A TRANSACTION MODEL FOR LONG RUNNING BUSINESS PROCESSES*

Jinling Wang, Beihong Jin, Jing Li
Institute of Software, Chinese Academy of Sciences, Beijing, China

Keywords: Long duration transaction, Extended transaction model, Transaction processing middleware

Abstract: Many business processes in the enterprise applications are both long running and transactional in nature, but currently no transaction model can provide full transaction support for such long running business processes. In this paper, we proposed a new transaction model — PP/T model. It can provide full transaction support for the long running business processes, so that application developers can focus on the business logic, with the underlying platform providing the required transactional semantics. Simulation results show that the model has good performance in processing the long running business processes.

1 INTRODUCTION

In many enterprise applications (such as the mortgage processing system and the insurance system), a business process can run for several hours or even longer. At the same time, these long running business processes (LRBP) should have the same transactional properties as the short running business processes. The existing transaction processing (TP) systems (such as TP Monitor and DBMS) mainly serve business processes that last for a very short time (e.g., several milliseconds), and will cause serious performance degradation if applied to such long running business processes. Therefore, we need a new type of transaction model to support the long running business processes. It should meet the following requirement:

a) It can ensure the long running business processes have the same transactional properties as the short running business processes.
b) It can enable the application developers to focus on the business logic, with the underlying platform providing the required functions to support the transactional semantics.
c) It can automatically solve concurrency conflicts between LRBPs and rollback a LRBP without human's participations.
d) When a long duration transaction conflict with traditional short transactions, it can ensure the long duration transaction has higher priority, because the long duration transaction may have run for many steps and has higher cost of failure.
e) Since many applications must deal with both the long running business processes and the short running business processes, it should be built on the existing TP systems, and shouldn't require the underlying TP system to change dramatically.

People have made a lot of research on the long duration transactions and proposed numerous long duration transaction models, but none of them can fully meet the above requirements. In this paper, we proposed a new transaction mode — PP/T model that can meet the requirement listed above. It incorporates four enhancements into the standard transaction model: sub-transactions, multiple versions, the semantics of transactions and the semantic constraints on the database states. In comparison with other long duration transaction models, the main advantage of our model is that it uses a pessimistic predicate/transform (PP/T) concurrency control mechanism. The PP/T mechanism combines the predicate/transform mechanism with the semantic constraints on the database states, so that it can ensure the priority of long duration transactions when solving conflicts between traditional transactions and long duration

* This work was supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312005, the National Hi-Tech Research and Development 863 Program of China under Grant No. 2003AA115440.

transactions. Our model is then named as PP/T model.

The remainder of the paper is organized as follows. In Section 2, we discuss related work in this area. In Section 3, we introduce the concept of the PP/T model. In Section 4, we present and analyze the simulation results. Finally, in Section 5, we conclude the paper with a summary.

2 RELATED WORK

People have made a lot of research on the long duration transaction and numerous transaction models have been proposed. One of the most influential models is the saga model (Garcia-Molina and Salem, 1987). The basic idea of this model is to decompose the long duration transaction into a set of sub-transactions, which can be executed separately. Each sub-transaction is a traditional transaction. Compensating transactions are used when the long duration transaction needs to be rolled back.

The saga model doesn't provide the strict atomicity property for long duration transactions, but provide the relaxed atomicity. The saga model also weakens the isolation property; the intermediate results will be exposed to other transactions after the execution of each sub-transaction. Therefore, when we want to roll back a long duration transaction, we should not only compensate its executed sub-transactions, but also compensate all other transactions that directly or indirectly used the intermediate results of the long duration transaction. Furthermore, since the set of transactions that used the intermediate results of a long duration transaction can't be predicated in advance, the compensating transactions can hardly be developed beforehand, and the compensating process will have to rely on the human's participation in most cases.

To improve the atomicity and isolation for long duration transactions, many other long duration transaction model (Kim et al. 1984; Du and Ghanta 1987; Gaede and Taylor 1998) use a check-out/check-in concurrency control mechanism. These models are mainly used in the CAD, CASE, and SIS areas. The basic idea of the check-out/check-in mechanism is as follows: when a long duration transaction want to operation certain data in database, it first checks out the data from database into its own local data space, and performs the operation on the data in its local data space. When the whole long duration transaction finishes, it checks in all the data in its local data space into the database. During the executing period of a long duration transaction, other transactions may have changed the data in the database, so when a long duration transaction checks in its data, the data in its local data space and the data in the database should be integrated into an unified data version. The responsibility of version integration mainly relies on the human who is running the transaction. The main disadvantage of the mechanism is that it lacks formal definitions, so the correctness criteria cannot be characterized mathematically. The users are responsible for solving conflicts and correcting errors, which may be too difficult for them. Therefore, it is not suitable for many enterprise applications.

The NT/PV model (Korth and Speegle, 1994) combines the semantic knowledge of transactions with multiple version and nested transaction techniques to support the long duration transaction. The model provides strong ability to express complex interactions, and proposed a set of correctness criteria for concurrency scheduling. Since the model also uses the compensating transaction to roll back a long duration transaction, it requires each sub-transaction to have a corresponding compensating transaction, which is impracticable for many real-world applications.

The LRUOW model (Bennett et al., 2000) is one of the newly proposed long duration transaction models in recent years. It can supports two types of concurrency control mechanisms, the most noticeable one being the predicate/transform mechanism. In the model, each long duration transaction has its own data space, and the processing of a long duration transaction can be divided into two phases: the rehearsal phase and the performance phase. The rehearsal phase spans from the beginning of the transaction to the beginning of commit. In this phase, when the user requests the system to execute a sub-transaction, the system just execute it on the local data space of the long duration transaction, and the data in the database remains unchanged. The performance phase spans from the beginning of commit to the end of commit. In this phase, the system actually executes all sub-transactions on the database in a batch. Each sub-transaction is a predicate/transform pair: the predicate is the precondition of the sub-transaction, and the transform is the actual operation of the sub-transaction. When the system actually executes each sub-transaction on the database, it first checks the predicate of the sub-transaction on the current database state. If the predicate is false, then the whole long duration transaction will fail.

The key idea of the LRUOW model is to postpone the execution of the actions in a long duration transaction until the committing time of the transaction, and then execute these actions in a batch. Therefore, the long duration transaction can be converted into a traditional short transaction, so it can have the same ACID semantics as the traditional transactions. For example, in a typically Internet shopping application, after a customer chooses a product and inputs the purchasing amount, the web application just saves the ordering information in the customer's Http session rather than updating the database immediately. Only after the customer has chosen all desired products and completed the paying process, will the web application actually change the database. The customer's shopping process can be considered as a long duration transaction, in which the concurrency control mechanism is predicate/transform.

The main disadvantage of the predicate/transform mechanism is, it can't ensure the priority of long duration transaction when it conflicts with traditional short transactions, but the cost of failure of the long duration transaction is much higher than that of the short transaction. What's more, it can't notify the user of the failure of a long duration transaction timely. For example, in the preceding Internet shopping application, after the customer finishes ordering a product, another transaction may change the stock of the product to an amount lower than the customer's purchasing amount. But the customer is not aware of the change, he may continue to order other products, and can't know the failure of his shopping session until the last minutes.

To overcome the disadvantage of the predicate/transform mechanism and ensure the priority of long duration transactions, let's take a look at a business process in a banking application. Assume a customer requests the bank to buying \$1000 bonds for him. The bank will forward the request to a stockjobber, but the result of the deal can't be known until the next day. For fear that the customer draw the money away from his account in the meantime, the bank will *freeze* part of the amount in the customer's account, i.e., although the balance of the account is X, the amount that the customer can draw out is X-1000 ($X \geq 1000$). Such a constraint on the data object is like a "lock" in the semantic layer. It doesn't prohibit other transactions' access to the data object, but requires that if other transactions change the value of the data object, the new value must satisfy certain condition. Therefore, it can ensure an operation will be successfully executed in the future. Our new transaction model is based on this practice.

It should be noted that there has been various works on the semantics-based concurrency control protocols (Garcia-Molina 1983; Weikum 1991; Agrawal et al. 1993). In comparison with these works, our work focus on providing transaction support for the long running business processes, and we combines the semantic constraints on database states with the multiple version techniques and the predicate/transform techniques to form a new transaction model.

3 THE CONCEPT OF THE PP/T MODEL

3.1 The Definition of Long Duration Transaction

Our model takes the same approach as the saga model and defines the long duration transaction as a set of sub-transactions:

Definition 1. A long duration transaction is defined as the following tuple:

$$lt = (ST, \rightarrow)$$

$ST=\{st_1, st_2, \ldots, st_n\}$, it is the set of steps that comprise the long duration transaction. $\rightarrow$ is a partial order on ST that should be satisfied in the execution of the long duration transaction.

We assume the concurrency-scheduling algorithm of the underlying traditional TP system is serializable, so an execution of the long duration transaction will form a total order on ST.

An executing history of a long duration transaction can be denoted as:

$$st_1 \circ st_2 \circ \ldots \circ st_n$$

We can give the following definition for each step of a long duration transaction:

Definition 2. A step of a long duration transaction is defined as the following tuple:

$$st_i = (p_i, f_i)$$

In the definition, p_i is the precondition of st_i, and f_i is the actual operation on the database. The predicate p_i should be the necessary and sufficient condition for the success of st_i, i.e.:

$$p_i(S) \iff \text{the success of } st_i$$

In the above expression, $p_i(S)$ means the value of the predicate p_i on the database state S.

When st_i is executed alone, it can be treated as a transition on the database state. So we can define f_i as a transition function on the database state:

$$S_2 = f_i (S_1)$$

In the above expression, S_1 is the database state before the execution of st_i, and S_2 is the database state after the execution of st_i.

3.2 Multiple Versions of Database State

We use the multiple version technique to achieve the isolation of different long duration transactions when they are executed concurrently. Every long duration transaction has its own local data space, the data inside which is invisible to other transactions. In the rehearsal phase of a long duration transaction, when step st_i wants to change data in the database, it copies the data from the database to its own local data space and changes the data inside the local data space, while the data in the actual database remain unchanged. It's not until the performance phase that the operations are actually executed on the database.

We call the state of the actual database as "global database state". In the rehearsal phase, each long duration transaction can only see its own version of database state, not the global database state. The version of database state seen by a long duration transaction is the combination of the global database state and the state of the local data space of the transaction.

We use LD_i^{lt} to denote the state of the local data space at the beginning of step st_i in lt. LD_i^{lt} includes the data changed from st_1 to st_{i-1}. $LD_1^{lt}=\Phi$. We use S_i^{lt} to denote the global database state at the beginning of step st_i, and V_i^{lt} to denote the version of database state seen by lt at the beginning of step st_i. V_i^{lt} is synthesized as following:

$$V_i^{lt} = LD_i^{lt} \textit{ override } S_i^{lt}$$

The operator *override* means that if a data object exists in the first operand, its value is got from the first operand; otherwise its value is got from the second operand.

After the execution of st_i, the state of the local data space changes from LD_i^{lt} to LD_{i+1}^{lt}, but the global database state remains unchanged:

$$f_i(V_i^{lt}) = LD_{i+1}^{lt} \textit{ override } S_i^{lt}$$

In the period between the end of st_i and the beginning of st_{i+1}, other transactions may change the global database state from S_i^{lt} to S_{i+1}^{lt}. Therefore, at the beginning of st_{i+1}, the database state visible to lt (i.e., V_{i+1}^{lt}) is once again synthesized from LD_{i+1}^{lt} and S_{i+1}^{lt}.

3.3 The PP/T Concurrency Control Mechanism

For step st_i in a long duration transaction to be successfully executed on database state S, p_i must be held on S. Therefore, for a long duration transaction to be successfully committed, the database state in the performance phase should satisfy the predicate of every step, but the predicate/transform mechanism can't ensure it. To overcome the drawback of the predicate/transform mechanism, we put forward a pessimistic predicate/transform (PP/T) concurrency control mechanism.

In the PP/T mechanism, a constraint table is set up for the database, which includes all constraints that a consistent database state should satisfy. These constraints are like "semantic locks" on the data objects. When a long duration transaction finishes every step in the rehearsal phase, it put the precondition of the step into the constraint table. After that, when any transaction prepares to commit, the system firstly checks whether the new database state satisfies all constraints in the constraint table. If any constraint is not satisfied, the transaction is not allowed to commit. Therefore, in the performance phase of the long duration transaction, the preconditions of every step are satisfied, so the execution of a long duration transaction can be successfully completed. We call the original predicate/transform mechanism as optimistic predicate/transform mechanism (since it does not exert any constraints on the database state), and call the new constraint-based predicate/transform mechanism as pessimistic predicate/transform mechanism.

In the rehearsal phase of a long duration transaction, for step st_i to be successfully executed, p_i must be satisfied on the transaction's own version of database state, i.e., $p_i(V_i^{lt})$=true. Since V_i^{lt} is synthesized from LD_i^{lt} and S_i^{lt}, we can divide p_i into two parts: one part involves the data in LD_i^{lt}, denoted as p_i^{LD}, and the other part doesn't involve the data in LD_i^{lt}, denoted as p_i^S. Apparently, we can get the following conclusion:

$$p_i(V_i^{lt}) \Rightarrow p_i^S(S_i^{lt})$$

Therefore, we can add p_i^S into the constraint table of the database, so that later transactions will not violate the constraints.

However, we can't conclude $p_i^{LD}(S_i^{lt})$=true from $p_i^{LD}(V_i^{lt})$=true, so we have to deal with p_i^{LD} specially. Agrawal et al. (1993) has proposed the idea of a wp function that is helpful to solving this problem. Following is the definition of the wp function:

Definition 3. The function wp(st_1, p_2) is the weakest condition that should be held on the database state before the execution of st_1, so that predicate p_2 is true after the execution of st_1. I.e.:

$$wp(st_1, p_2)(S) \Longleftrightarrow p_2(f_1(S))$$

According to Definition 3, we can get the following conclusion:

$$wp(st_1 \circ \ldots \circ st_{i-1}, p_i^{LD})(S_1^{lt}) \Longleftrightarrow p_i^{LD}(V_i^{lt})$$

The predicate wp($st_1 \circ \ldots \circ st_{i-1}$, p_i^{LD}) is the weakest condition that should be held on S_1^{lt} to ensure $p_i^{LD}(V_i^{lt})$ = true. Therefore, we can firstly check whether predicate wp($st_1 \circ \ldots \circ st_{i-1}$, p_i^{LD}) is true on the current global database state S_i^{lt}. If it is not true, then step st_i can't be successfully executed, and the system will return an error message. If the predicate is true, the system will put the predicate into the constraint table, so that later transactions will not violate the constraints.

However, the process of computing wp($st_1 \circ \ldots \circ st_{i-1}$, p_i^{LD}) from p_i^{LD} may be very difficult (sometimes even impossible) in practice. Therefore, we can weaken the PP/T mechanism to just recording p_i^S in the constraint table, and don't deal with p_i^{LD}. In such case, the PP/T mechanism is still an improvement over the original predicate/transform mechanism, but it can't strictly ensure the successful execution of the long duration transaction in the performance phase.

3.4 The Behaviour of the Transaction Manager

For a traditional TP system to support the PP/T model, we can add a new component called "long transaction manager" into the TP system. The long transaction manager takes charge of the management and execution of long duration transactions, while the traditional transaction manager takes charge of the management and execution of traditional short transaction.

In the rehearsal phase of a long duration transaction, when a user requests the system to perform step st_i, the long transaction manager takes the following actions (all these actions are encapsulated into a traditional short transaction):

1. Check whether p_i is true on the transaction's version of database state V_i^{lt}. If $p_i(V_i^{lt})$ is false, the execution fails and error messages will be returned.
2. Record sti and relevant parameters in the operation log.
3. Execute sti on the transaction's local data space. All changed data are recorded in the transaction's local data space, not in the actual database.
4. Put piS into the constraint table of the database.
5. Optionally, put wp(st1 ° … ° sti-1, piLD) into the constraint table of the database.

When the user commits a long duration transaction, the long duration transaction goes into the performance phase. In this phase, the long transaction manager takes the following actions (all these actions are encapsulated into a traditional short transaction):

1. Clear out all the constraints from the constraint table that are added by the current long duration transaction.
2. Execute every step on the actual database according to the operation log. Before the execution of every step, the predicate of the step is checked on the current database state. If the predicate is false, the performance phase of the long duration transaction will fail.
3. Delete the local data space of the long duration transaction after all steps being successfully finished.

To support the long duration transaction, the traditional transaction manager should also change a little. In the committing phase of a traditional short transaction, the traditional transaction manager should check whether the new database state satisfies all constraints in the constraint table. If all constraints are satisfied, the transaction can commit as usual, otherwise the transaction will be rolled back.

3.5 The Recovery Mechanism

In the PP/T model, the local data space of the long duration transaction is persistent, so the recovery mechanism is relatively simple. There are two kinds of the recovery mechanism: backward recovery and forward recovery. Backward recovery (i.e., rollback) means to undo the effect of executed steps of a long duration transaction when a failure occurs. Forward recovery (e.g., the recovery after the system crash) means to recover the state of a long duration transaction to the most recent state so that the transaction can be resumed.

When a failure occurs, an unfinished long duration transaction may be in one of the following states:

a) At the interval between two steps. I.e., the previous step has finished and the next step doesn't begin yet. At this stage, since all intermediate information is persistent in the local data space and the operation log, the system needn't do any work for the forward recovery. For the backward recovery, the system need to delete the local data space of the transaction, and clear out all the constraints that are added by the transaction from the constraint table.

b) At the executing process of a step in the rehearsal phase. Since the executing process is encapsulated into a traditional transaction, the recovery work of this step is done by the recovery mechanism of the underlying traditional TP system. After that, the other recovery works are the same as stated in 1).
c) At the executing process of the performance phase. Since the executing process is encapsulated into a traditional transaction, the recovery work of this phase is done by the recovery mechanism of the underlying traditional TP system. After that, the other recovery works are the same as stated in 1).

4 PERFORMANCE ANALYSIS

To evaluate the proposed PP/T model, we implemented it in a simulation environment and compared the simulation results with the LRUOW model.

4.1 Simulation Environment

In the simulation experiments, X traditional transactions and Y long duration transactions were generated and executed in 20 minutes, so that we could observer the performance of the PP/T model and the LRUOW model in a variety of simulated loads. The simulation program was developed in Java and was executed on a common desktop PC with an Intel Pentium IV CPU at 1.5GHz and 256MB RAM running Windows 2000 Professional

Our simulation scenario was a banking business system. Suppose there were N accounts in the database, the initial balance of each account being \$5000.00. Each account allowed two kinds of operations: depositing and drawing. The balance of accounts couldn't be lower than zero. If an operation would cause the balance of an account to be lower than zero, the operation would fail. The executing time of each operation was set to 5 ms.

Suppose each traditional transaction was a transferring account process, including a depositing operation on an account and a drawing operation on another account. The transferring amount was randomly generated in the zone of (0, max_amount). If one of the operations in a transaction failed, the transaction would be rolled back. The beginning time of the traditional transactions was randomly generated in the period from 0 to 20 minutes.

Suppose the traditional transaction manager used the strict 2PL scheduling protocol to manage concurrency and used the time-out mechanism to detect deadlocks. The time-out value was set to 5000ms, and the timed-out transactions were rolled back.

Suppose each long duration transaction was a kind of banking business process that needed multi-people to coordinate and spanned several minutes. For example, when a customer required the bank to draw a bank draft for him, several steps should be performed by the bank clerks, such as recording the ledger, reviewing the ledger, getting authorization from the manger, inputting the draft information, reviewing the draft information, etc. In each step, a bank clerk executed a traditional short transaction. In our experiments, we supposed that each long duration transaction was composed of 5 sub-transactions, and each sub-transaction was a transferring account business process. The duration time of each long duration transaction was set to 3 minutes, and the beginning times of its sub-transactions were randomly generated in the duration period of the long duration transaction. The beginning time of the long duration transactions was randomly generated in the period from 0 to 17 minutes.

4.2 Workload Parameters

In the simulation experiment, we use the failing rate of long duration transactions to evaluate performance of the PP/T model and the LRUOW model. It's the proportion of failed long duration transactions in all of the long duration transactions, reflecting the concurrency management ability of the long duration transaction models.

In the PP/T model and the LRUOW model, the failing rate of long duration transactions may be affected by the following factors:

a) The transferring amount. The amount was randomly generated in the zone of (0, max_amount). If the value of max_amount became larger, the drawing operations on accounts were more possible to fail, so the failing rate of long duration transactions would become larger. It influenced the probability of semantic conflicts between transactions.
b) The number of accounts, i.e., the number of data objects shared by all transactions. It influenced the probability of concurrency conflicts of reading and writing operations between traditional transactions.

c) The number of traditional transactions. It influenced the probability of concurrency conflicts of reading and writing operations between traditional transactions.
d) The number of long duration transactions. It influenced the probability of concurrency conflicts between long duration transactions.

4.3 Simulation Results

In the experiments, we took the maximum transferring amount, the number of accounts, the number of traditional transactions, and the number of long duration transactions as variables respectively, so that we could observe the failing rate of long duration transactions under different conditions. For each set of parameters, we repeatedly executed the simulation program for 30 times and use the average result as the final simulation result.

Fig. 1 shows the influence of maximum transferring amount on the failing rate of long duration transactions. In the figure, the X-axis represents the maximum transferring amount, and the Y-axis represents the failing rate of long duration transactions. The experiment parameters were as follows: the maximum transferring amount increased from \$250.00 to \$450.00, the number of accounts was 200, the number of traditional transactions was 60000, and the number of long duration transactions was 300. From the figure we can see that the failing rate under the LRUOW model increased rapidly (from 4.71% to 19.05%) with the increment of maximum transferring amount. By contrast, the failing rate under the PP/T model increased very slowly (from 1.46% to 5.82%). This phenomenon could be explained by the fact that the PP/T model makes use of the semantic constraints on database states to improve its concurrency management ability, so with the probability of semantic conflicts of transactions increasing, the advantage of PP/T model became more and more obvious.

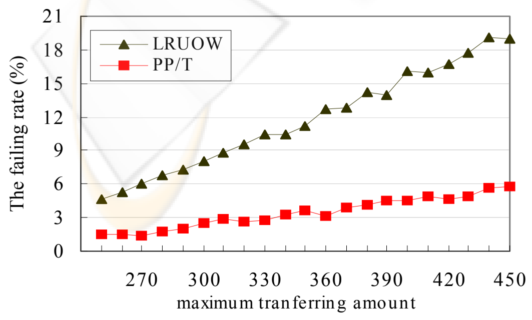

Figure 1: The influence of maximum transferring amount

Fig. 2 shows the influence of the number of accounts on the failing rate of long duration transactions. In the figure, the X-axis represents the number of accounts, and the Y-axis represents the failing rate of long duration transactions. The experiment parameters were as follows: the number of accounts decreased from 300 to 100, the maximum transferring amount was \$350.00, the number of traditional transactions was 60000, and the number of long duration transactions was 300. From the figure we can see that the failing rate increased more rapidly under the LRUOW model (from 8.13% to 17.7%) than under the PP/T model (from 2.35% to 6.6%).

Fig. 3 shows the influence of the number of the traditional transactions on the failing rate of long duration transactions. In the figure, the X-axis represents the number of the traditional transactions, and the Y-axis represents the failing rate of long duration transactions. The experiment parameters were as follows: the number of the traditional transactions increased from 50000 to 90000, the maximum transferring amount was \$350.00, the number of accounts was 200, and the number of long duration transactions was 300. From the figure we can see that the failing rate under the LRUOW model increased moderately from 10.32% to 15.01%, while the failing rate under the PP/T model increased slightly from 2.97% to 4.5%.

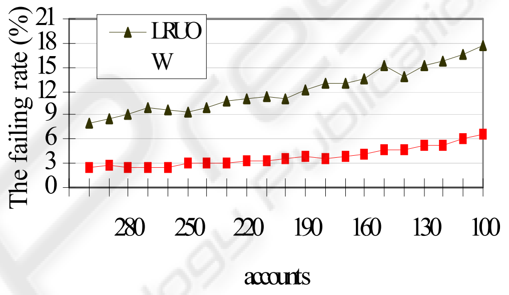

Figure 2: The influence of the number of accounts

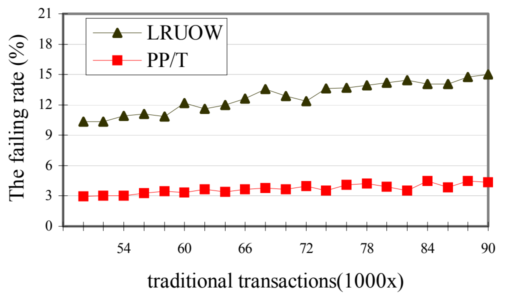

Figure 3: The influence of the number of traditional transactions

Fig. 4 shows the influence of the number of the long duration transactions on the failing rate of long duration transactions. In the figure, the X-axis represents the number of the long duration transactions, and the Y-axis represents the failing rate of long duration transactions. The experiment parameters were as follows: the number of the long duration transactions increased from 200 to 600, the maximum transferring amount was \$350.00, the number of accounts was 200, and the number of traditional transactions was 60000. From the figure we can see that the number of long duration transactions has no obvious influence on the failing rate of long duration transactions under the LRUOW model. Under the PP/T model, the failing rate increased a little with the increment of the number of long duration transactions, from 2.6% to 4.71%.

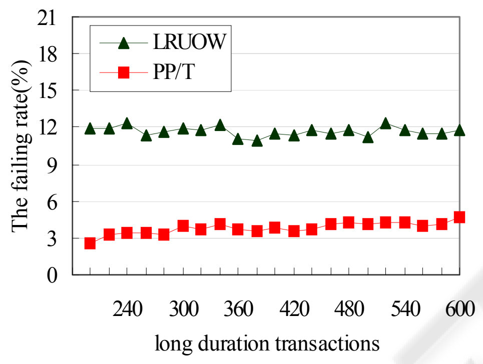

Figure 4: The influence of the number of long duration transactions

From the above simulation results we can see that the performance of the PP/T model is better than that of the LRUOW model, and the advantage of PP/T model is more obvious when there is a higher possibility of semantic conflicts between transactions.

5 CONCLUSION

In this paper, we proposed a new long duration transaction model — PP/T model. It can provide full transaction support for the long running business processes in enterprise applications. In comparison with the LRUOW model, the advantage of our model is that it can ensure the priority of long duration transactions, so that the failure rate of long duration transactions can be greatly decreased.

The key idea of the PP/T model is to postpone the actions of a long duration transaction to the committing time, so that the long duration transaction can be converted into a traditional short transaction. At the same time, constraints are exerted on the database state to ensure that the postponed operations can be successfully executed in the committing time. Simulation results show that the model has sound concurrency management ability.

REFERENCES

Garcia-Molina, H., and Salem, K., 1987. Sagas. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM press, pp. 249-259.

Kim, W., Lorie R., Mcnabb D., and Plouffe W., 1984. A transaction mechanism for engineering design databases. In *Proceedings of the 10th International Conference on Very Large Databases*. VLDB Endowment, pp. 355-362.

Du, H. C., and Ghanta, S., 1987. A Framework for Efficient IC/VLSI CAD Databases. In *Proceedings of the 13th International Conference on Very Large Databases*. VLDB Endowment, pp. 619-625.

Kuo, D., Gaede, V., and Taylor, K., 1998. Using Constraints to Manage Long Duration Transactions in Spatial Information Systems. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*. IEEE Computer Society, pp. 384-395.

Korth, H. F., and Speegle, G., 1994. Formal aspects of concurrency control in long-duration transaction systems using the NT/PV model. A*CM Transactions on Database Systems*, vol. 19, no. 3, September, pp: 492-535.

Bennett, B., Hahm, B., Leff, A., Mikalsen, T., Rasmus, K., Rayfield, J., and Rouvellou, I., 2000. A distributed object oriented framework to offer transaction support for long running business processes. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer-Verlag.

Garcia-Molina, H., 1983. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, vol. 8, no. 2, June, pp: 186-213.

Weikum G., 1991. Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems,* vol. 16, no. 1, March, pp: 132-180.

Agrawal, D., Abbadi, A. E. and Singh, A. K., 1993. Consistency and orderability: semantics-based correctness criteria for databases. *ACM Transactions on Database Systems*, vol. 18, no. 3, September, pp: 460-486.