A pipeline-based approach for long transaction processing in web service environments

Feilong Tang*

School of Software,
Shanghai Jiao Tong University,
Shanghai 200240, China
E-mail: tang-fl@cs.sjtu.edu.cn
*Corresponding author

Ilsun You

School of Information Science,
Korean Bible University,
16 Danghyun 2-gil, Nowon-gu, Seoul, South Korea
E-mail: ilsunu@gmail.com

Li Li

School of Software,
Shanghai Jiao Tong University,
Shanghai 200240, China
E-mail: lilijp@cs.sjtu.edu.cn

Cho-Li Wang

Department of Computer Science,
The University of Hong Kong, Hong Kong
E-mail: clwang@cs.hku.hk

Zixue Cheng and Song Guo

School of Computer Science and Engineering,
The University of Aizu, Fukushima 965-8580, Japan
E-mail: z-cheng@u-aizu.ac.jp
E-mail: sguo@u-aizu.ac.jp

Abstract: In web service environments, long transactions need to lock resources – often database services – for a long time during their long execution duration. This would bring down the performance of transaction processing systems. The transaction compensation is a feasible solution through allowing sub-transactions to independently commit, however, it is not able to speed up the transaction processing. This paper proposes a novel pipeline-based transaction processing (*PLbTP*) model for Serial Long Transactions (SLTs), which parallelises the transaction processing to reduce the transaction execution duration. Furthermore, we design a time-stamp-based deadlock

prevention mechanism for the control of multiple concurrent transactions. The simulation results demonstrate that our approach can significantly improve performance of SLTs without the aid of compensating transactions.

Keywords: pipeline; long transaction; compensating transaction; concurrency control.

Reference to this paper should be made as follows: Tang, F.L., You, I.S., Li, L., Wang, C-L., Cheng, Z.X. and Guo, S. (2011) 'A pipeline-based approach for long transaction processing in web service environments', *Int. J. Web and Grid Services*, Vol. 7, No. 2, pp.190–207.

Biographical notes: Feilong Tang received his PhD in Computer Science and Technology from Shanghai Jiao Tong University (SJTU), China, in 2005. From September 2007 to October 2008, he was a Visiting Researcher in the University of Aizu, Japan. Currently, he is a JSPS (The Japan Society for the Promotion of Science) research fellow in Japan. His research interests include grid and pervasive computing, wireless sensor networks, reliability computing and distributed systems.

Ilsun You received his MS and PhD in the Division of Information and Computer Science from the Dankook University, Seoul, Korea, in 1997 and 2002, respectively. He is now an Assistant Professor in the School of Information Science at the Korean Bible University. His research interests include MIPv6 security, key management, authentication and access control. He is a member of the IEEK, KIPS, KSII and IEICE.

Li Li received her MS in Computer Science from The University of Aizu, Japan, in 2005. Now, she is an Assistant Researcher in the School of Software, Shanghai Jiao Tong University, China. Her research interests include grid and pervasive computing, and distributed transaction processing.

Cho-Li Wang received his PhD in Computer Engineering from the University of Southern California in 1995. He is currently an Associate Professor with the Department of Computer Science at The University of Hong Kong. His research focuses on the system software for pervasive computing, cluster/grid computing and wireless sensor networks. He serves as an editorial board member of *IEEE Transactions on Computers, International Journal of Pervasive Computing and Communications, and Multiagent and Grid Systems*.

Zixue Cheng earned his Master's and Doctor Degrees in Engineering from the Tohoku University Japan in 1990 and 1993, respectively. He joined the University of Aizu in April 1993, and has been a Full Professor since 2002. His interests are design and implementation of protocols, distributed algorithms, distance education, ubiquitous computing, ubiquitous learning, embedded systems and functional safety. He has been the head of the Division of Computer Engineering University of Aizu, since April 2010.

Song Guo received the PhD in Computer Science from the University of Ottawa, Canada, in 2005. He is currently an Associate Professor at the School of Computer Science and Engineering, the University of Aizu, Japan. His research interests are in the areas of protocol design and performance analysis for communication networks, with a special emphasis on wireless ad hoc and sensor networks.

1 Introduction

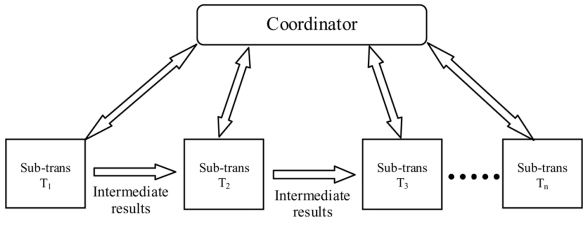
Long transactions, in general, cost a long time such as several minutes even days so that the mechanism locking accessed resources is not a good solution. A travel plan, including reserving a flight ticket and reserving a hotel, for example, is usually a long transaction because the reservation may need to check available tickets, modify databases and confirm the customer. During this period, if another reservation transaction tries to access the same database, it will be blocked. In most cases, the long resource occupation in long transactions mainly is caused by the fact that some sub-transactions have to wait for results of other sub-transactions. Furthermore, after one sub-transaction finished, it cannot commit the result immediately because it could not ensure that the whole transaction will succeed. The conservative strategy is to wait for a commit instruction from a Transaction Manager (TM). Accordingly, many resources accessed by the transaction will be locked for a long time. It is obvious that the system efficiency will be reduced.

A possible solution is the transaction compensation, which creates an associated transaction with the opposite effect for each sub-transaction in advance. In compensation-based transaction models, sub-transactions are allowed to commit independently. If the whole transaction fails for some reasons, the compensating transactions of committed sub-transactions will be executed. The compensating transactions will undo the results of the original transactions. Several protocols have been proposed for the transaction compensation (Schäfer et al., 2008). However, the compensation mechanism may not work in some cases because associated compensating transactions are very difficult even impossible to be created. On the other hand, although the transaction compensation guarantees the system consistency, it cannot speed up the transaction processing.

In this paper, we concentrate on how to reduce the execution duration of *SLTs* with the following characteristics:

- in an SLT $T = \{T_1, T_2, \ldots, T_n\}$, any sub-transaction T_i cannot start until it receives intermediate results of T_{i-1} ($2 \le i \le n$), illustrated in Figure 1
- partial or all sub-transactions in T includes a lot of operations.

Figure 1 Serial long transaction processing



As a consequence, executing an SLT needs a long time so that T locks many resources for too long period during the transaction processing unless compensating transactions are used. On the other hand, multiple concurrent SLTs possibly try to lock the same resource(s) simultaneously owing to the randomicity of transaction requests in web service systems. Accordingly, dead-lock potentially occurs from time to time.

This paper is motivated to address the above-mentioned two issues. First, we propose a *PLbTP* model, which divides each sub-transaction into a set of blocks and executes the blocks in parallel. Hence, the serial sub-transactions could be executed nearly synchronously and the compensating transactions are no longer required, which will increase the efficiency and concurrency of transaction systems. Next, to solve deadlocks under our model, we develop a distributed deadlock prevention mechanism based on a time-stamp-based victim selection that can break a deadlock cycle. Though deadlock detection is a possible method, it costs a lot of resources and time because at least one of the transactions in the deadlock cycle has to be aborted. Instead, our scheme can avoid the waste of resources.

The rest of this paper is organised as follows. Section 2 reviews related work. Section 3 presents our PLbTP model together with communication mechanism among sub-transactions. In Section 4, we propose a concurrency control approach to avoid deadlocks caused from simultaneous access to the same resource(s). Experimental results are reported in Section 5. Finally, we conclude this paper along with the discussion on our future work.

2 Related work

In this section, we review the existing work related to long transaction processing in traditional distributed systems as well as service-oriented environments.

2.1 Long transaction models in distributed systems

Most existing long-running transaction models were built on compensating transactions, firstly proposed by Gray (1981). Sagas (Garcia-Molina and Salem, 1987) is a classical long-lived transaction model and was extended to many Extended Transaction Models (ETMs) (Liang and Tripathi, 1996; Garcia-Molina et al., 1991). In Sagas, a transaction consists of a set of sub-transactions with ACID (atomicity, consistency, isolation, durability) properties, and a set of associated compensating transactions, where each sub-transaction T_i associates with a compensating transaction C_i that can semantically undo the effect caused by the commit of T_i. In Sagas, all the committed sub-transactions have to be undone if a subsequent sub-transaction fails.

ACTA (Chrysanthis and Ramamriham, 1992) is a comprehensive transaction framework that permits a transaction model to specify the effects of extended transactions on each other and on objects in databases. ACTA allows specifying interactions between transactions in terms of relationships and transactions' effects on objects' state and concurrency status. ACTA provides more powerful and flexible reasoning ability than Sagas through a series of variations to the original Sagas.

A Transaction Specification and Management Environment (TSME) (Georgakopoulos et al., 1996) is a customised transaction management system that supports implementation-independent specification of application-specific ETMs and configuration of transaction management mechanisms to enforce specified ETMs. To support ETM specification, the TSME provides a transaction specification language that describes dependencies between transactions. Flow composition languages permit the construction of long-running transactions from collections of independent, atomic

services. Because of environmental limitations, such transactions usually cannot be made to conform to standard ACID semantics. The set consistency (Fischer and Majumdar, 2007) was proposed to validate the consistency of long-running transactions. Set consistency generalises cancellation semantics, a standard consistency requirement for long-running transactions, where failed processes are responsible for undoing any partially completed work, and can express strictly stronger requirements such as mutual exclusion or dependency.

2.2 Business transaction specifications in service-oriented systems

Transaction processing in service-oriented systems presents new requirements owing to their loose coupling and autonomy (Goel et al., 2003; Lizcano et al., 2009; Tang et al., 2004). Thus, traditional long transaction models are generally not applicable for applications that comprise web-based business services (Aikebaier and Takizawa, 2009; Dalal et al., 2003). In the Business Transaction Protocol (BTP) (Ceponkus et al., 2002) and Web Services Transaction (WS-Transaction) (Cabrera et al., 2002), the use of compensating transaction for coordination of long-running activities was proposed, but no details are given on how to provide compensating transactions. To facilitate Grid users, a transaction model based on agent technologies atomic transaction and compensating transaction has been proposed (Tang et al., 2003). This model shields users from complex transaction process and provides the abilities for users to execute transaction, without knowing of process. Furthermore, CALGT (Tang et al., 2006) is a model to generate compensating transactions automatically, which frees application programmers from the complexity to provide compensating transactions.

Cost-based web services transaction management (Choudry et al., 2006) proposed monetary semantics in bookings to increase the success rate for long-running transactions, which increases the chances of success without compromising the loosely coupled autonomous nature of web services. Yahyaoui et al. (2010) looked into the coordination of web services following their acceptance to participate for service composition. They identify two types of behaviours associated with component web services: operational and control behaviours. These behaviours are used to specify composite web services that are built upon component web services. Moschoyiannis et al. (2008) focused on describing the forward behaviour of a transaction, which concerns the coordination of the underlying services and will not consider compensation mechanisms, semantics and schemas (compensating behaviour). Schäfer et al. (2008) designed a contract-based approach, which allows the specification of permitted compensations at runtime. They introduce abstract service and adapter components, which allow us to separate the compensation logic from the coordination logic. In Heinzl et al. (2010), the authors propose a temporal policy language to facilitate temporal management of structured documents. Temporal aspects can be applied to documents, such as service descriptions, or even properties in structured documents. Validity periods can be added to these properties, such that customers can easily check whether certain properties (e.g., prices) in a document are valid.

2.3 Concurrency control for distributed transactions

Researchers have proposed centralised and distributed deadlock detection algorithms (Taniar et al., 2008). To detect deadlocks introduced by highly concurrent access,

a graph-based detection algorithm was proposed in Elmagarmid (1986). Omran Bukhres compared two different Wait-For-Graph (WFG)-based algorithms (Central Controller and Distributed deadlock detection algorithms) in terms of the throughput and performance (Elmagarmid, 1986). Ezpeleta et al. (1995) proposed Petri-net-based deadlock prevention policy for flexible manufacturing systems. This approach uses a dynamic resource allocation policy to develop an online controller based on net liveness condition or the reach-ability graph of Petri net models. However, this model for deadlock avoidance is only available for FMS. Wang et al. (1995) proposed guaranteed deadlock recovery based on run-time dependency graph and incorporated it into distributed deadlock detection algorithm. Unfortunately, their designs can only support message-passing applications. In this paper, we provide efficient prevention mechanism for different types of transactions.

Though some researchers proposed probabilistic analysis method based on time-out mechanism (Hofri, 1994) to detect deadlocks in distributed systems, the timeout itself or deadlock detection time is the key factor for the transaction throughput, whatever timeout-based or graph-based detections (Dotoli et al., 2004). Also, security issue was investigated in recent years (Fuchs and Pernul, 2010; Thi and Dang, 2010). In our current design, we make full use of benefits of resource managers for distributed transactions to control concurrently resource access for deadlock avoidance. It is especially useful for independent business services, which have prior knowledge of what resources they will access.

3 Pipeline-based transaction processing

3.1 A motivated scenario

As mentioned earlier, we focus on the long transactions whose sub-transactions have to be executed serially. The essential reason is any sub-transaction takes the results of the last sub-transaction as its input parameters. Let a "goods order" transaction T consist of the following four steps:

- fill an order form (T_1)
- order the goods in the order form (T_2)
- arrange the transportation to deliver the goods to customers (T_3)
- store the transaction result (T_4).

We describe the transaction T as $T = \langle T_1, T_2, T_3, T_4 \rangle$. The second step cannot be compensated sometimes because after the order is submitted, corresponding products may have been produced. We cannot execute a compensating transaction to destroy them. For a company, this information is probably stored at different places, and the tasks are always dispatched to different departments. So, these sub-transactions will be executed at different nodes. When an order form contains a large quantity of goods, every step may cost a long time, so we can treat it as a long transaction.

In this scenario, the sub-transactions have to be executed serially. More specifically, before the order form is processed, we do not know what goods to order from the agency; before the goods are ordered, we cannot arrange the transportation to ship the goods to

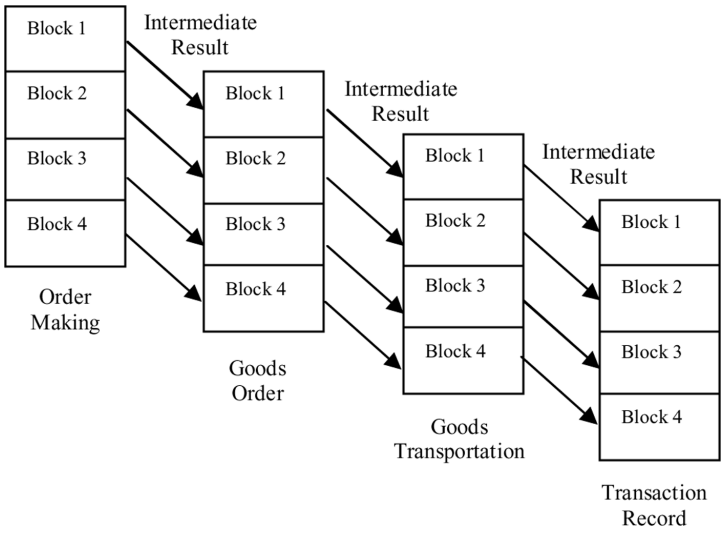
customers; after all these have been done, we are able to record the steps into a database. On the other hand, the compensation-based transaction model is not a good solution to the above-mentioned SLTs because not only many cases cannot be compensated but also it cannot reduce the execution duration. In the next section, we will present a pipeline model to speed up SLT processing.

3.2 Pipeline-based transaction processing model

The existing transaction models always treat a sub-transaction as a whole unity. Hereby, each sub-transaction in an SLT has to wait for the results of the last sub-transaction. If any sub-transaction could be divided into several blocks, however, it could partially pass the intermediate results to the succedent sub-transaction. Accordingly, the transaction processing can be speeded up. Enlightened by this idea, we use the pipeline mechanism to parallelise the SLT processing.

In the above-mentioned scenario, the first step is processing the order form, and it may take a long time to process the whole order form. As a matter of fact, the order-making service may return the partial result during the processing. Once the order-making service processes a number of goods, it sends the list and the detail information, size, colour, and so on to the ordering service. The ordering service could order these goods from the agency according to the received list. At the same time, the order-making service will process the remaining goods. Once the ordering service orders the first batch of goods from relative agencies, the transportation information, for instance, the address of the agencies, the weight of the goods and so on, will be sent to the transportation service. Then, the transportation service could arrange vehicle to ship the goods to customers. Still at the same time, the ordering service will order the second batch of goods received from order-making service. The last link of the transaction is storing all the processing information into the database, and the flow is pretty much the same thing. Figure 2 presents the procedure how to parallelise the 'goods order' transaction.

Figure 2 Pipeline-based processing for the 'goods order' transaction



The above-mentioned transaction is executed in the pipeline way, i.e., the four sub-transactions are executed almost in parallel, which will save much time if the order list is very large. When the first sub-transaction finishes the order form, it does not need to wait a long time for other services, then it is not necessary to lock the resources for a long time. Meanwhile, compensating transactions are no longer required.

Formally, our PLbTP model coordinates an SLT $T = \langle T_1, T_2, \ldots, T_n \rangle$ as follows.

- each sub-transaction T_i is divided as a set of blocks: $T_i = \langle B_{i,1}, B_{i,2}, ..., B_{i,m} \rangle$, where $B_{i,j}$ is executed prior to $B_{i,j+1}$ ($1 \le i \le n$; $1 \le j \le m$–1)
- T_i is finished if and only if all blocks in the set $\{B_{i,j} \mid 1 \le j \le m\}$ are serially executed
- n blocks $B_{i,j}$($1 \le i \le n$), from n sub-transaction, respectively, consist of a pipeline, where the execution results of $B_{i,j}$ are the input of $B_{i+1,j}$.

On the basis of the above-mentioned model, a total of m pipelines $PL_j = \{B_{i,j} | 1 \le i \le n\}$ ($1 \le j \le m$) execute in parallel in an SLT processing. Furthermore, the duration of an SLT is reduced to the lifetime of the longest pipeline PL_j in terms of the execution time.

As a matter of fact, it is very hard to divide a sub-transaction into several blocks. We have to analyse the computing property of sub-transactions and parallelise them. In most cases, it is very difficult to divide a sub-transaction into blocks. However, we can divide the intermediate results into blocks. Every sub-transaction will build an output buffer. When the sub-transaction is executed, the intermediate results will be put in the output buffer. Once the intermediate results in the output buffer are enough, they will be sent to an input buffer of the next sub-transactions, which will be described in Section 3.5.

3.3 Communication among sub-transactions

During a transaction processing, the transaction coordinator connects all the sub-transactions. For SLTs, results of a sub-transaction in general are returned to the coordinator. The coordinator, then, passes the results to the next sub-transaction. A lot of traffic is spent on the network communication.

For saving message overhead, our PLbTP model directly transfers execution results of a sub-transaction to the next sub-transaction. In fact, since the communication content between participants is simply intermediate results, the coordinator only needs to construct a data channel between the paired participants and define the transmission method, the participants could cooperate without any knowledge of other participants' interfaces.

For an SLT $T = \langle T_1, T_2, ..., T_n \rangle$, we describe a sub-transaction as a four-tuple $T_i = \{NAME_i, IN_i, OUT_i, OPS_i\}$, and suppose $OUT_i = IN_{i+1}$. In service-oriented systems, service providers publish their service interfaces in a register centre and a coordinator could enquiry the service address and interfaces from the register centre. In our MPbTP model, however, after enquiring services, the coordinator only sends a sub-transaction to a corresponding service but does not start it. Instead, the participant controls the execution of the sub-transaction. We need to add an interface for receiving data to the service, which receives the input and starts or resumes the sub-transaction. Thereby,

the coordinator should notify the previous service of the address and accessing method of the last participant. In this way, the last service may directly pass parameters to its next through this interface. As we supposed, the output of the last service is the input of its next service, so it is easy to pass parameters between each pair of neighbouring sub-transactions. The pseudo-code for data transmission among sub-transaction is shown in Figure 3.

Figure 3 The pseudo-code for data transmission directly through data channels

```
service[0] = InquiryService(t[0].name);
SendTransaction(service[0], t[0]);
foreach (t[1]..t[n])
{
        service[i]=InquiryService(t[i].name);
        SendInterface(service[i-1],
                service[i].data_interface);
}
foreach (service[0]..service[n])
        BeginTransaction(service[i]);
```

3.4 A heuristic approach for pipelining transactions

A sub-transaction running in one service often generates a series of intermediate results. To parallelise the transaction processing, each sub-transaction in our model sends its intermediate results to the next sub-transaction once the results are ready. The results are generally composed of many items and we cannot pass them one by one. Otherwise, most of time will be wasted on network transmission. On the other hand, the block should not be too large so that it has no difference from serially executing.

Considering this point, we can create an input buffer and an output buffer for each sub-transaction. When the sub-transaction receives the data from the prior one, it stores the data in the input buffer. The service fetches these input data, executes relative operations in the sub-transaction and stores the results in its output buffer. Once the intermediate results in the output buffer are beyond the threshold value, or the block size, the sub-transaction will pass the data to the next sub-transaction, as illustrated in Figure 4. It seems that the results are flowing in the pipeline-based processing system. Figure 5 is the pseudo-code for pipelining sub-transactions.

Figure 4 Pipelining sub-transactions

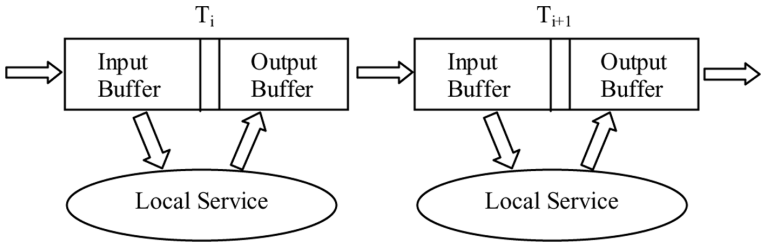


Figure 5 The pseudo-code for pipelining sub-transactions

```
while (!finishSignal)
{
      if (inputBuf.isNotEmpty)
      {
            data = inputBuf.FetchData();
            result = execSubTrans(data);
            outputBuf.store(result);
      }
      if (outputBuf.size > THRESHOD_VALUE)
      {
            sendData(nextService, outputBuf.Data();
      }
}
data = inputBuf.FetchLeftData();
result = execSubTrans(data);
outputBuf.store(result);
sendLeftData(nextService, outputBuf);
```

4 Deadlock prevention

4.1 Problem statement

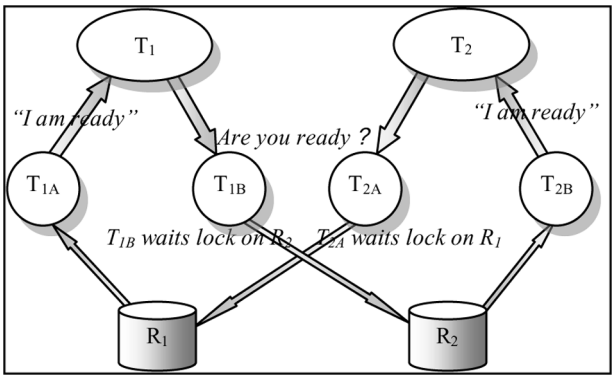
In distributed especially web service environments, transactions are often executed in different sites. Each sub-transaction in an SLT has to hold the requested resources before it actually commits, which potentially causes serious deadlocks because more than one transaction possibly requests the same resource.

Without losing generality, let two concurrent SLTs, $T_1 = \{T_{1A}, T_{1B}\}$ and $T_2 = \{T_{2A}, T_{2B}\}$, access the same resources. A deadlock occurs owing to the following resource lock request.

- T_1 asks T_{1A} and T_{1B} to prepare their sub-transactions, respectively. T_2 does the same thing on T_{2A} and T_{2B}.
- T_{1A} enters the prepare phase. It requests resource R_1 and sets its lock on R_1.
- T_{2B} enters the prepare phase too. It requests resource R_2 and sets its lock on R_2.
- T_{1B} begins to request R_2 at this time, but it must wait for the lock on R_2.
- At the same time, T_{2A} begins to request R_1. But it also needs to wait for the lock on R_1.

The above-mentioned resource request flow can be described in Figure 6, where T_1 waits for T_{1B}'s response but T_{1B} waits for T_{2B} to release the lock on resource R_2 and, on the other hand, T_{2A} waits for the T_{1A} to release R_1. As a result, a deadlock appears owing to the resource competition.

Figure 6 A deadlock scenario



4.2 Deadlock prevention mechanism

4.2.1 Resource-reservation-based deadlock prevention

Existing technologies for deadlock avoidance in general expect sequential resource access. In the most conservative case, a transaction locks all resources in advance. It is a static resource allocation algorithm that needs to exploit prior knowledge of transaction access patterns (Taniar and Goel, 2007).

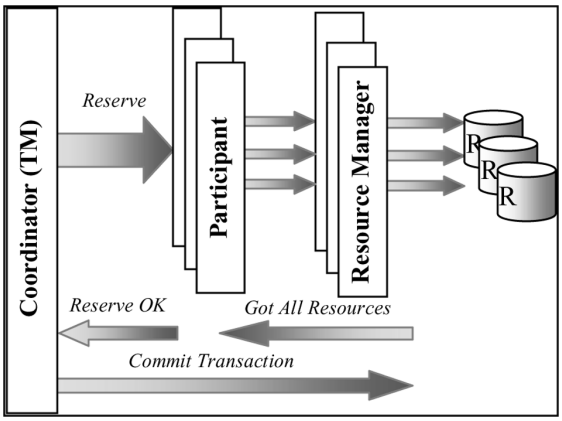
In web service environments, business transactions usually access multiple functional services distributed in different nodes. Each service knows what resources it would request. That is to say, for every transaction, it is appropriate to exploit prior knowledge of related resources. Following this idea, we add a new *resource reservation* phase before every sub-transaction actually executes. At this stage, the coordinator dispatches all sub-transactions to corresponding services, and then these services communicate with each resource manager to check the resource status. If the resources are available, the services will hold them and return OK to the coordinator. In cast that a transaction gets the resource lock, the transaction can go on. Otherwise, it will return false. After receiving all positive feedback from sub-transactions, the coordinator will start actually handling and committing the sub-transactions.

Figure 7 shows the deadlock prevention mechanism, which works as follows.

- Transaction manager (TM, i.e., Coordinator) receives a transaction request and produces a global unique root transaction ID. This ID can be a function of current time to distinguish which transaction starts earlier. TM distributes sub-transactions onto different sites, which host specified web services.
- After receiving *reserve instruction*, a participant tries to get all the needed resources from the resource manager. Assume T_{1A} requests R_1 from resource manager RM_1 and T_{2A} requests R_1 from RM_1, for example, the resource manager cache their root transaction IDs. In case that they both successfully obtain the locks of required resources, sub-transaction T_{1B} starts to acquire the lock of R_2 by sending request to RM_2. RM_2 checks its cache to make sure if any transaction has the same root ID with T_{1B}. If not, it then notifies T_{1B} that it cannot access resource R_2 so that T_{1B} will not be blocked but returns *reserve failure* information to $T1$'s TM.

- If the coordinator receives positive checked messages from all participants, it sends a prepare message to each participant, and the sub-transaction starts executing. Otherwise, it should be regarded as not meeting its prerequisite to continue. So, the T_1 decides to give up, and T_{1A} releases its lock on R_1. It is free of deadlock that if some sub-transaction T_{2B} starts to request the resource R_1, it can acquire the lock successfully. This ensures that T_2 can continue its work without a deadlock.

Figure 7 Resource-reservation-based deadlock prevention



This new added *reserve phase* is beneficial to the whole execution process of a transaction besides deadlock avoidance. Although every participant reserves the needed resources, it does not need to enter in some critical region to execute transactions if precondition is not satisfied. This reservation is very quick for most transactions, especially for long transactions, which may fail during transaction preparation.

4.2.2 Time-stamp-based deadlock elimination

Though it is possible to prevent potential deadlock by releasing held resources when resource conflict is detected, a live-lock may happen if a transaction restarts again but still cannot acquire needed resources.

To solve this problem, we use time-stamp-based mechanism to choose which transaction should quit when resource competing occurs. As we mentioned before, every transaction has a unique ID by which we can recognise which transaction starts earlier. So at the resource manager, it can determine which transaction could acquire the resource by comparing their transaction IDs. A transaction with a larger ID will be aborted when a resource conflict occurs.

5 Experiment and evaluation

5.1 Experimental environments and system architecture

To validate the performance of our solution, we developed a PLbTP system through the middleware *transaction service* between the top-layer application and the bottom-layer web services, illustrated in Figure 8. We install the transaction service on each node,

which provides specified services to outside applications. In the system, we developed four services: order-making service, goods ordering service, goods shipping service and transaction recording service. They are deployed on four nodes, respectively. Each node has a 2.4 GHz CPU and 2G memory. Furthermore, we deployed a service register centre, which is based on UDDI on the fifth node and published the four services in the service register centre. The coordinator could query service interfaces through the service register centre. Then, we developed an SLT transactional application, which accesses the four services.

Our simulation system handles transactions in the following flow.

1 Coordinator accepts a long transaction (*T*) request.

2 The coordinator queries web services in the register centre according to sub-transactions in *T*.

3 The coordinator dispatches the sub-transactions to different web services through the associated participants.

4 Each participant receives a corresponding sub-transaction and analyses input and output of the sub-transaction.

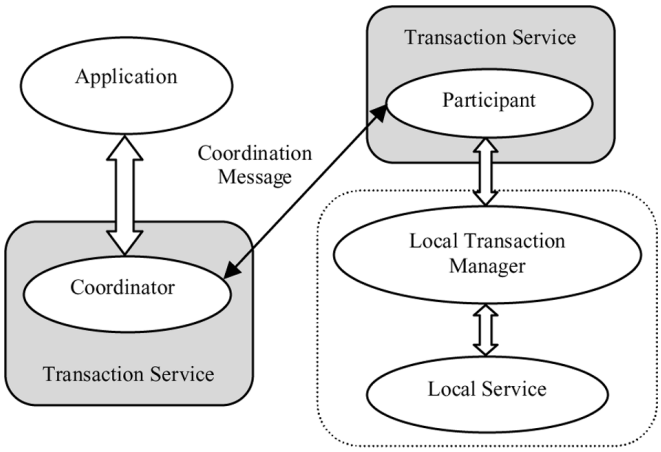
5 The coordinator constructs a data channel between paired services.

6 Each participant divides the sub-transaction as into a set of blocks in terms of the scale of the sub-transaction and its processing capacity.

7 The participant begins to execute the sub-transaction, passes the intermediate results to the next sub-transaction through the data channel and reports the status of the sub-transaction to the coordinator.

8 If all participants finish sub-transactions successfully, the coordinator sends the 'commit' command to all the participants. Once one of the sub-transactions fails, the whole transaction fails, and the coordinator will send the 'rollback' command to all the participants.

Figure 8 The system architecture



5.2 Performance evaluation

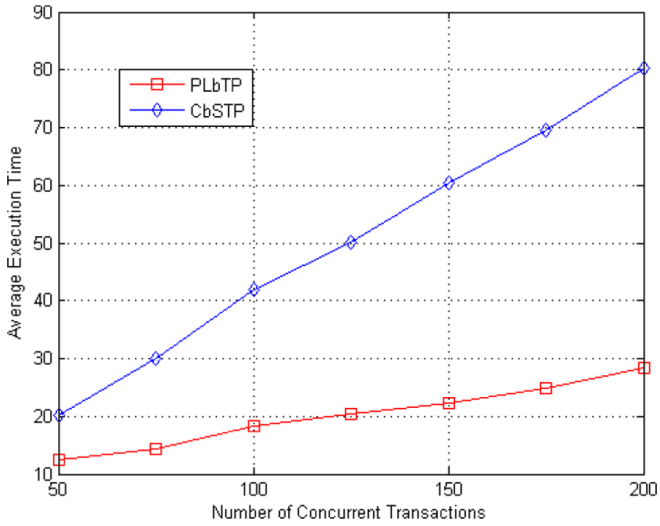
To analyse the performance improvement of our model, we evaluated our PLbTP model in different environment and compared it with the related work.

5.2.1 Average execution duration

We test *Average Execution Duration (AED)* of our *PLbTP* and the compensation-based serial transaction processing (*CbSTP*), both for SLTs. In the CbSTP model, sub-transactions in an SLT independently commit once they hold necessary resources. If a part of sub-transactions fail, the committed sub-transactions will be compensated and the global transaction is aborted. It is a classical long transaction model.

In the experiments, the AED is an average of the duration of all global transactions in a given time. The duration of a transaction is the interval from starting the transaction to committing the transaction (for successful transactions) or aborting the transaction (for failed transactions). The results are shown in Figure 9. From this figure, we can observe that the AED in CbSTP grows much faster than that in our PLbTP with increasing concurrent transactions.

Figure 9 Average execution duration against the number of concurrent transactions (see online version for colours)

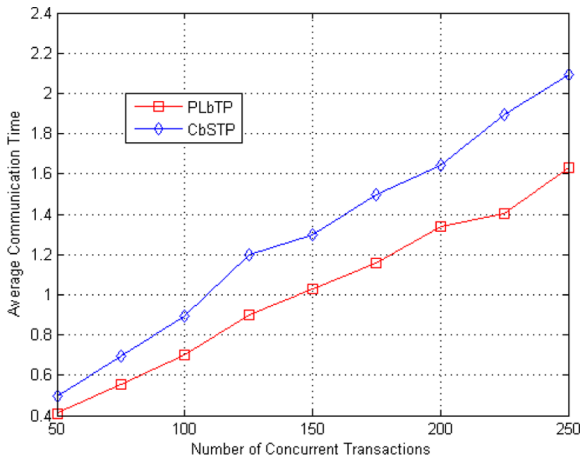


As a matter of fact, CbSTP model only provides a strategy for consistency recovery once the transaction fails. Essentially, it still processes the sub-transactions serially. Instead, our PLbTP parallelises operations in each sub-transaction, which reduces the execution time of each sub-transaction. On the other hand, when a sub-transaction fails, committed sub-transactions in the CbSTP model have to be compensated while our PLbTP only needs to rollback all sub-transactions. As we know, the average time to compensate a transaction is far more than that to roll it back. As a consequence, our PLbTP significantly reduces the execution duration.

5.2.2 Average communication time

In this experiment, we investigate how much time is spent on message communication. Figure 10 shows that *average communication time* in both the CbSTP and our PLbTP increases as more and more concurrent transactions. However, our PLbTP has a little improvement in terms of communication time. The reason is as follows. As mentioned earlier, our PLbTP builds a data channel among sub-transactions. The intermediate results are passed between sequential two sub-transactions directly. In the CbSTP model, intermediate results are passed first from a sub-transaction to the coordinator and then from the coordinator to the next sub-transaction. On the other hand, our PLbTP cannot outperform two times over CbSTP in terms of the communication time. The reason is that the CbSTP transfers all results of a sub-transaction only once while our PLbTP needs to pass the results of each transaction more times.

Figure 10 Average communication time against the number of concurrent transactions (see online version for colours)



5.2.3 Wait time ratio

Different sub-transactions in an SLT have different number of operations and accordingly need different execution time. Let $t^{\min}$ and $t^{\max}$ be the execution time of the fastest sub-transaction and the slowest sub-transaction in an SLT, respectively. We define a *Time Fluctuation (TF)* as follows.

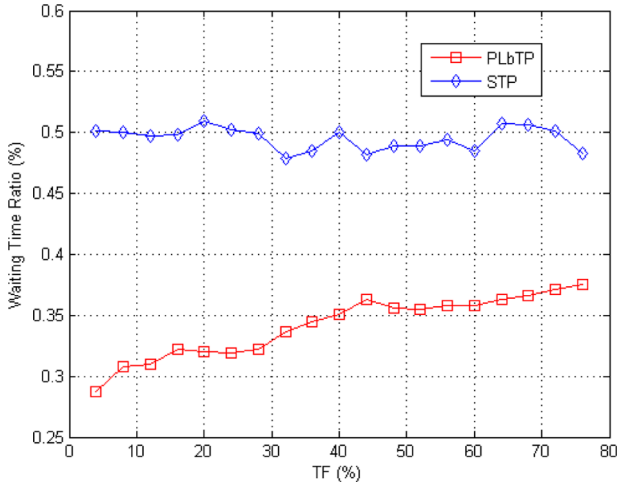
$$TF = \frac{t^{\max} - t^{\min}}{t^{\min}} \times 100\%.$$

In the PLbTP model, a fast sub-transaction has to wait for another slow sub-transaction. In this experiment, we test how the *waiting time ratio*, which is the ratio of the waiting time to the total of transaction duration, changes with the TF. Intuitively, the less the TF of a transaction is, the better the PLbTP will perform.

We simulated services, which take different time and computed the increasing of waiting time. The compensating transaction model does not need to spend time on waiting because the sub-transactions commit once they finish. So, we compared the PLbTP with Serial Transaction Processing (*STP*) strategy, which still serially

executes long transactions but without compensating function. The results are shown in Figure 11.

Figure 11 Waiting time ratio against TF (see online version for colours)



From Figure 11, we can find that as TF increases, the ratio of the time spent on waiting for other sub-transactions grows accordingly in our PLbTP. It represents that our model is more suitable for the long transactions whose sub-transactions have similar execution duration.

6 Conclusions and future work

This paper proposes a novel Pipeline based Transaction Processing (PLbTP) model for serial long transactions, which parallelises the transaction processing to reduce the execution time. This model could improve the performance of the transaction processing evidently without needing compensating transactions. Moreover, we propose a time-stamp-based deadlock prevention mechanism. The experimental results demonstrate that our LPbTP outperforms over traditional long transaction model.

As a part of our future work, we are going to investigate how to divide a sub-transaction into a set of blocks with mathematical models.

Acknowledgements

Feilong Tang thanks The Japan Society for the Promotion of Science (JSPS) for providing the excellent research environment for his JSPS Postdoctoral Fellow (ID No. P 09059) Program in The University of Aizu, Japan.

This work was supported by the National Natural Science Foundation of China (NSFC) with Grant Nos. 61073148 and 60773089, the National Science Fund for Distinguished Young Scholars with Grant Nos. 61028005 and 60725208, and Hong Kong RGC with Grant No. HKU 717909E.

References

Aikebaier, A. and Takizawa, M. (2009) 'A protocol for reliably, flexibly, and efficiently making agreement among peers', *International Journal of Web and Grid Services (IJWGS)*, Vol. 5, No. 4, pp.356–371.

Cabrera, F., Copel, G. and Coxetal, B. (2002) *Web Services Transaction (WS-Transaction)*, http://www.ibm.com/developerworks/library/ws-transpec

Ceponkus, A., Cox, W., Brown, G. and Furniss, P. (2002) *Business Transaction Protocol V1.0*, http://www.oasis-open.org/committees/download.php

Choudry, B., Bertok, P. and Cao, J. (2006) 'Cost based web services transaction management', *Int. J. Web and Grid Services (IJWGS)*, Vol. 2, No. 2, pp.198–220.

Chrysanthis, P. and Ramamriham, K. (1992) 'ACTA: the SAGA continues', *Chapter 10 of Transactions Models for Advanced Database Applications*, Morgan Kaufmann, San Francisco, CA, USA.

Dalal, S., Temel, S., Little, M., Potts, M. Webber, J. (2003) 'Coordinating business transactions on the web', *IEEE Internet Computing*, Vol. 7, No. 1, pp.30–39.

Dotoli, M., Fanti, M.P. and Iacobellis, G. (2004) 'Comparing deadlock detection and avoidance policies in automated storage and retrieval systems', *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, Netherlands, pp.1607–1612.

Elmagarmid, A.K. (1986) 'A survey of distributed deadlock detection algorithms', *ACM SIGMOD Record*, Vol. 15, No. 3, pp.37–45.

Ezpeleta, J., Colom, J.M. and Martinez, J. (1995) 'A petri net based deadlock prevention policy for flexible manufacturing systems', *IEEE Transactions on Robotics and Automation*, Vol. 11, No. 2, pp.173–184.

Fischer, J. and Majumdar, R. (2007) *Ensuring Consistency in Long Running Transactions*, UCLA Computer Science Department, Technical Report TR-070011.

Fuchs, L. and Pernul, G. (2010) 'Reducing the risk of insider misuse by revising identity management and user account data', *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, Vol. 1, No. 1, pp.14–28.

Garcia-Molina, H. and Salem, K. (1987) 'SAGAS', *Proceedings of ACM SIGMOD'87*, Vol. 16, No. 3, pp.249–259.

Garcia-Molina, H., Gawlick, D., Klein, J., Kleissner, K. and Salem, K. (1991) 'Modeling long-running activities as nested sagas', *Bulletin of the IEEE Technical Committee on Data Engineering*, Vol. 14, No. 1, pp.14–18.

Georgakopoulos, D., Hornick, M.F. and Manola, F. (1996) 'Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation', *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol. 8, No. 4, pp.630–649.

Goel, S., Sharda, H. and Taniar, D. (2003) 'Preserving data consistency in grid databases with multiple transactions', *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing (GCC 2003)*, LNCS 3033, Springer, Shanghai, China, pp.847–854.

Gray, J. (1981) 'The transaction concept: virtues and limitations', *Proc. the 7th International Conference on VLDB*, Cannes, France, pp.144–154.

Heinzl, S., Schmeling, B. and Freisleben, B. (2010) 'Using temporal policies for managing changing meta-data of web services', *International Journal of Web and Grid Services (IJWGS)*, Vol. 6, No. 4, pp.331–356.

Hofri, M. (1994) 'On timeout for global deadlock detection in decentralized database systems', *Information Processing Letters*, Vol. 51, No. 6, pp.295–302.

Liang, D. and Tripathi, S. (1996) 'Performance analysis of long-lived transaction processing systems with rollbacks and aborts', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 5, pp.802–815.

Lizcano, D., Soriano, J., Reyes, M. and Hierro, J.J. (2009) 'A user-centric approach for developing and deploying service front-ends in the future internet of services', *International Journal of Web and Grid Services (IJWGS)*, Vol. 5, No. 2, pp.155–191.

Moschoyiannis, S., Razavi, A.R., Zheng, Y. and Krause, P. (2008) 'Long-running transactions: semantics, schemas, implementation', *Proceedings of Second IEEE International Conference on Digital Ecosystems and Technologies*, Thailand, pp.20–27.

Schäfer, M., Dolog, P. and Nejdl, W. (2008) 'An environment for flexible advanced compensations of web service transactions', *ACM Transactions on the Web*, Vol. 2, No. 2.

Tang, F., Li, M. and Cao, J. (2003) 'A transaction model for grid computing', *Lecture Notes in Computer Science*, Vol. 2834, pp.382–386.

Tang, F., Li, M. and Huang, J. (2006) 'Automatic transaction compensating for reliable grid applications', *Journal of Computer Science and Technology (JCST)*, Vol. 21, No. 4, pp.529–536.

Tang, F., Li, M., Huang, J. (2004) 'Real-time transaction processing for autonomic grid applications', *Engineering Applications of Artificial Intelligence (EAAI)*, Vol. 17, No. 7, pp.799–807.

Taniar, D. and Goel, S. (2007) 'Concurrency control issues in grid databases', *Future Generation Computer Systems*, Vol. 23, No. 1, pp.154–162.

Taniar, D., Leung, C.H.C., Rahayu, W. and Goel, S. (2008) *High Performance Parallel Database Processing and Grid Databases*, John Wiley & Sons (book), USA.

Thi, Q. and Dang, T. (2010) 'Towards side-effects-free database penetration testing', *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, Vol. 1, No. 1, pp.72–85.

Wang, Y., Marritt, M. and Romanovsky, A. (1995) 'Guaranteed deadlock recovery: deadlock resolution with rollback propagation', *Proceedings of Pacific Rim International Symposium on Fault-Tolerant Systems*, USA, pp.1–21.

Yahyaoui, H., Maamar, Z. and Boukadi, K. (2010) 'A framework to coordinate web services in composition scenarios', *International Journal of Web and Grid Services (IJWGS)*, Vol. 6, No. 2, pp.95–123.