ACID. This new principle is known as BASE. In order to deploy transaction processing systems over cloud environments, we need to reveal the characteristics of transaction behavior under the BASE principle. The paper proposes a CPN (Colored Petri Net) based comprehensive integrity evaluation model with high modularity, which can easily be customized, or enhanced for diverse application domains and cloud environments.

1 INTRODUCTION

Cloud computing is recently used in many application domains as a flexible, highly available, scalable, and low-cost platform. However, most applications in this environment are so-called non-mission-critical or ad-hoc ones, whereas one of the most high-volume applications is the database transaction processing, which is thought to be not suitable for cloud computing. The major reason for this unsuitability is "data integrity", which traditional database and transaction systems have regarded as the most important issue. In these systems, the principle for the data integrity is called "ACID", standing for *Atomicity*, *Consistency*, *Isolation*, and *Durability* (Weikum and Vossen, 2001). This principle requires fully isolated operations on databases for data integrity.

On the other hand, cloud computing adopts more relaxed principle for data integrity, being referred to as BASE, standing for *Basically Available*, *Soft state*, and *Eventually consistent* (Pritchett, 2008). In order to deploy database transaction systems over cloud computing environments, we first have to reveal how the characteristics of the BASE principle affect the data integrity in transaction processing. Since the BASE principle allows non-isolated concurrent access to databases, the behavior of BASE transactions becomes much more complicated than traditional ACID ones. Therefore, it seems unrealistic to evaluate the data integrity logically, considering all the possible data access patterns. Instead, it seems realistic to evaluate it through simulation using accurate BASE transaction models.

The simulation model must represent the functionality and behavior of both the cloud platform and application programs for the transactions. While the platform is common among diverse application domains, each application program is unique to a specific domain. In order to make the model comprehensive to deal with many kinds of different domains, it should be built in high modularity, so that the domain unique parts can easily be plugged in or out.

This paper proposes a simulation based evaluation model for data integrity in cloud transaction processing. For the modeling and simulation, Colored Petri Net (CPN) is used, since it can express complicated concurrent systems from behavioral, functional, and structural viewpoints simultaneously with high modularity. The rest of the paper is organized as follows. In section 2, we discuss the characteristics of the behavior of the BASE transactions, along with the concept of data integrity in generalized transaction processing. In addition, CPN based evaluation framework is introduced. Section 3 presents how the BASE transaction processing platform and application logic are modeled and integrated using CPN. Section 4 shows the mechanism and algorithm for the evaluation of the data integrity in the above CPN models.

2 CLOUD TRANSACTION MODEL FOR DATA INTEGRITY

In the cloud environments, data integrity shows

Nishida S. and Shinkawa Y..
A Comprehensive Evaluation Model for BASE Transaction Processing.
DOI: 10.5220/0005099703930400
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 393-400
ISBN: 978-989-758-036-9

the different facets from traditional database transaction processing. Before discussing the integrity evaluation, we first reveal the characteristics of the cloud regarding data integrity

2.1 Data Integrity in the Cloud

Data integrity is one of the most important issues in traditional database transaction processing, or transaction processing in short. The ACID principle assures the isolated access to databases for each transaction, in order to prevent unintentional concurrent updates on the same entry in a database, which might break the data integrity.

This rigorous control for the data integrity often reduces the performance and throughput of the transaction processing, because of the locking and serialization overhead. On the other hand, cloud computing is aiming for rather high availability and scalability, and therefore "ACID" principle is regarded as conflicting with the goal of cloud computing.

In order to adapt transaction processing to cloud computing, a more relaxed principle for data integrity is proposed, which is often referred to as "BASE". Following the BASE principle, transactions in the cloud environments show the different behavior from traditional ones, because of its unique integrity preservation mechanism.

Therefore, if we plan to run transactions in cloud environments, we need to understand their behavioral characteristics from the data integrity viewpoint. In order to understand it rigorously, we should formalize the concept of *data integrity*, along with its preservation mechanism. The data integrity is mainly an application oriented matter, and could be differently defined among application domains. Therefore, there seems no way to give the common definition of it. Instead, it seems more practical to define the standardized notation for data integrity rules or constraints.

One of the rigorous ways to express these constraints is to use the predicate logic formulae (Shinkawa and Matsumoto, 2001)(Shinkawa, 2012). Since the logic formulae for data integrity define the constraints on database values, the domain of discourse is composed of

$$\mathcal{D} = (\bigcup_i D_i) \cup (\bigcup_{i,j} r_{ij}) \cup (\bigcup_{i,j,k} a_{i,j}^{(k)})$$

where D_i is the i-th database, r_{ij} is the $j-$th entry or record in the database D_i, and $a_{ij}^{(k)}$ is the k-th attribute of the r_{ij}.

In addition to this domain of discourse, we have to define the functions, predicates, variables, and constants rigorously. Assuming we have defined all these elements of the logic, any constraints on the databases can be represented by a prenex conjunctive normal form (PCNF) as

$$Q_1 \cdots Q_n (\bigvee_j \bigwedge_i P_{ij}(t_1^{(ij)} \cdots t_{m_{ij}}^{(ij)}))$$

where Q_i is a variable with the quantifier "$\forall$" or "$\exists$", e.g. $\forall x_i$ or $\exists x_i$, P_{ij} is a predicate, and $t_k^{(ij)}$ is a term composed of variables, constants, and functions (Schoening, 2008).

On the other hand, the mechanism for preserving data integrity in the cloud is an implementation of the BASE principle, and is equivalent to the optimistic locking (Kung and Robinson, 1981). This mechanism allows arbitrary concurrent access to any entry in the databases, and the integrity preservation is attempted only at the commit point, examining whether the referred entries have been modified during the transaction execution.

The concurrent database access makes the behavior of the transactions more complicated than the ACID ones, since all the database references and updates might be interleaved between the transactions. In order to evaluate the data integrity for such a complicated transaction behavior, a simulation approach is more suitable than logical analysis.

2.2 CPN Based Integrity Evaluation Model

For effective and efficient simulation, we need to build an executable model reflecting the behavior and functionality of all the related transactions, along with the cloud platform structure including databases. Therefore, we have to select a modeling tool having the capability for expressing the three orthogonal aspects of a system simultaneously, namely, the functional, behavioral, and structural aspects. In addition, the created models must be executable for simulation. For these requirements, Colored Petri Net (CPN) is one of the most suitable modeling tools, since it extends Petri Net from functional viewpoint, which can express the behavior and the structure of systems precisely (Jensen and Kristensen, 2009) (Jensen et al., 2007).

CPN is formally defined as a nine-tuple *CPN=(P, T, A, Σ, V, C, G, E, I)* , where

P : a finite set of places.
T : a finite set of transitions.
(a transition represents an event)
A : a finite set of arcs $P \cap T = P \cap A = T \cap A = \emptyset$.
Σ : a finite set of non-empty color sets.
(a color represents a data type)

V : a finite set of typed variables.

C : a color function $P \rightarrow \Sigma$.

G : a guard function $T \rightarrow$ expression.
(a guard controls the execution of a transition)

E : an arc expression function $A \rightarrow$ expression.

I : an initialization function : $P \rightarrow$ closed expression.

In order to make the model comprehensive and reusable, modularity is the key to build it. In this paper, we focus on the two types of modularity in building the model. One is the modularity in the evaluation process, and the other is that in transaction execution.

The first modularity is implemented as the CPN model that is composed of serially-connected three major modules as shown in Figure 1. In this figure, each double-lined rectangle represents a module in the form of a CPN hierarchical substitution transition including the detailed CPN model inside. The second modularity is implemented by separating the application unique part from the common functional part. This modularity is discussed in the next section.

In Figure 1, the "INIT" module initializes the whole model by marking the places with initial tokens. These tokens are generated by this module according to the tokens in the "DBIP" and "TRIP" places, which mean *Database Initialization Parameter* and *Transaction Initialization Parameter*. The "EXEC" module performs the database updates by simulating the behavior and functionality of each transaction according to the cloud platform characteristics embedded in the module. The "EVAL" module evaluates the simulation results by examining the tokens marked in the database place "DBBody". Each model represents a phase in the integrity evaluation process, and is controlled by the token in the "PC" (Phase Control) place.

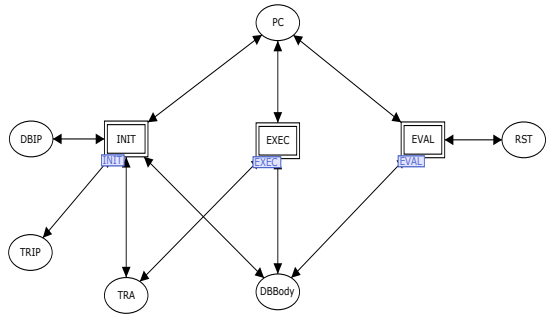

Figure 1: Top Level CPN Model.

3 CPN BASED CLOUD TRANSACTION MODEL

For the data integrity evaluation in the cloud, we first need to build a simulation model reflecting the cloud characteristics of both the platform and application. This model is designated by the "EXEC" module in Figure 1. The purpose of the "EXEC" module is to simulate the behavior and functionality of each transaction based on the cloud platform characteristics. Since transaction processing is logically composed of three major functional components, namely, "scheduling", "application logic", and "database access", we build the "EXEC" module with three functional sub-modules, namely, "Scheduling", "Application", and "Database" as shown in Figure 2.

3.1 Scheduling Sub-module

The "Scheduling" sub-module throws transactions into the system, maintaining appropriate concurrency level. In order to control the concurrency level, this module uses an integer list "COMPA", each element of which represents the thread availability. The element value of "0" means an *available thread*, while the value of "1" means a *busy thread*. This sub-module works as follows.

1. Examine the token in the place "COMPA" whether there is an available thread.
2. If there is an available thread, put a transaction token into the place "Q", extracting it from the place "TRA". Otherwise, wait until there is an available thread.
3. When one of the transaction running in the "Application" sub-module terminates, set "0" to the corresponding element of the "COMPA" token.

The "Scheduling" sub-module is common among all the application domains. The only customizable point is the COMPA token which represents the concurrency level.

3.2 Application Sub-module

The second sub-module "Application" simulates each application logic. Therefore it is application domain unique, and must be built for each application. From data integrity viewpoint, a transaction is regarded as a series of database access requests. Consequently it seems natural to express a transaction as a series of transitions, each of which generates a token for database access. However, the sequence and the content of database access requests may vary with the arguments of the transaction passed at its starting time,

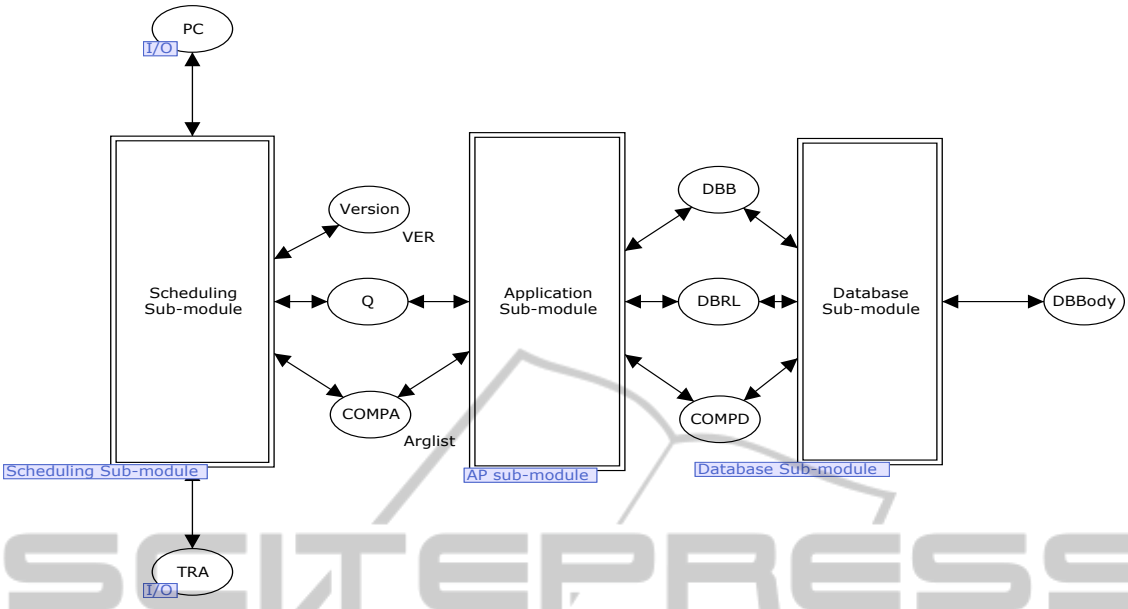

Figure 2: EXEC Modulel.

or with the database values that have been read during the execution. In addition, the sequence includes the control structures like *if-then-else* branches and *while* loops. Taking the above conditions into account, we compose the "Application" sub-module as follows.

As shown in Figure 3, the "Application" sub-module is composed of a series of transitions sharing the places "COMPD", "DBB", "DBRL", and "ARG" [1]. The place "COMPD" serializes the transition firings, along with interfacing with the "Database" sub-module. This place is associated with the color set

closet COMPD = product TID * VER * SEQ * OP * RC;

where "SEQ" designates the transition sequence number to be fired. Each transition is assigned a unique sequence number, and it examines this field by its guard function. The place "DBRL" contains a set of tokens each of which represents a list of database access request tokens "DBR" from a specific transition. The "DBR" has the color set

colset DBR = TID * VER * SEQ * OP * DBID * KEY * AttList;

where "OP" represents the operation on the database "DBID" by the key value "KEY" and the attribute values "AttList". The place "DBB" works as a *database buffer* for the database operations, and contains the accessed database entries in the form of lists. The place "ARG" contains the arguments or parameters of the transaction. These arguments determine the application logic. The place "Q" contains the *transaction tokens* with the color set

closet TRAN = product TID * VER * ArgList;

where TID is a unique transaction id, and VER is a version to express the timestamp, which will be discussed later in conjunction with *optimistic locking*.

The transitions are "L_i", "LS_i", "LE_i", "TE", and "TF". "L_i" represents a single database access operation, possibly including the *if-then-else* structure. "LS_i" and "LE_i" represent the starting and ending points of a *loop* structure respectively. "LE" and "LF" represent the *transaction end* and *finalization* processes respectively.

Using the above places and transitions, the sub-module works as follows.

1. Obtain the *transaction token* with the proper "TID" from the "Q" place.
2. The first transition set the "SEQ" field value of the token "COMPD" with "1". This value is incremented by the "Database" sub-module, so that the next transition becomes eligible to fire.
3. If the firing transition is "L_i" or "LS_i", a database access token "DBR" is appended to the "DBRL" token.
4. If the firing token is "LE_i", it sets the "SEQ" field value of the "COMPD" token with either the sequence number of "LS_i" or "++SEQ", according to the *loop* condition.
5. If the firing transition is "TE", perform the *end-of-transaction* process. Usually, it depends on the application requirements, e.g. deleting some work

[1] In order to avoid crossing arcs, the figure uses *fusion places*. The places with the same tag designate the same single place.

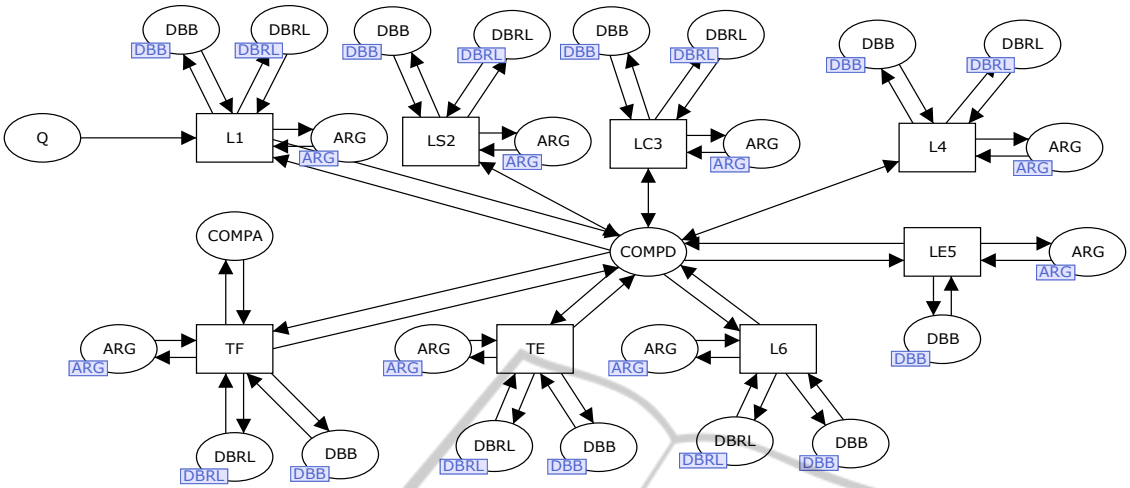

Figure 3: Application Sub-module.

entries in the databases.

6. If the firing transition is “TF”, perform the *finalization* process of the transaction. It deletes the related elements in the “ARG”, “COMPD”, and “DBRL” tokens. In addition, reset a *busy* entry of the “COMPD” token to “0”, that is, to *available*.

Since this sub-module represents a specific application logic, a unique instance must be built for each application. The structure of each instance is basically the same. Each instance is assigned an unique “TID” (a transaction id).

3.3 Database Sub-module

Each time a transition in the “Application” sub-module marks a token for the database access request, “Database” sub-module is activated to process it. In order to simulate the concurrent access to the databases, this sub-module consists of multiple instances for each database. All the instances have the same structure as shown in Figure 4, and work as follows.

1. The first transition examines the value of the field “DBID” in the “DBR” token to determine whether the request is directed toward this instance.
2. According to the value of the “OP” field in DBR, one of the conflicting transitions “READ”, “WRITE”, or “COMMIT” is fired, which performs database read, write or commit operation respectively. The value of “OP” field is defined as “1” (select), “2” (update), “3” (insert), “4” (delete), or “5”(commit).
3. The “READ” transition of the *i*th instance searches for the token with the given “KEY” in the “DBBody$_i$” place, which consists of a list of database entries. If found, append the database entry to the list token in “DBB” place, otherwise do nothing. The DBB place works as a database buffer for each transaction thread. Then set the operation result code in the “RC” field of “COMPD” field.
4. The “WRITE” transition also search the place “DBBody$_i$” for the designated key in the same way as “READ”. If found, in case of “OP = 2” or “OP = 4”, that is, for the update or delete operation, append the DBR token to the list in the “DBB” place. On the other hand, in case of “OP = 3”, that is, for the insert operation, do nothing. Then set the result code in the same way as “READ”. If not found, only the insert operation is performed.
5. The “COMMIT” transition performs the commit process that finalizes the database modification request on the database, which is implemented as the place “DBBody$_i$”. This process follows the optimistic locking mechanism stated below.

The basic operation of the optimistic locking is to ensure that “no other transactions update the database entries that have been referred to during this transaction scheduling. In order to identify transactions that update the specific database entries, we prepare a common place “WSET among the “Database sub-module instances, which includes the token representing the history of database updates by the transactions. This token is expressed as a list of each database update with the associated transaction information. The list element has the color set

colset WSETE = product TID * VER * DBID * KEY * AttList

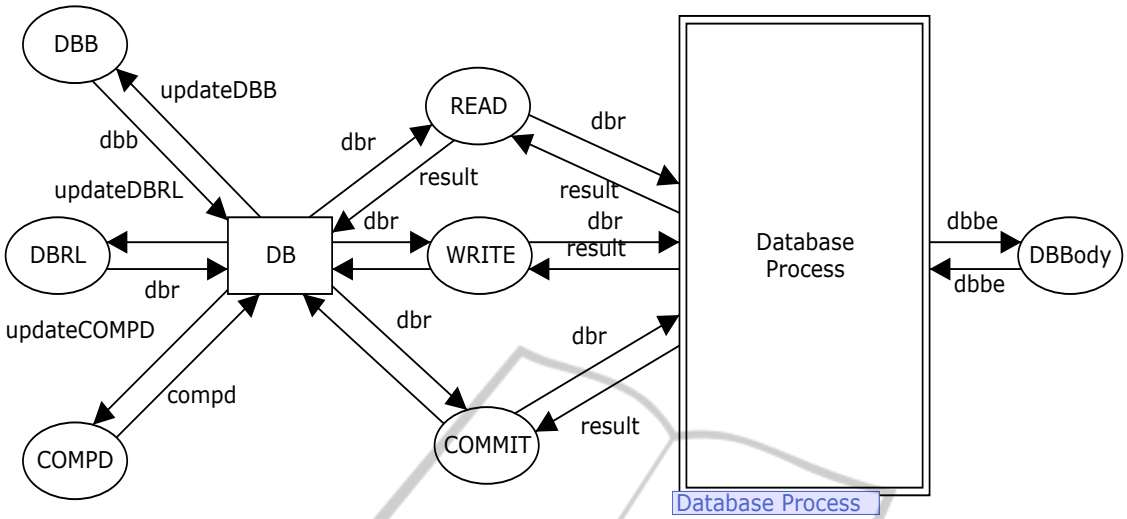

Figure 4: Database Sub-module.

This element is appended to the token after the successful comit.

The "COMMIT" process examines the "WSET" list whether it has the element that satisfies

1. The "KEY" field value is identical to that of an "DBB" element with "OP = 1", namely, an entry that has been read from a database.
2. The "VER" (version) value v satisfies $v_1 < v < v_2$, where v_1 is the "VER" value assigned to the currently committing transaction, and v_2 is the current "VER" value.

If the same key is found, the transaction is aborted, otherwise it is committed.

In case of "commit, "WSETE token is created and appended to the "WSET token, by extracting the DBB elements with the OP field of "2", "3", or "4". In this process, the "VER" value is changed to "v_2". Then the "write elements in the DBB token are moved to the "DBBody$_i$" place to reflect the database update. On the other hand, in case of "abort, all the updates in DBB token are discarded. In both cases, the COMPD token is modified to indicate that the transaction has been committed or aborted.

As discussed so far, the modularity of the "EXEC" module is a bit complicated. The module consists of three sub-modules. Two of them, namely, "Scheduling" and "Database" are common among application domains, and therefore non-customizable. On the other hand, "Application" sub-module is unique for each application, and therefore is fully customizable.

4 CPN BASED DATA INTEGRITY EVALUATION

After the simulation of the model that was built in the previous section is completed, the database tokens are updated according to the initial database state, the application logic, and the concurrency control that are performed on the actual cloud platform being considered.

In order to perform the simulation , we have to prepare the appropriate sets of databases and transactions in the form of CPN tokens. As for the databases, they should reflect the actual database characteristics, in terms of database schema and key/attribute distribution.

For simulation convenience, we assume all the keys and attributes are expressed by integers. Therefor a database record can be expressed as a tuple of integers. However, since the number of attributes is variable, they are packed into a list, and a database entry is a tuple of a key and an attribute list. As a color set, a database entry and database itself are expressed as

closet DBRec = product KEY * AttList;
closet DBBody = product DBID * DBRecL;

where "AttList" is a list of attributes, "DBRecL" is a list of "DBRec", and "DBID" represents a database identifier.

On the other hand, a transaction has a more simple form, namely, is composed of a a pair of a *transaction id* and an *argument list*. For the same reason as databases, we assume transaction id and argument list are also expressed in the form of integers.

As stated in section 3, these database and transaction tokens are generated by the "INIT" module in Figure 1.

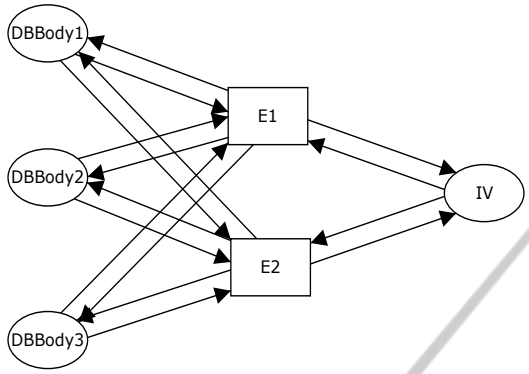

Figure 5: EVAL Module.

When generating a database token, the distribution of the key values and attribute values must be as similar as possible to the actual database to be modeled. In order to realize this requirement, the database generation transition "GENDB" is designed to receive the initialization parameters including the information on the probabilistic distribution. This information is expressed as a color set

closet GenInfo = product KEY * KEY *
MEAN * VAR * FUNC;

where the first two elements represent the key value range, "MEAN" represents the mean value of the key values, "VAR" represents the variance of the key distribution, and "FUNC" represents the distribution function that the key values follow. The "FUNC" values are currently defined as

1. uniform distribution
2. normal distribution
3. exponential distribution

, however, we can easily add a new distribution function from the CPN ML library. The arc function on the arc from the transition "GENDB" to the place "DBBody$_i$" use the above information to generate the database transitions.

As for transaction generation, we need the information on the *transaction mix* and the probabilistic distribution of the argument values. Such information can be implemented in the similar way to the databases discussed above.

After all the needed databases and transactions are generated in the form of tokens by the "INIT" module, the "EXEC" module commences to update the generated databases by simulating the behavior and functionality of the generated transactions. When all the generated transactions are committed or aborted, the "EVAL" module is initiated. This module has rather simpler structure than other two modules, as shown in Figure 5.

As stated in section 2, the data integrity is defined as a set of predicate logic formulae $\{P_i\}$. For each P_i, an evaluation transition E_i is located between the database places DBBody$_j$s and the place "IV" which holds the evaluation result token with the color set "IV". The color set "IV", standing for *integrity vector*, is a pair of integer lists $([c_1, c_2, \cdots, c_n], [r_1, r_2, \cdots, r_n])$, where n is the number of the constraint logic formulae, c_i represents the completion of integrity evaluation for the formula P_i. If $c_i = 1$, the r_i represents the evaluation result for the formula P_i. The evaluation process works as follows.

1. Mark the initial token $([0,0,\cdots,0],[0,0,\cdots,0])$ at the place "IV".
2. When the evaluation process starts, each E_i transition examines the value of c_i. If $c_i = 0$, then examine the database places DBBody$_i$s whether they satisfy the formula P_i.
3. If the P_i is NOT satisfied, then set "$r_i = 1$", else set "$r_i = 0$".
4. set "$c_i = 1$".

After all the logic formulae are evaluated, that is, "IV" token becomes $([1,,1,\cdots,1],[r_1, r_2, \cdots, c_n])$, the second element indicates whether the data integrity is preserved by the transaction executions. If $r_i = 1$ holds, it means the ith constraint formula P_i is not satisfied. In such case, the transactions must be reconsidered from the application and cloud platform viewpoints.

5 CONCLUSIONS

The BASE is a new concept for data integrity for transaction processing in cloud environments. In order to deploy newly developed or currently used transaction systems over cloud environments smoothly, we need to evaluate the systems from the data integrity viewpoint, since the BASE is too different concept from the traditional ACID. This paper introduced a CPN based BASE transaction evaluation model with high modularity and simulation capability.

In order to make the model comprehensive, the model is composed of three major modules based on the evaluation process, namely, *data generation* phase, *model execution* phase, and *integrity checking* phase. In addition, the model is divided into two parts, that is, *common functionality* part and *application unique* part.

The above model characteristics make it possible to plug-in any application model to the proposed model, or to plug-out any unnecessary one. Therefore, we can evaluate many kinds of applications to run in cloud environments at lower cost than preparing the exclusive models for them.

The data integrity in this paper is assumed to be expressed in the form of the first order predicate logic formulae, however there could be other forms of integrity constrains, e.g. those expressed in the form of modal logic like temporal logic, or those by higher order logic. For these enhancements, we need to develop more CPN ML codes to handle these constraints.

The paper focused on the generalized BASE transaction processing, which can commonly be applicable to any cloud environments. However, each environment, e.g. Google App Engine (Sanderson, 2009) or Amazon EC2 (van Vliet and Paganelli, 2011) has its own options. In order to reflect such options, more platform oriented models are needed.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number 25330094.

REFERENCES

Jensen, K. and Kristensen, L. (2009). *Coloured Petri Nets: Modeling and Validation of Concurrent Systems*. Springer-Verlag.

Jensen, K., Kristensen, L. M., and Wells, L. (2007). Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In *International Journal on Software Tools for Technology Transfer (STTT) Volume 9, Numbers 3-4*, pages 213–254. Springer-Verlag.

Kung, H. T. and Robinson, J. T. (1981). On Optimistic Method for Concurrency Control. In *ACM Transactions on Database Systems (TODS) Volume 6 Issue 2*, pages 213–226. ACM.

Pritchett, D. (2008). BASE: An ACID alternative. In *ACM QUEUE Volume 6 Issue 3*, pages 48–55. ACM.

Sanderson, D. (2009). *Programming Google App Engine*. Oreilly & Associates Inc.

Schoening, U. (2008). *Logic for Computer Scientists (Modern Birkhaeuser Classics)*. Birkhaeuser Boston.

Shinkawa, Y. (2012). CPN Based Data Integrity Evaluation for Cloud Transactions. In *Proc. 6rd International Conference on Software Paradigm Trends*, pages 267–272.

Shinkawa, Y. and Matsumoto, M. (2001). An Information System View of Consistency and Integrity in Enterprise Operations. In *Proc. 3rd International Conference on Enterprise Information Systems Volume 2*, pages 709–716.

van Vliet, J. and Paganelli, F. (2011). *Programming Amazon EC2*. Oreilly & Associates Inc.

Weikum, G. and Vossen, G. (2001). *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann.