PETRI NET MODELLING OF CONCURRENCY CONTROL IN DISTRIBUTED DATABASE SYSTEM

Djoko Haryono, Jimmy Tirtawangsa, Bayu Erfianto

***Abstract**- The life time of transaction is divided into two stages: executing stage and committing stage. At the executing stage, transaction access data through a concurrency control, while at the committing stage, a commit protocol is executed to ensure failure atomicity. A transaction that requests a lock can be blocked by a committing transaction for a long time due to a long delay in completing the committing procedure. The potential long delay in transaction commitment makes concurrency control wait until transaction finish the committing stage. This study will modify concurrency control, the modified of concurrency control allows give the locks that are still on hold by another transaction in their completion of committing stage. In modeling the concurrency control, Petri Net is used. The simulation has show increase the commit throughput of transaction, but the issue of abort transaction has significant impact to modified concurrency control, the simulation has show increase the abort throughput of transaction.*

***Keywords**: Petri Net, GSPN, Distributed Database Systems, Concurrency Control, Commit Protocol.*

I. INTRODUCTION

A transaction is considered as sequences of read and write operations on database together with computation steps [2]. A transaction can be thought of as a program with embedded database access queries. Let us first consider transaction according to their application areas. If data is distributed, the management of the transaction becomes more involved in coordinating the transactions and this may require special measures. The transactions that operate on distributed data are commonly known as distributed transactions. Data distribution offer opportunities for improving performance through parallel query execution. In order to reap the potential performance benefits, the cost of maintaining data consistency must be kept at an acceptable level in spite of added complexity of the environment.

The life time of transaction is divided into two stages: execution stage and committing stage [4]. During the execution stage, transactions access data through a concurrency control, while in the committing stage, a commit protocol is executed to ensure failure atomicity. For example, in Two Phase Locking protocol, if a transaction in executing stage requests data which is being locked by another transaction in conflicting modes, then the lock request will be blocked until the lock released. The lock of data cannot be released until the transaction completes the committing stage. A transaction that requests a lock can be blocked by a committing transaction for a long time due to a long delay in completing the commit procedure. The potential long delay in the committing stage will block the transaction that needs access to a data item. The concurrency control cannot access the data in their committing stage.

In the concurrency control area, this challenge has led to the development of a large number of concurrency control algorithms. The potential long delay in transaction commitment makes concurrency control wait until transaction finishes its committing stage. This is an important problem for the performance of transaction in distributed database systems. We present a modification of concurrency control algorithms that use the commit protocol in distributed database system as an aid to concurrency control.

In this paper, studied about how to improve the performance of distributed database systems by modifying locking based concurrency control using the concept of resource borrowing and lending from Haritsa's *et al.* committing protocol. In modeling the concurrency control, Petri Net is used.

The model assumptions are listed below:

1. The transactions have a long delay in finishing the commitment stage.
2. The transactions can by using one operation either single read or single write access a data item.

3. The operations in transaction access only one data item at one time in distributed database systems.
4. The issues of supporting real-time communication and the impact of different network issues on system performance will not be addressed.
5. It is assumed that the network has no failure condition.
6. It is assumed that the network has enough capacity to support the transmission of message.
7. It is assumed of negligible delay in communication.

II. BASIC CONCEPT

2.1 Concurrency Control

"Concurrency control in a database is the activity of coordinating the actions of transactions that operate in parallel, access shared data, and therefore potentially interfere with one another" [2]. A transaction is an atomic action. An atomic action is a group of operations that must be executed as a whole, without interference from other operations.

Locking is a mechanism commonly used to solve the problem of synchronizing access to shared data. The idea behind locking is intuitively simple. Each data item has a lock associated with it. Before a transaction T1 may access a data item, the scheduler first examines the associated lock. If no transaction holds the lock, then the scheduler obtains the lock on behalf of T1. If another transactionT2 does hold the lock, then T1 has to wait until T2 gives up the lock. That is, the scheduler will not give T1 the lock until T2 releases it. The scheduler thereby ensures that only one transaction can hold the lock at a time, soon one transaction can access the data item at a time.

2.2 Commit Protocol

Distributed database systems implement a transaction commit protocol to ensure a transaction atomicity. A variety of commit protocol have been devised, most of which are based on the classical two phase commit (2PC) protocol.

2PC protocol, as suggested by its name, operates in two phase: in the first phase, called the voting phase, the coordinator reaches a global decision (commit or abort) based on the local decisions of the participant. In the second phase, called the decision phase, the coordinator conveys this decision to the participants. For its successful execution, the protocol assumes that each participant of the transaction is able provisionally perform the actions of the transaction in such a way that they can be undone if the transaction is eventually aborted.

An optimistic 2PC-based commit protocol PROMPT (Permits Reading of Modified Prepared-data for Timeliness) is based on the assumption that a distributed transaction will not be aborted at commit time [3]. The committing transaction can lend data to other transactions so that it does not block them. In the algorithm, two situations may arise depending on the finishing times of the committing transactions.

Lender Finishes First. In this case the lending transaction receives its global decision before the borrowing transaction. If the global decision is to commit, both transactions are allowed to commit. If the decision is to abort, both transactions are aborted. The lender is naturally aborted because of the abort decision. The borrower is aborted because it has read inconsistent data.

Borrower Finishes First. In this case the borrower has reached its committing stage before the lender. The borrower is now made to wait and not allowed to send a WORKDONE messages to its coordinator. The borrower has to wait until such time as the lender receives its global decision or its own deadline expires, which ever comes earlier. In the former case, if the lender commits, the borrower is allowed to respond to the coordinator's message. In the latter case, the borrower is aborted since it has read inconsistent data.

2.3 Modeling Using Petri Net

"Petri nets are a graphical tool for the formal description of systems whose dynamics are characterized by concurrency, synchronization, mutual exclusion, and conflict, which are typical features of distributed environments" [6]. In modeling which uses concept of conditions and events, place represent conditions and transitions represent events. A transition has a certain numbers of input and output places representing pre-condition and post-conditions of event. The presence of a token in a place is interpreted as of the conditions associated with place.

The arcs of the graph are classified (with respect to transitions) as: input arcs: arrow-headed arcs from places to transitions, output arcs: arrow-headed arcs from transitions to places, inhibitor arcs: circle-headed arcs from places to transitions

Multiple (input, output, or inhibitor) arcs between places and transitions are permitted and annotated with a number specifying their multiplicities. Places can contain tokens that are drawn as black dots within places, as shown in Figure 2.6. The state of a Petri Net is called marking and is defined by the number of tokens in each place. As in classical automata theory, in Petri Net there is a notion of initial state (initial marking).

A transition is enabled if all its input places are marked at least with one token. An enabled transition may fire. If a transition fires, it destroys one token on each of its input places and creates one token on each of its output places.

A Petri net model can be formally defined in the following way [5]:

Definition 2.1 A Petri Net model is a 6-tuple = (P, T, I, O, H, PAR, PRED, MP}) where:

- P is the set of places.
- T is the set of transitions.
- I, O, H are the input arcs, output arcs, and inhibition arcs function respectively.
- PAR is set of parameters.
- PRED is a set of predicates restricting parameter ranges.

MP is the function that associates with each place either a natural number or a parameter ranging on the set of natural numbers.

Analytical models can be broadly classified into non-state and state space models, where the most commonly used state space models are Markov chains. In order to determine steady-state probabilities of a finite Markov chain, at least three different approaches for solution of the linear system are commonly considered: direct and iterative numerical methods, and a technique that yields closed form results. When real world problems are studied, Markov chains tend to become very large. Therefore it is attractive to be able to specify such systems in a compact way avoiding error-prone and tedious creation of models, and allowing designers to focus more on the system being modeled than on low-level modeling details [7]. GSPN (Generalized Stochastic Petri Net) is a prominent member of such generation models.

A GSPN is defined by a set of places, a set of transitions, relations describing pre conditions, post conditions, and inhibition conditions; and a mapping from the set of places to the natural numbers describing the model's state. The set of places represents the set of resources, local states and system variables.

The set of transitions represents the set of actions. This set is divided into two subsets: the set of immediate transitions (they are graphically represented by thick bars) that depicts a set of irrelevant actions under the performance point of view; and the subset of timed transitions (they are graphically represented by boxes) [5].

Besides, two other functions are taken into account for representing timing and priorities. The timing function associates to each timed transition a non-negative real number, depicting the respective exponential transition delay (or rate) [5]. The priority function associates to each immediate transition a natural number that represents the respective transition priority level. Transitions are fired under interleaving firing semantics, a common semantics adopted even in the untimed place/transition model. However, as defined, immediate transitions have higher priority than those timed transitions. A GSPN model is a 8-tuple = (P, T, I, O, H, PAR, PRED, MP, π, W) [6], where:

- P, T, I, O, H, PAR, PRED, MP are Petri Net model in Definition 2.1
- π : is the priority function that maps transitions onto natural numbers representing their priority level.
- W is a (possibly marking dependent) rate of a negative exponential distribution specifying the firing delay, when transition is a timed transition, or is a (possibly marking dependent) firing weight, when transition is an immediate transition.

Recall that markings in the reachability set can be classified as tangible or vanishing. A tangible marking is a marking in which (only) timed transitions are enabled. A vanishing marking is a marking in which (only) immediate transitions are enabled. A marking in which no transition is enabled is tangible. The time spent in any vanishing marking is deterministically equal to zero, while the time spent in tangible markings is positive with probability one. Similarly, a place is said to be vanishing if it can contain tokens only in vanishing markings, see the above example for GSPN.

III. MODELING AND ANALSIS CONCURRENCY CONTROL IN DISTRIBUTED DATABASES SYSTEM

3.1 Concurrency Control in Distributed Databases System

The method applied in this study was a modification of concurrency control method for distributed database systems. When transaction arrived at originating site, the transaction created a coordinator process. The coordinator process then created several participants process to access data item in several sites. In order to allow concurrent accesses of data item in conflicting modes, several participants tried to request read lock or update lock before processing the data, as shown in Figure 1.

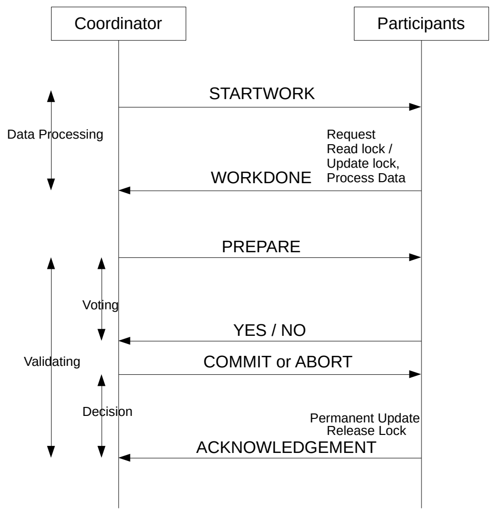


Figure 1 Data processing and Validating Phase

Let P1 be a participant holding a lock at data item X and let P2 be another participant requesting the same data item X. We have two situations may happen depending status of P1. First situation is P1 in data processing phase and second situation is P1 in validating phase.

Situation 1: P1 and P2 at data processing phase. In this situation, we have two participants conflicting in data processing phase, as shown in Figure 3.5. If there are have more than two participants that request the same data item, the rest of two participants are made to wait until concurrency control resolving the conflict between these two participants. P2 have two operations, either read mode or update mode, as shown Figure 2.

```
procedure resolvingConflict1 (participant1, participant2)
{
    if ((participant1 in data processing phase) AND
    (participant2 in dara processing phase))
    {
        if (participant2 request a read lock  on X)
        {
            if (participant1 hold an update lock)
                participant2 block
            else
                participant2 hold a read lock
        }
        else if (participant2 request an update lock on X)
        {
            if (participant1 hold an update lock)
                participant2 block
            else
            {
                restart all participants that holdind read lock or
                participant2 hold an update lock
            }
        }
    }
}
```

Figure 2 Resolving Conflicts between Data Processing Phases

Situation 2: P1 at validating phase and P2 at data processing phase. In this situation, we have conflict between P2 at data processing phase and P1 at validating phase.

We proposed the modification to our concurrency control to permits given the lock that still hold by another participant in their validating phase into new participant in their data processing phase, as shown in Figure 3.

```
procedure resolvingConflict2 (participant1, participant2)
{
    if ((participant1 in the validating phase) AND
    (participant2 in the data processing phase))
    {
        if (participant2 request read lock on X)
        {
            if ((participant1 hold a update lock) AND
            (commit dependency of participant1 is EMPTY))
            {
                add abort dependency list from participant2 to participant1
                participant2 hold a read lock on X
            }
        }
        else if (participant2 request an update lock on X)
        {
            if ((participant1 hold a read lock) AND
            (commit dependency list on participant1 is EMPTY))
            {
                add commit dependency list from participant2 to participant1
                participant2 hold an update lock on X
            }
            else if ((participant1 hold a update lock) AND
            (abort dependency of participant1 is EMPTY))
            {
                add abort dependency list from participant2 to participant1
                participant2 hold an update lock on X
            }
        }
        else
            participant2 block
    }
}
```

Figure 3 Resolving Conflict between Data Processing and Validating Phase

3.2 Petri Net Model of Concurrency Control for Single Site

In this section, describe simulation setup are used in this study. The tool for modeling Petri Net is PIPE (Platform Independent Petri Net Editor). PIPE is an open source, platform-independent tool for creating and analyzing Generalized Stochastic Petri Nets (GSPNs). PIPE offer a set of modules to carry out different types of qualitative and quantitative analysis. The available module in PIPE used in this study is GSPN analysis. This module calculates throughput of timed transitions by exploring the state space of the given Petri net and determining the steady state solution of the model. Petri Net model shown in Figure 8. The description place and transition are represented in Table 1 and Table 2.

Table 1 Description of Places

Place	Description
siteR	Participant arrives in site 1 to read the data item D1
waitRL	Participant waits to request read lock
readProcess	Participant holds read lock and accessing data item D1
waitCommitR	Participant waits for entering validating phase from their coordinator
readProcessB	Participant holds lock for read lock and read dirty data from other participant that lend their data in validating phase
borrowRAbort	Participant decides to abort because the lender global decision is abort
preparer	Participant in the validating phase
waitGlobalDecR	Participant waits for global decision from their coordinator
abortR	Participant aborts if the decision from coordinator is to abort the transaction
commitedR	Participant commits if the decision from coordinator is to commit the transaction
dataItem1	Representation available of lock for data item D1
commitPhaseR	There is participant in the validating phase for updating
siteW	Participant arrives in site 1 to update the data item D1
waitWL	Participant waits to request update lock
writeProcess	Participant holds update lock and updating temporary data item D1
writeProcessB1	Participant holds update lock, borrowing data, and updating temporary data from other participant in their validating phase
writeProcessB2	Participant holds update lock, borrowing dirty data, and updating temporary data from other participant in their validating phase
borrowWAbort	Participant decides to abort because the lender global decision is abort
waitCommitW	Participant waits for entering validating phase from their coordinator
prepareW	Participant in the validating phase
waitGlobalDecW	Participant waits for global decision from their coordinator
abortW	Participant aborts if the decision from coordinator is to abort the transaction
commitedW	Participant commits if the decision from coordinator is to commit the transaction and permanent update the data item D1
dataItem1	Representation available of lock for data item D1
commitPhaseW	There is participant in the validating phase for updating
commitPhaseR	There is participant in the validating phase for reading
tempDataItem1, tempDataItem2	The temporary place for hold the lock for updating

Table 2 Description of Transitions

Transition	Description
reqR	Participant enter data processing phase for reading data item D1
reqRL	Participant requests read lock on data item D1
borrowFromWrite	Participant borrows lock for read from lender which is still updating data process
borrowRAborting	Borrower aborts the data processing phase because the lender is abort
sendWorkdoneRB	Borrower finish data processing and sending WORKDONE message to their coordinator
borrowerGlobalDecToAR	Borrower finish data processing phase,vote abort to their coordinator, and aborting the transaction.
sendWorkdoneR	Participant finish data processing and sending WORKDONE message to their coordinator
startToCommitR	Participant enters the validating phase
voteR	Participant votes to coordinator
globalDecR	Participant waits for global decision from coordiantor
globalDecToAR	Participant chooses to commit the transaction
globalDecToCR	Participant chooses to abort the transaction
aborting	Participant aborts the transaction
commitingR	Participant commits the transaction
res1, res2	Restart the participant from the beginning and release the read lock
reqW	Participant enters data processing phase for updating data item D1
reqWL	Participant requests update lock on data item D1
borrowFromRead	Participant borrows lock for update from lender which is still reading data process
borrowWrite	Participant borrows lock for update from lender which is still updating data process
borrowWAborting	Borrower aborts the data processing phase because the lender is abort
borrowerGlobalDecToAW	Borrower finish data processing phase, vote abort to their coordinator, and aborting the transaction.
sendWorkdoneW	Participant finish data processing and sending WORKDONE message to their coordinator
sendWorkdoneW1	Borrower finish data processing and sending WORKDONE message to their coordinator
sendWorkdoneW2	Borrower finish data processing and sending WORKDONE message to their coordinator
startToCommitW	Participant enters the validating phase
voteW	Participant votes to coordinator
globalDecW	Participant waits for global decision from coordiantor
globalDecToAW	Participant chooses to commit the transaction
globalDecToCW	Participant chooses to abort the transaction
aborting	Participant aborts the transaction
CommitingW	Participant committs the transaction
Release	Participant which finished their validating phase will release lock that have been for update

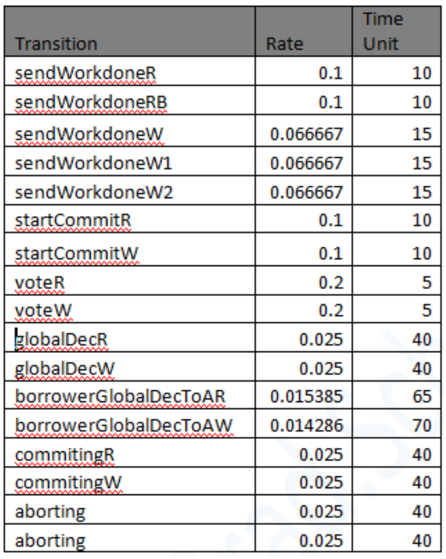
In this study, the concurrency control in distributed database system was modeled in a single site. The distributed database system accessed by the number of different transaction arrived in this site.

In this study, transaction divided by each operation, either read or update. In Petri Net modeling, the number of participant process will represent by the number of token in a place. From the above Petri Net model, the number of token in place siteR and siteW representation the number of participant process to perform read or update data item. In this study, defined the number of token in each places reqR and reqW are between 1 into 5 token. The reason for using maximum 5 token in each read and update mode is the limitation of PIPE tool when we set more tokens it is increasing complexity of the graphical layout of the net as well as of its state space.

The experiments were carried out using these models for analyzing the performance behavior of concurrency control for distributed databases under different conditions and components parameterization. The chosen evaluation method was stationary simulation (steady-state). The metrics calculated were the system throughput of committing and aborting transaction of participants. Evaluated measure is the system throughput varying the number of participant process.

A Petri Net model augmented with set of rates. The rates will fill in every transition in the model. We set transitions in the Petri Net model by fill the rates of transition (obtained by inverse of the mean), as shown Table 3.

Table 1 The Specification of Transition in Petri Net Model



Transition	Rate	Time Unit
sendWorkdoneR	0.1	10
sendWorkdoneRB	0.1	10
sendWorkdoneW	0.066667	15
sendWorkdoneW1	0.066667	15
sendWorkdoneW2	0.066667	15
startCommitR	0.1	10
startCommitW	0.1	10
voteR	0.2	5
voteW	0.2	5
globalDecR	0.025	40
globalDecW	0.025	40
borrowerGlobalDecToAR	0.015385	65
borrowerGlobalDecToAW	0.014286	70
commitingR	0.025	40
commitingW	0.025	40
aborting	0.025	40
aborting	0.025	40

IV. SIMULATION RESULT

The scenario 1a illustrates a situation when the update mode occurs more often. We find the basic concurrency control for update mode when 6 participants involves into the site, the concurrency control will grant the update mode first and there is no read mode occurs. But, when we used the modified concurrency control, the participants commit for read mode every 507.6 (1/0.00197) unit time. For update mode, the basic concurrency control has reach a constant value when 6 participants involves into the site. The modified concurrency control has a trend of committing throughput greater than the basic concurrency control, as shown in Figure 4.

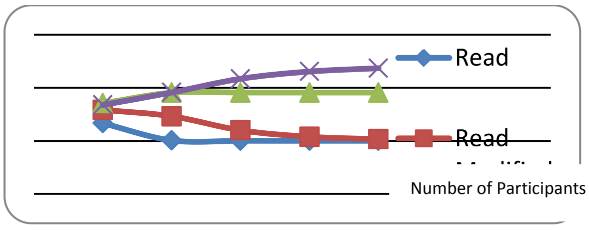


Figure 4 Throughput of Commit Transaction from Scenario 1a

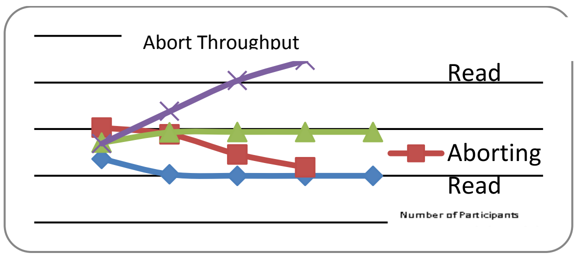


Figure 5 Throughput of Abort Transaction from Scenario 1b

The scenario 2a illustrated a situation when the read mode occur more often than the update mode. It has a trend of the committing throughput greater than the basic concurrency control. Our modified concurrency control can improve the throughput for commit of participants under the read mode or the update mode. It is happened when the read mode occur more often the modified concurrency control allows the new participants hold the lock and process the data. If the site has an update mode in the committing stage, the modified concurrency control will allow the new participants hold the lock and process data, as shown in Figure 6.

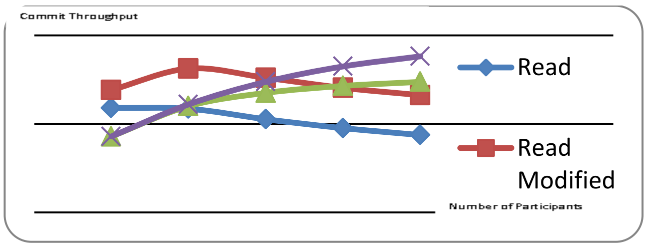


Figure 6 Throughput of Commit Transaction from Scenario 2a

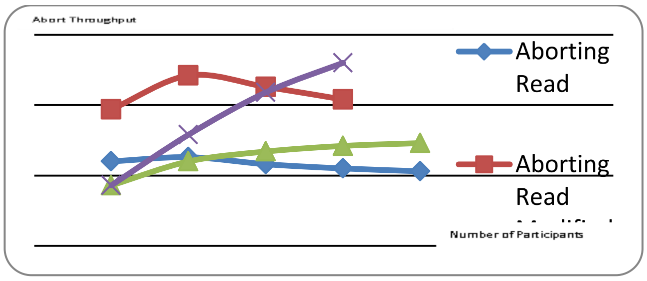


Figure 7 Throughput of Abort Transaction from Scenario 2b

The scenario 1b and 2b, we find the abort throughput of modified concurrency control for the read mode and the update mode are greater than the commit throughput of basic concurrency control. It shown our concurrency control has abort decision often occur than the basic concurrency control. This is happen because the borrower depends on the decision of the lender, as shown in Figure 5 and 7.

V. CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The simulation has shown increase the throughput of committing transaction, but the issue of the aborted transaction has significant impact to our concurrency control. This happened in situation where the aborting transactions occur more often because the borrowers depended to their lender.

5.2 Recommendations

Our performance studies are based on the assumption that there is no replication. Hence, a study of relative performance of the topic discussed here deserves a further look under assumption of replicated distributed database system. There is need for modeling Petri Net for multiple sites. The multiple sits will shown the performance of the concurrency control for distributed database system on multiple sites.

VI. REFERENCES

[1] P.A. Bernstein and N.Goodman, “Concurrency Control in Distributed Database Systems”, Addison-Wesley, U.S.A., 1981.

[2] M. Tamer Ozsu and Patrick Valduriez, “Principles of Distributed Database Systems”, Prentice Hall, U.S.A., 1991.

[3] J. R. Haritsa, K. Ramamritham, and R. Gupta, “The PROMPT real time commit protocol,” IEEE Transaction on Parallel and Distributed Systems, Vol. 11, No. 2, pp.160–181, 2000.

[4] Y. Lam, C.L. Pang, S. H. Son, and J. Cao, “Resolving executing-committing conflicts in distributed real-time database systems”, Computer Journal, Vol. 42, No. 8, pp.674–692, 1999.

[5] James L. Peterson, “Petri Nets”, Computing Surveys, Vol 9, No. 3, U.S.A, 1977.

[6] M. Ajone Marsan, G. Balbo, S. Donatelli, G. Franceschinis, “Modeling with Generalized Stochastic Petri Nets”, Jhon Willey & Sons, England, 1995.

[7] Roberta A. A. Fagundes, Paulo R. M. Maciel, and Nelson S. Rosa, “Performance Eveluation of CORBA Concurrency Control Service Using Stochastic Petri Nets”, RITA, Vol 14, 2007.

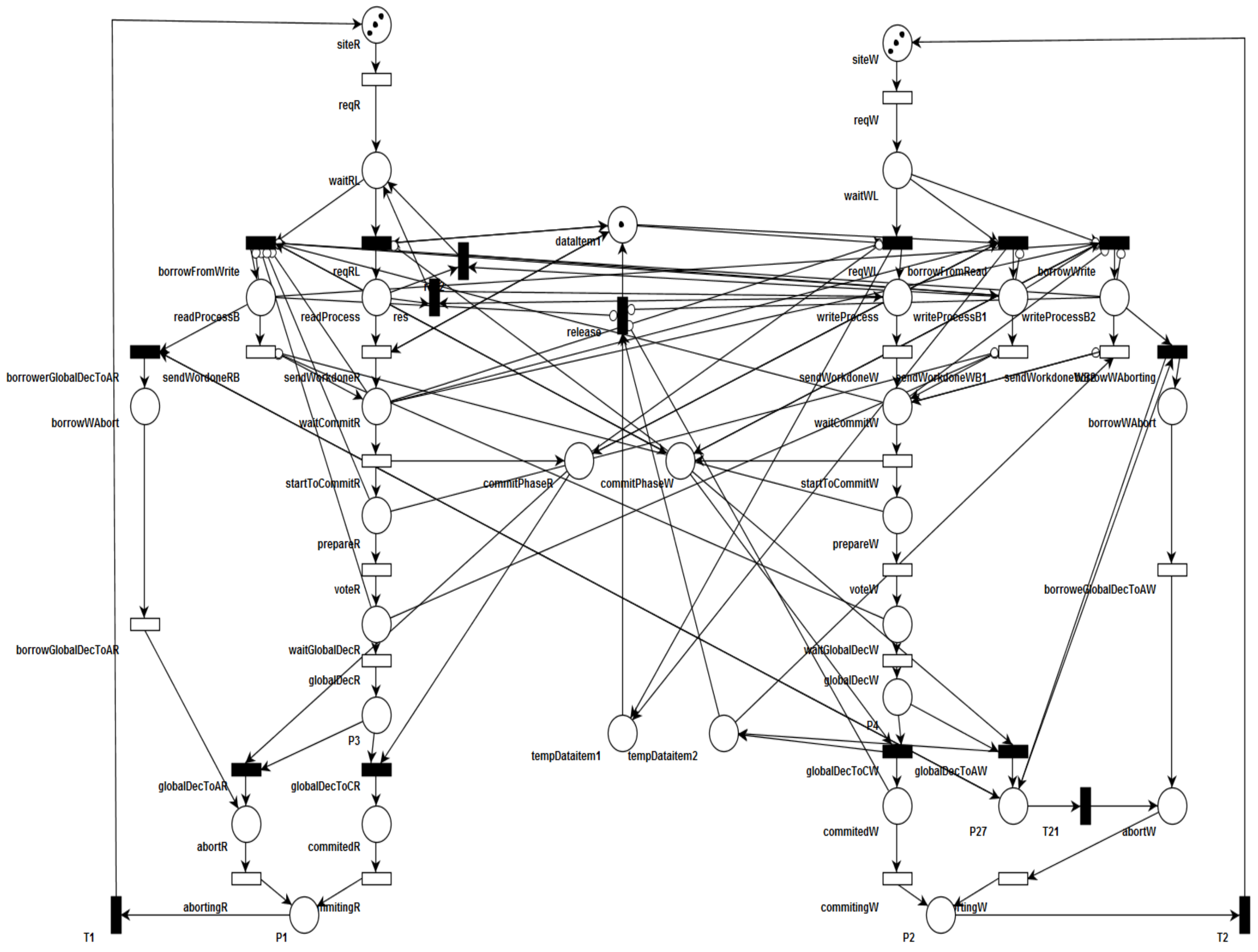


Figure 8 Petri Net Model of Modified Concurrency Control in Distributed Database System