Zhao L, Yang JW. Resources snapshot model for concurrent transactions in multi-core processors. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(1): 106–118 Jan. 2013. DOI 10.1007/s11390-013-1315-7

# Resources Snapshot Model for Concurrent Transactions in Multi-Core Processors

Lei Zhao (赵 雷), *Member*, *CCF*, *ACM*, and Ji-Wen Yang (杨季文), *Member*, *CCF*, *ACM*

*School of Computer Science and Technology, Soochow University, Suzhou 215006, China*

E-mail: {zhaol, jwyang}@suda.edu.cn

Received June 1, 2011; revised August 20, 2012.

**Abstract** Transaction parallelism in database systems is an attractive way of improving transaction performance. There exists two levels of transaction parallelism, inter-transaction level and intra-transaction level. With the advent of multi-core processors, new hopes of improving transaction parallelism appear on the scene. The greatest execution efficiency of concurrent transactions comes from the lowest dependencies of them. However, the dependencies of concurrent transactions stand in the way of exploiting parallelism. In this paper, we present Resource Snapshot Model (RSM) for resource modeling in both levels. We propose a non-restarting scheduling algorithm in the inter-transaction level and a processor assignment algorithm in the intra-transaction level in terms of multi-core processors. Through these algorithms, execution performance of transaction streams will be improved in a parallel system with multiple heterogeneous processors that have different number of cores.

**Keywords** multi-core, database transaction, parallelism, concurrency, conflict detection

## 1 Introduction

Transaction is a very important concept of database systems. Since transaction processing is the guarantee for the consistency and integrity of databases, transaction models used to be an active topic in database technology area for a long period of time. Researchers have proposed many transaction models[1-5] in this period. Nevertheless, the hotspots of this area changed in recent years while a lot of technologies developed, such as system architectures, networks, operating systems, and so on. These developments lead to some applied research branches in this area.

A lot of researchers worked on the transaction processing of databases for mobile devices. This is a topic about transaction processing under unstable network connections. In [6-7], the authors focused on the efficiency and quality of service in real-time mobile databases. In [8-10], the authors talked about the specification, dependency and heterogeneity of mobile databases.

Some other researchers focused on the transaction processing for real-time databases. In [11-13], scheduling algorithms are the points. Scheduling is one of the most important parts in real-time systems. In [14-16], concurrency control is discussed. The problem of concurrency control is classic and well researched, but it exposes new problems with the development of the architecture of computer.

In recent years, parallel computers and multi-core processors are widely used. Transaction processing for parallel databases[17-19] is becoming a hot spot. Although those transaction models mentioned previously have been proved to be successful and useful for regular databases, they are not powerful enough to process concurrent transactions in advanced parallel database systems. They have at least two major drawbacks.

1) The existing transaction models are mostly made up of long running activities. It is well known that, execution of transactions must take some resources, e.g., CPU timeslices, shared memories, shared buffers and I/O channels. Long running activity means occupying some resources for a long time. This situation will lead to concurrent transactions executed sequentially in parallel database systems. The reason is the cost. Long running activities fit for loose-coupled systems. Loose-coupling systems, e.g., grids and clusters, are more cheaper than tight-coupling machines and easier to obtain. Parallel computer no longer means costliness because of the popularity of multi-core technology. Parallel computers become our desktop computers or laptops nowadays. It is time to cut the transactions to short running activities for better parallelism, which is an intra-transaction problem.

Regular Paper

This work is supported by the National Natural Science Foundation of China under Grant No. 61073061.

2) Locking and unlocking operations always take too much time in transaction processing. There will be a great progress if non-locking transaction processing can be achieved in parallel systems. Now non-locking execution for concurrent processes has been presented relying on transactional memory, a king of special storage. It means non-locking processing for database transactions is feasible. But non-locking transaction processing was never discussed in the existing work. It is an inter-transaction problem.

In this paper, we present Resource Snapshot Model (RSM) for resource dependency modeling of transactions streams in which there may exists concurrent transactions. The purpose is to set up a mechanism of transaction processing in advanced parallel systems without expensive locking and unlocking operations.

Firstly, RSM can be used on inter-transaction parallelism. Transactional memory allows non-lock processing. As a result, it causes restarting while conflicts occur. Restarting means decrease of performance. RSM can help to find a way of non-restart processing for concurrent transactions.

Secondly, RSM can also be used on intra-transaction parallelism. It is well known that, a database transaction must be atomic, consistent, isolated and durable. These are required to insure that a database is running in a safe way. This is from the point of view of DBMS, but is quite different in execution. It does not mean that a database transaction cannot be decomposed in execution. Actually, it must be divided into a series of database operations if it needs to be executed. Some of these operations may cause resources racing. If so, they cannot execute concurrently in a parallel system. But if we reschedule these conflicting operations and make them execute concurrently with other independent operations, the situation may change. That is what we have met in traditional multi-processor systems.

However, things may change while multi-core processors appear. Suppose, a multi-processors system has several multi-core processors with different number of cores. What shall we do while facing a transaction which can work in separate threads? How could we know which processor should be assigned to this transaction? If the number of cores is greater than that of threads, it will make some cores idle while make some threads wait. Both of these two situations will lead to decrease of performance. So we need a strategy to solve this problem. RSM can help.

RSM is for resource modeling in both intra-transaction level and inter-transaction level. We propose a non-restarting scheduling algorithm in inter-transaction level and a processor assignment algorithm in intra-transaction level in terms of multi-core processors. Through these algorithms, we can improve execution performance of transaction streams in a simulated parallel system with asymmetric multiple processors.

The rest of the paper is organized as follows. Section 2 presents some related work in this area. Section 3 gives some examples to show the motivations of our work. We provide some basic definitions, theorems in Section 4. Section 5 proposes some relevant algorithms. Section 6 shows some experimental results of simulations. Finally, we conclude the paper in Section 7.

## 2 Related Work

Transaction dependency is a complicated problem. Some related problems, such as transaction models, concurrency, conflicts testing and detection are all involved in. Lots of researches have been issued in this field.

Chrysanthis and Ramamrithan presented a formalism for extended transaction model[20] and introduced ACTA, a formal framework for specifying extended transaction models[2]. ACTA allows intuitive and precise specification of extended transaction models by characterizing the semantics of interactions between subtransactions in terms of different dependencies between subtransactions, and in terms of subtransaction's effects on data objects. During the past few years, ACTA has been used for specifying and reasoning about advanced transaction models. The authors proposed a synthesis form of extended transaction models using ACTA[21].

Biliris *et al.* developed a flexible transaction facility called ASSET[4]. ASSET allows the specification of arbitrary transaction models and provides support for programming transactions that have relaxed correctness requirements. The goal is to facilitate the construction of transactions that cooperate and interact in application-specific ways. Through several examples, it shows how three novel transaction primitives, namely, *delegate*, *permit* and *form_dependency*, allow the construction of arbitrary transaction models and the realization of relaxed correctness notions.

Schwarz *et al.* introduced the concept of transaction closure[22-23] as a generalization of the well-known concept of nested transactions together with a set of transaction dependencies. They thought transaction closure was a suitable framework for the design of complex applications. They discussed the execution dependency of transaction closure[22]. The execution dependencies between transactions can be expressed in terms of begin and end events associated with the corresponding transactions, and they restrict the temporal occurrence of the events. The authors presented three execution dependencies, *parallel strict overlapping*, *parallel including* and *sequential*, after combining the basic events of two interrelated transactions. Moreover, they also introduced transitive properties of execution dependencies[23]. This issue is important to get a grasp of the entire semantics of a complex application. By this way, it is possible to conclude how two arbitrary transactions are interrelated.

In [17], the authors proposed a 3-level transaction scheduling model and scheduling algorithm, PDCC, based on priority. Compared with GCC[24], PDCC uses complex priority algorithms for the transactions instead of simple timestamps. PDCC not only supports correctness of the data, but also can improve the effectiveness of the grid database system.

In [25], the authors designed an optimistic concurrency control algorithm for P2P databases, named SODA. SODA is lightweight and addresses P2P characteristics. The processing time of SODA is decreased by applying the concept of sequential order. At the same time, the transactions abort rate is reduced through dynamically adjusting the sequential order.

Deng *et al.* focused on testing database transactions[26]. They described a substantial extensions to AGENDA, allowing it to test transactions with multiple queries and with complex intended behavior. The paper introduces an improved input generation heuristic and a technique for checking complex properties of the database state transition performed by the transaction. Results of using AGENDA to test three nontrivial applications with seeded faults were presented. Actually, the authors focused on testing database transaction concurrency[27] in two years before Deng *et al.* They proposed a framework and implemented a tool set to partially automate the testing process. They identified the potential offline concurrency problems in database applications. Two approaches were suggested to execute a given schedule. Preliminary empirical evaluation based on the TPC-C benchmark was presented and demonstrated the effectiveness and efficiency.

In [28], the authors addressed the problem of dependency conflicts existing in advanced transaction models. They discussed how dependencies could affect the execution of advanced transactions, and presented the means how to analyze conflicts of dependencies. In that paper, the authors described the properties of dependencies, and studied different kinds of dependency conflicts. They also proposed algorithms to automatically detect the conflicts. The conflict free dependencies ensure that the dependencies are physically realizable in execution. It ensures that the advanced applications are free from deadlocks and unavailability issues.

Thread-level speculation (TLS)[29-31] is an important way of increasing transaction parallelism in thread level issued in 1998 firstly. Its key point is speculation which empowers the programmer to parallelize code while being concerned only with performance rather than correctness. Specifying where to break a transaction into epochs is what the programmers need to do. The TLS mechanism executes them in parallel preserving the original sequential semantics of the program. But some epochs may be restarted when their execution diverges from the original sequential execution.

Christopher and his collaborator presented their method of incrementally parallelizing transactions[32]. It provides a possible solution to the problem of parallelizing the central loop of a transaction that can reduce transaction latency and hence decrease contention for resources used by the transaction. It also provides a methodology for eliminating the data dependencies that limit parallel performance, describing three specific techniques for eliminating these dependencies and examples of their application.

However, not much work appears in the researching on transaction parallelism in asymmetric multiprocessors systems. And not much work is found on transaction processing in a database management system without locking and unlocking operations. The prime contribution of this paper is to present a nonlocking transaction processing model in asymmetric multi-processors systems.

The main differences between our work and the previous work is as follows.

1) This paper addresses parallelism of concurrent transactions not only on inter-transaction level but also on intra-transaction level.

2) The inter-transaction level scheduling presented in this paper is a non-locking scheduling, differing from most previous work.

## 3 Motivations

Suppose that we have a stream of transactions in a parallel database system which is executed on a server with asymmetric multiple processors, shown in Fig.1. Some of them may work in parallel according to the requirements of resources. Some transactions may have possibilities of inner parallelism. That is, we need to tackle a few key problems.

1) How many parallel threads does a transaction have at most?

2) How can we assign a processor with appropriate number of cores to a transaction?

3) How to schedule these transactions? That is, how can we know that a scheduling has no conflicts and all these transactions are consistent.

There are three examples demonstrating the above three questions. Fig.1 demonstrates an asymmetric multi-processor system. It is composed of processors with different number of cores. In Fig.2 and Fig.3, each row represents a core. In Fig.2, the notation $R_n^{r/w}$ means instance of critical resource $R_n$ is required, and the superscript $r$ or $w$ means *read* or *write*, respectively. And the notation $R_n^r \to R_m^w$ will be presented in Definition 6.

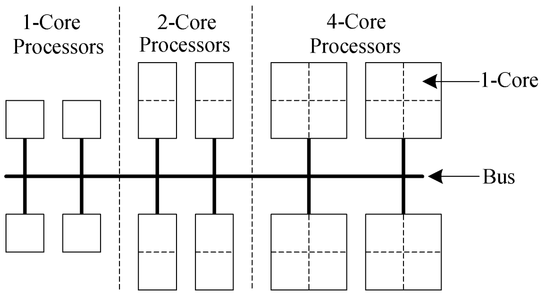
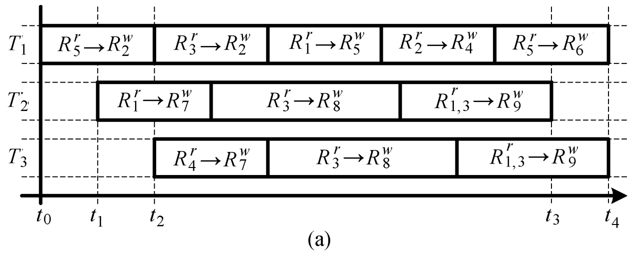


Fig.1. Asymmetric multi-processor system.



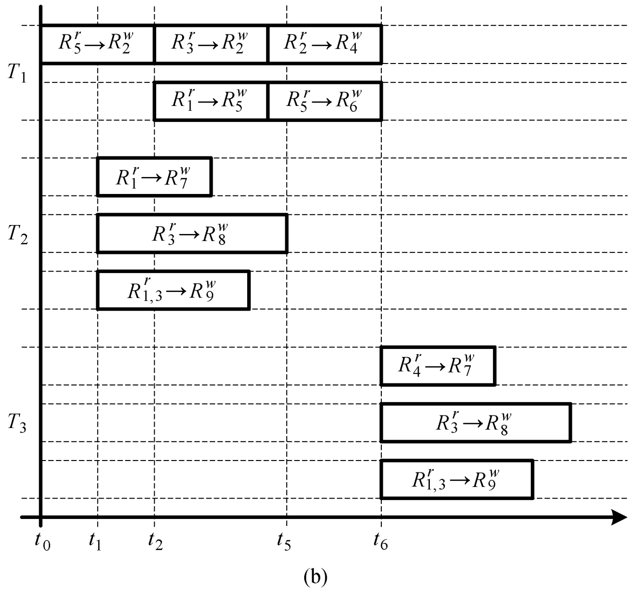


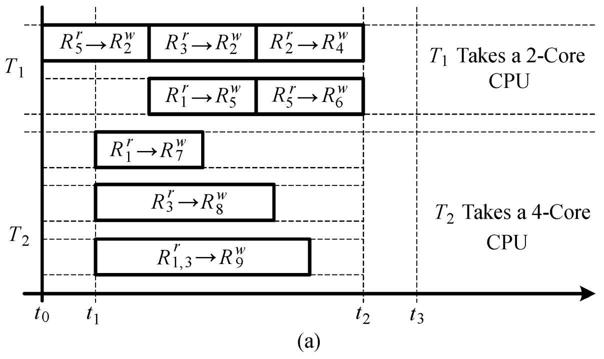
Fig.2. Executing in parallel.

## 3.1 Motivation 1

In Fig.2(a), transactions $T_1$ has five events. Fig.2(b) shows a possible scheduling. Apparently, two processors will make $T_1$ execute faster after $t_1$ and no more processors are necessary. We can make the decision artificially since $T_1$ is a very simple and short transaction. If a transaction is more complicated and has much more events, how could we do that?

## 3.2 Motivation 2

In Fig.3, there are two transactions, $T_1$ and $T_2$. Both of them can work in parallel. As we can see, $T_1$ needs less cores than $T_2$ though it has more events than $T_2$. Figs. 3(a) and 3(b) are two feasible solutions of assignment. In Fig.3(a), $T_1$ gets a 2-core processor and $T_2$ gets a 4-core one. In Fig.3(b), the situation gets reversed. The difference sticks out a mile. Giving a processor with appropriate cores, instead of maximum cores, does make sense. What makes us decide to assign some transactions processors with more cores and others with less? What is the criterion?



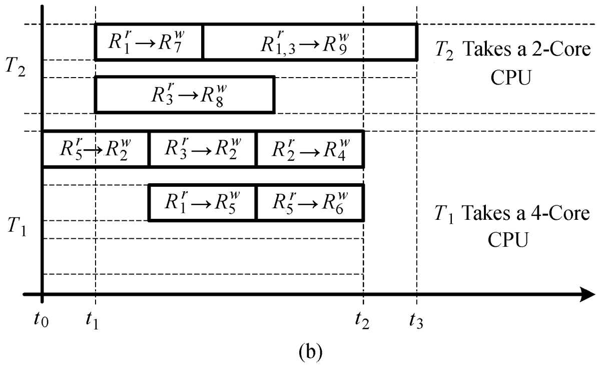


Fig.3. Difference of durations.

## 3.3 Motivation 3

Suppose, transaction $T_3$, shown in Fig.2, arrives at $t_2$. What is our decision at $t_2$? Shall we activate $T_3$ at $t_2$? Apparently, $T_3$ conflicts with $T_1$ and $T_2$. So we must wait until $T_1$ and $T_2$ submit or abort, otherwise $T_3$ may have a possibility to be restarted for consistency. How could we find conflicts rapidly out of hundreds of, even thousands of transactions?

## 4 Resources Snapshot Model

### 4.1 Terminologies

**Definition 1** (Event). *An event is an atomic operation to database, denoted as $E$.*

A selection operation without nested selection is a typical event. An insertion, update, or deletion operation is also an event.

**Definition 2** (Transaction). *A transaction is usually treated as a sequence of events, denoted as*

$$T_x = (E_x^1, E_x^2, \ldots, E_x^{k_x}),$$

*in which the events with less superscripts are supposed to be executed earlier than those with lager superscripts. In this paper, we use a form with a little difference. We represent it as a 3-tuple, denoted as*

$$T_x = \langle s\_time, r\_time, ES\rangle,$$

*or*

$$T_x = \langle s\_time, r\_time, (E_x^1, E_x^2, \ldots, E_x^{k_x})\rangle,$$

*for which s_time is the submitting time, r_time is the time at which $T_x$ starts to run and ES is the events sequence. If r_time $= -1$, then it means $T_x$ has not been activated.*

**Definition 3** (Transaction Stream). *A transaction stream is a sequence of transactions, denoted as*

$$\widetilde{T} = (T_1, T_2, \ldots, T_n),$$

*for which $\forall T_x, T_y \in \widetilde{T}$, it follows that if $x < y$, then $T_x.s\_time < T_y.s\_time$.*

**Definition 4** (Critical Resource). *In concurrent transaction processing, critical resource means a kind of resource which has only one instance, denoted as $R_n$, for which $n$ is the category ID of the resource. That is, $R_n$ cannot be concurrently accessed by more than one transaction unless all the transactions only read from it.*

In the following of this paper, the word resource only refers to critical resource if no other particular statements are claimed.

**Definition 5** (Read/Write Set). *A read/write set $\Theta$ is a set of resources read/writen by an event or a transaction during its execution. The read/write set for event $E_j^i$ is denoted as $\Theta^{r/w}(E_j^i)$, and the read/write set for transaction $T_x$ is denoted as $\Theta^{r/w}(T_x)$.*

*Suppose, $T_x = \langle t_0, t_1, \{E_x^1, E_x^2, \ldots, E_x^{k_x}\}\rangle$, then the read/write set for transaction $T_x$ is*

$$\Theta^{r/w}(T_x) = \bigcup_{i=1}^{k_x} \Theta^{r/w}(E_x^i).$$

In Fig.2(a), the read and write sets of the five events in $T_1$ are

$$\begin{aligned}
\Theta^r(E_1^1) &= \{R_5\}, \quad \Theta^w(E_1^1) = \{R_2\},\\
\Theta^r(E_1^2) &= \{R_3\}, \quad \Theta^w(E_1^2) = \{R_2\},\\
\Theta^r(E_1^3) &= \{R_1\}, \quad \Theta^w(E_1^3) = \{R_5\},\\
\Theta^r(E_1^4) &= \{R_2\}, \quad \Theta^w(E_1^4) = \{R_4\},\\
\Theta^r(E_1^5) &= \{R_5\}, \quad \Theta^w(E_1^5) = \{R_6\}.
\end{aligned}$$

The read sets of $T_1$ is $\Theta^r(T_1) = \{R_5, R_3, R_1, R_2\}$, and the write sets of $T_1$ is $\Theta^w(T_1) = \{R_2, R_5, R_4, R_6\}$.

**Definition 6** (Resource Pair). *A resource pair is a read set and its corresponding write set, which belong to a same event or transaction, denoted as $\Theta^r(E/T) \to \Theta^w(E/T)$ or $\Theta(E/T)$.*

In Fig.2(a), the resource pairs of the five events in $T_1$ are

$$\begin{aligned}
\Theta(E_1^1) &= \{R_5\} \to \{R_2\},\\
\Theta(E_1^2) &= \{R_3\} \to \{R_2\},\\
\Theta(E_1^3) &= \{R_1\} \to \{R_5\},\\
\Theta(E_1^4) &= \{R_2\} \to \{R_4\},\\
\Theta(E_1^5) &= \{R_5\} \to \{R_6\}.
\end{aligned}$$

The resource pair of $T_1$ is

$$\Theta(T_1) = \{R_5, R_3, R_1, R_2\} \to \{R_2, R_5, R_4, R_6\}.$$

**Definition 7** (Resource Snapshot). *The resource snapshot of $\widetilde{T}$ at time $t_i$, denoted as $\Xi_{t_i}(\widetilde{T})$, is a set of resource pairs. It represents the resource occupations and requests of transactions stream $\widetilde{T}$ at $t_i$. Let $\widetilde{T} = (T_1, T_2, \ldots, T_n)$, then we define*

$$\Xi_{t_i}(\widetilde{T}) = \{\Theta(T_1), \Theta(T_2), \ldots, \Theta(T_n)\}.$$

We can find out if a transaction is running or not at $t_i$ by checking its *r_time*.

In Fig.2(b), the resource snapshot at $t_0$, $t_1$, $t_2$, $t_5$ and $t_6$ are

$$\begin{aligned}
&\Xi_{t_0} = \{\Theta(T_1)\}, \quad \Xi_{t_1} = \{\Theta(T_1), \Theta(T_2)\},\\
&\Xi_{t_2} = \{\Theta(T_1), \Theta(T_2), \Theta(T_3)\},\\
&\Xi_{t_5} = \{\Theta(T_1), \Theta(T_3)\}, \quad \Xi_{t_6} = \{\Theta(T_3)\},
\end{aligned}$$

respectively. And in $\Xi_{t_2}$ and $\Xi_{t_5}$, $T_3$ is not active. $T_3$ cannot be activated until both $T_1$ and $T_2$ finish.

### 4.2 Resource Precedence Diagram

Pictorialization can give an intuitionistic view of resource snapshot. The Precedence Diagram Method is a tool for scheduling activities in a project plan. It is a method of constructing a project schedule network diagram that uses nodes to represent activities and connects them with arrows that show the dependencies.

In this paper, we use a digraph, named Resource Precedence Diagram (RPD), to represent a sequence of resource pairs. In the most common sense of the term, a digraph is a 2-tuple, $G = (V, A)$, comprising a set $V$ of vertices together with a set $A$ of arrows, which are ordered pairs of vertices.

**Definition 8** ($\gamma$-Function). *$\gamma$-function $\Gamma$ is a mapping from two resource pairs to one boolean value, i.e., $\Gamma : (\Theta_a, \Theta_b) \rightarrow true/false$, or $\Gamma(\Theta_a, \Theta_b) = true/false$.*

*Suppose,*

$$(\Theta_a^r \cap \Theta_b^w) \cup (\Theta_a^w \cap \Theta_b^r) \cup (\Theta_a^w \cap \Theta_b^w) = A,$$

*then*

$$\Gamma(\Theta_a, \Theta_b) = \begin{cases} false, & if \ \ A = \phi, \\ true, & if \ \ A \neq \phi. \end{cases}$$

In Fig.2,

$$\Gamma(\Theta(E_1^1), \Theta(E_1^2)) = \text{true},$$
$$\Gamma(\Theta(E_1^2), \Theta(E_1^3)) = \text{false},$$

and

$$\Gamma(\Theta(E_2^1), \Theta(E_2^2)) = \text{false},$$
$$\Gamma(\Theta(E_2^1), \Theta(E_2^3)) = \text{false},$$
$$\Gamma(\Theta(E_2^2), \Theta(E_2^3)) = \text{false},$$

and

$$\Gamma(\Theta(E_3^1), \Theta(E_3^2)) = \text{false},$$
$$\Gamma(\Theta(E_3^1), \Theta(E_3^3)) = \text{false},$$
$$\Gamma(\Theta(E_3^2), \Theta(E_3^3)) = \text{false},$$

and

$$\Gamma(\Theta(T_1), \Theta(T_2)) = \text{false},$$
$$\Gamma(\Theta(T_1), \Theta(T_3)) = \text{true},$$
$$\Gamma(\Theta(T_2), \Theta(T_3)) = \text{true}.$$

**Definition 9** (Resource Precedence Diagram). *Let $(\Theta_1, \Theta_2, \ldots, \Theta_n)$ be a sequence of resource pairs. The resource precedence diagram (RPD), $G = (V, A)$, is defined as follows,*

1) $V = \{\Theta_1, \Theta_2, \ldots, \Theta_n\}$,

2) $A = \{(\Theta_x, \Theta_y) | \Gamma(\Theta_x, \Theta_y) = true,\ x < y\}$.

In an RPD, a vertex at the endpoint of an arrow must execute after the starting point finishes. An RPD can be used in both inter-transaction and intra-transaction levels to research the possibility of parallelism among transactions or events.

Fig.4 shows two examples of RPD. We omit the symbols of $\Theta$ in the figure in order to make it more clear at a glance. Fig.4(a) shows an intra-transaction level RPD of transaction $T_1$. Fig.4(b) shows an inter-transaction level RPD of $\Xi_{t_2}$ of Fig.2(b).

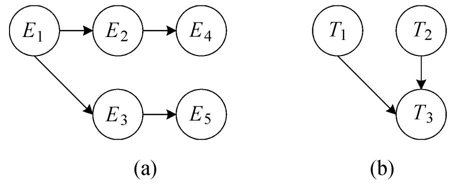


Fig.4. Examples of RPD.

**Definition 10** (Width of RPD). *Suppose, $G = (V, A)$ is an RPD and $V_0 \subseteq V$, for which there is no path between any two vertices in $V_0$. If $\forall V_x \subseteq V$, for which there is no path between any two vertices in $V_x$, it follows that $|V_x| \leqslant |V_0|$, then $|V_0|$ is defined as the width of $G$, denoted as $|G| = |V_0|$.*

We use a macro adjacency matrix to represent an RPD. A regular adjacency matrix of a digraph is an $n \times n$ square matrix $\boldsymbol{M}$, where $n$ is the number of vertices in the graph. If there is an arrow from some $V_x$ to some $V_y$, then the element $M[x, y]$ is 1, otherwise it is 0. A macro adjacency matrix is derived from a regular adjacency matrix. A macro adjacency matrix is also an $n \times n$ square matrix $\boldsymbol{M}$. If there is a path, instead of an arrow, from some $V_x$ to some $V_y$, then the element $M[x, y]$ is 1, otherwise it is 0.

## 5 Inter-Transaction Scheduling

### 5.1 $\delta$-Function

**Definition 11** ($\delta$-Function). *$\delta$-function $\Delta$ is a mapping from a sequence of resource pairs $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ to a set of resource pairs, named $\delta$-set, i.e., $\Delta : S \rightarrow \delta$-set, or $\Delta : S \rightarrow \{\Theta_{\delta_1}, \Theta_{\delta_2}, \ldots\}$, or $\Delta(S) = \delta$-set.*

*Suppose, $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$, then the $\delta$-set of $S$ must cover the following terms,*

1) *$\forall \Theta_x \in \delta$-set, $!\exists \Theta_y \in S$, such that $y < x$ and $\Gamma(\Theta_x, \Theta_y) = true$,*

2) *$\forall \Theta_y \in (S - (\delta$-set$))$, $\exists \Theta_x \in S$, such that $y > x$ and $\Gamma(\Theta_x, \Theta_y) = true$.*

In Fig.2, $T_1$, $T_2$ and $T_3$ all exist at $t_2$. If let

$$S = (\Theta(T_1), \Theta(T_2), \Theta(T_3)),$$

then

$$\Delta(\Xi_{t_2}) = \{\Theta(T_1), \Theta(T_2)\}.$$

**Theorem 1.** *Suppose, $\Xi_{t_i}$ is the resource snapshot at $t_i$, $G = (V, A)$ is the PRD of $\Xi_{t_i}$. If there exist $V_0 \subseteq V$ and $V_0 \neq \phi$, for which $\forall \Theta_x \in V_0$, it follows that* $\deg^-(\Theta_x) = 0$, *and* $\forall \Theta_y \in (V - V_0)$, *it follows that $deg^-(\Theta_y) > 0$, then $V_0 = \Delta(\Xi_{t_i})$.*

The notation $\deg^-()$ represents the indegree of a vertex in a graph.

*Proof.* As we know, $\forall\Theta_x \in \Delta(\Xi_{t_i})$, it follows that $!\exists\Theta_y \in \Xi_{t_i}$, such that $y < x$ and $\Gamma(\Theta_x, \Theta_y) = \text{true}$. This means that $\forall\Theta_x \in \Delta(\Xi_{t_i})$, there does not exist any arrow pointing to $\Theta_x$. It follows that $\deg^-(\Theta_x) = 0$.

Suppose, $\Theta_y$ be any element in $\Xi_{t_i} - \Delta(\Xi_{t_i})$, it follows that $\exists\Theta_x \in \Xi_{t_i}$, such that $y > x$ and $\Gamma(\Theta_x, \Theta_y) = \text{true}$. This means that $\forall\Theta_y \in (\Xi_{t_i} - \Delta(\Xi_{t_i}))$, there exists at least one arrow pointing to $\Theta_y$. It follows that $\deg^-(\Theta_y) > 0$.

Therefore, $V_0 = \Delta(\Xi_{t_i})$. □

It is known that, in Fig.2(b), the $\delta$-set of $\Xi_{t_2}$ is

$$\Delta(\Xi_{t_2}) = \{\Theta(T_1), \Theta(T_2)\}.$$

It was dug out by $\Delta$-function in the previous part of this section. Now PPD is an alternative solution. Fig.4(b) is the RPD of $\Xi_{t_2}$. It is abundantly clear which vertices have the indegrees of 0.

## 5.2 Algorithms

Inter-transaction scheduling is to find out which transactions can execute in parallel at time $t$. It is known that the $\delta$-set of $\Xi_t$ will not occur any conflicts. Theorem 1 provides a very quick method, counting the vertices with indegree of 0, to get a $\delta$-set.

The inter-transaction scheduling algorithm should be invoked as soon as some transactions finish or arrive. The two possible situations are as follows.

1) Once a transaction arrives at time $t$, we add it and its inbound arrows into the RPD and figure out the $\delta$-set of $\Xi_t$.

2) Once a transaction finishes at time $t$, we drop it and its outbound arrows from the RPD and figure out the $\delta$-set of $\Xi_t$.

Algorithm 1 and Algorithm 2 describe the details of how to deal with these two situations, respectively. These two algorithms both construct on the macro adjacency matrix of RPD.

Fig.5 explains inter-transaction scheduling by RPD and $\delta$-set. There are two transaction $(T_1, T_2)$ at $t_2$ and the $\delta$-set is $\{T_1, T_2\}$. This means $T_1$ and $T_2$ can execute in parallel. So $T_2$ is activated immediately when it arrives.

**Algorithm 1.** Arrive($T_a$)

```
1:  Let $R$ be the running queue
2:  Let $W$ be the waiting queue
3:  $n \Leftarrow |R| + |W|$
4:  $W \Leftarrow W \cup T_a$
5:  Add the $(n+1)$-th column into the macro adjacency matrix $\boldsymbol{M}$
6:  Add the $(n+1)$-th row into $\boldsymbol{M}$
7:  for $i = 1$ to $n$ do
8:      $M[n+1, i] \Leftarrow 0$
9:      if $\Gamma(\Theta_n, \Theta_a) =$ ture then
10:         $\boldsymbol{M}[n, n+1] \Leftarrow 1$
11:     end if
12:     $\boldsymbol{M}[n+1, n+1] \Leftarrow 0$
13: end for
```

**Algorithm 2.** Finish($T_a$)

```
1: Let $R$ be the running queue
2: Let $W$ be the waiting queue
3: $n \Leftarrow |R| + |W|$
4: Drop the $a$-th row
5: Drop the $a$-th column
6: for each $T_i$, $1 \leqslant i \leqslant n-1$, such that each element in column $i = 0$ do
7:     $R \Leftarrow R \cup T_i$
8:     $W \Leftarrow W - T_i$
9: end for
```

The $\delta$-set at $t_3$ is still $\{T_1, T_2\}$. So $T_3$ is not activated when it arrives because it is not in the $\delta$-set. Moreover, $T_3$ is still not activated at $t_5$ because of the same reason.

## 5.3 Finiteness

Exploring the $\delta$-set now becomes the extremely important task, otherwise the inter-transaction scheduling will be endless.

**Theorem 2.** *An PRD is a directed acyclic graph (DAG).*

*Proof.* Let, $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ be a sequence of resource pairs, and $G = (V, A)$ is the corresponding PRD.

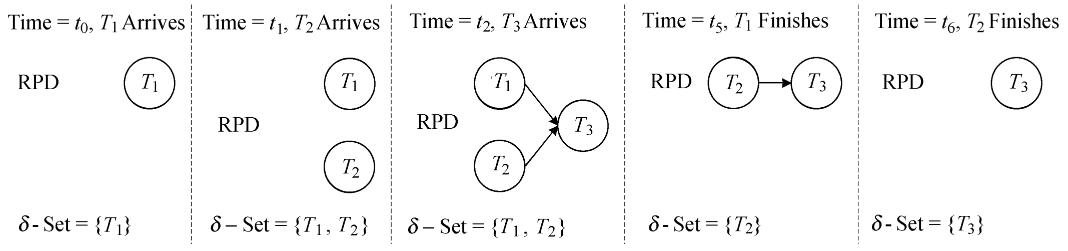

Time = $t_0$, $T_1$ Arrives
RPD
$T_1$
$\delta$- Set = $\{T_1\}$
Time = $t_1$, $T_2$ Arrives
RPD
$T_1$
$T_2$
$\delta$ – Set = $\{T_1, T_2\}$
Time = $t_2$, $T_3$ Arrives
RPD
$T_1$
$T_2$
$T_3$
$\delta$ – Set = $\{T_1, T_2\}$
Time = $t_5$, $T_1$ Finishes
RPD
$T_2$
$T_3$
$\delta$- Set = $\{T_2\}$
Time = $t_6$, $T_2$ Finishes
RPD
$T_3$
$\delta$- Set = $\{T_3\}$

Fig.5. Illustration of inter-transaction scheduling.

Suppose, there exists a cycle in $G$, it follows that

$$\exists(\Theta_{c_1}, \Theta_{c_2}, \ldots, \Theta_{c_m}),$$

for which

$$c_1 < c_2 < \cdots < c_m,$$

such that

$$\begin{aligned}\Gamma(\Theta_{c_1}, \Theta_{c_2}) &= \text{true},\\ &\vdots\\ \Gamma(\Theta_{c_{m-1}}, \Theta_{c_m}) &= \text{true},\\ \Gamma(\Theta_{c_m}, \Theta_{c_1}) &= \text{true}.\end{aligned}$$

According to Definition 9, if $\Gamma(\Theta_{c_m}, \Theta_{c_1}) = \text{true}$, then $c_m < c_1$. It breaks the hypothesis, $c_1 < c_m$, apparently.

Therefore, an PRD is a directed acyclic graph. □

**Corollary 1.** *Suppose, $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ is a sequence of resource pairs, if $n > 0$ then, $\Delta(S) \neq \phi$.*

*Proof.* Suppose $G = (V, A)$ is the corresponding PRD of $S$. Suppose, $\Delta(S) = \phi$, it follows that $!\exists V_x \in V$, such that $\deg^-(V_x) = 0$. It follow that $\deg^-(V_0) \neq 0$. This means that $\exists V_x \in V, x > 0$, such that $(V_x, V_0) \in A$. It breaks the conditions of Definition 9.

Therefore, the hypothesis, $\Delta(S) = \phi$, is not true, i.e., $\Delta(S) \neq \phi$. □

Theorem 2 and Corollary 1 prove that a $\delta$-set always exists with a non-empty sequence of resource pairs. So the inter-transaction scheduling is a finite algorithm.

## 5.4 Overhead of Scheduling

Once a transaction starts or finishes, the RPD and macro adjacency matrix should be updated, and the algorithms should be triggered.

Suppose $R$ is the running queue, $W$ is the waiting queue, and each transaction holds, on average, no more than $k$ resources. It follows that

- Algorithm 1 invokes $n$ ($n = |R| + |W|$) times of $\Gamma$-function which needs $3k^2$ times of comparisons for each time.
- Algorithm 2 does not invoke $\Gamma$-function. It goes through the waiting queue to launch unrestricted transactions released by the leaving transaction. It needs $n - 2$ times of comparisons to see if a transaction is with the indegree of 0. And the worst situation is that the entire waiting queue needs to be checked.
- Each invoking of Algorithm 1 will result in a consequent invoking of Algorithm 2.

Therefore, the overhead of inter-transaction scheduling is $H = 3nk^2 + (n - 2) \times |W|$. The worst situation makes the maximum $|W|$ be $n - (|R| - 1)$. $|R|$ is limited by the number of processors, so usually $|R| \ll n$.

It is easy to find that the complexity of $H$ falls into $O(n^2)$. It is not acceptable absolutely.

Actually, the macro adjacency matrix is a sparse matrix. Most of the comparisons are useless. So the RPD uses cross linked list to store the matrix, instead of a regular $n \times n$ array.

To add a transaction into an RPD with $n$ vertices needs up to $n$ times of $\Gamma$-function. Algorithm 1 with cross linked list still needs $3nk^2$ times of comparisons. To remove a transaction needs to release up to $2 \times n$ nodes in the cross linked list first.

After that, it becomes extremely easy to find unrestricted transactions. The scheduler just needs to check up to $|W|$ column pointers to see if they are NULL or not, which means indegree of 0. Therefore, the complexity of updating an RPD with cross linked list is

$$\begin{aligned}H &= 3k^2n + 2n + |W| = 3k^2n + 2n + (n - |R|)\\ &= 3k^2n + 3n - |R| < 3k^2n + 3n.\end{aligned}$$

For $k \ll n$, $k$ can be treated as a small constant, let $3k^2 = c$, it follows that $H < (c + 3)n$, therefore the complexity of $H$ is $O(n)$.

## 6 Intra-Transaction Scheduling

Intra-transaction scheduling is to find out how many threads a transaction can be divided into at most and pick an appropriate processor for the transaction.

### 6.1 $\psi$-Function

**Definition 12** ($\psi$-Function). *$\Psi$-function $\Psi$ is a mapping from a resource pair sequence to a positive integer. The result of a $\psi$-function is called a $\psi$-value, i.e., $\Psi : (\Theta_1, \Theta_2, \ldots, \Theta_n) \to \psi$-value, or $\Psi(\Theta_1, \Theta_2, \ldots, \Theta_n) = \psi$-value.*

*Suppose, $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$, and $S'$ is such a subset of $S$ that if $\forall \Theta_x, \Theta_y \in S' (x < y)$, it follows that $!\exists(\Theta_{x_1}, \Theta_{x_2}, \ldots, \Theta_{x_m})$, where $m \geqslant 0$ and $x < x_1 < x_2 < \cdots < x_m < y$, such that*

$$\begin{aligned}\Gamma(\Theta_x, \Theta_{x_1}) &= \textit{true},\\ \Gamma(\Theta_{x_1}, \Theta_{x_2}) &= \textit{true},\\ &\vdots\\ \Gamma(\Theta_{x_m}, \Theta_y) &= \textit{true}.\end{aligned}$$

*Then the cardinality of the largest $S'$ is the $\psi$-value of $S$, denoted as $\Psi(S) = |S'|$.*

In Fig.2, transaction $T_1$ has five events. If let $S = \{\Theta(E_1^1), \Theta(E_1^2), \Theta(E_1^3), \Theta(E_1^4), \Theta(E_1^5)\}$, then it has four subsets which cover all the terms of Definition 12. And it is easy to see,

$$\begin{aligned}|\{\Theta(E_1^2), \Theta(E_1^3)\}| &= 2,\\ |\{\Theta(E_1^2), \Theta(E_1^5)\}| &= 2,\end{aligned}$$

$$|\{\Theta(E_1^3), \Theta(E_1^4)\}| = 2,$$
$$|\{\Theta(E_1^4), \Theta(E_1^5)\}| = 2.$$

Therefore, $\Psi(S) = 2$.

**Theorem 3.** *If $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ is a sequence of resource pairs, and $G = (V, A)$ is the corresponding RPD, then $|G| = \Psi(S)$.*

*Proof.* Suppose, $S' = \{\Theta_{s_1}, \Theta_{s_2}, \ldots, \Theta_{s_m}\}$ be the largest subset of $S$, for which $\forall \Theta_x, \Theta_y \in S' (x < y)$, it follows that $!\exists(\Theta_{x_1}, \Theta_{x_2}, \ldots, \Theta_{x_m})$, where $m \geqslant 0$ and $x < x_1 < x_2 < \cdots < x_m < y$, such that

$$\begin{aligned}\Gamma(\Theta_x, \Theta_{x_1}) &= \text{true},\\ \Gamma(\Theta_{x_1}, \Theta_{x_2}) &= \text{true},\\ &\vdots\\ \Gamma(\Theta_{x_m}, \Theta_y) &= \text{true}.\end{aligned}$$

It follows that $\Psi(S) = |S'| = m$, and there does not exist any path in subgraph $G' = (S', A')$. Therefore, $|G| \geqslant |G'| = m$.

Suppose, $|G| > \Psi(S)$, it follows that $\exists S'' \in V, |S''| > |S'|$, such that there does not exist any path in subgraph $G'' = (S'', A'')$. It is not hard to see, $S''$ covers all the conditions covered by $S'$. This means that $S'$ is not the largest eligible subset. It breaks the hypothesis.

Therefore, $|G| = \Psi(S)$. $\square$

In Fig.2(a), as we know, the $\psi$-value of $T_1$ is 2. Fig.4(a) is the RPD of $T_1$. It shows the result abundantly clear.

Now the $\psi$-value can be got by figuring out the width of an RPD, since $\psi$-value of a resource pair sequence is equivalent to the width of its corresponding RPD.

```
Algorithm 3. ψ-Function($\Theta_1, \Theta_2, \ldots, \Theta_n$)
1:  $max \Leftarrow 0$
2:  Let $G = (V, A)$ be the RPD of the input sequence
3:  Let $\boldsymbol{M}$ be the macro adjacency matrix of $G$
4:  for $i = 1$ to $n$ do
5:      $\boldsymbol{M}' \Leftarrow \boldsymbol{M}$
6:      Drop the first $j - 1$ rows
7:      Drop the first $j - 1$ columns
8:      for $j = 1$ to $n - i + 1$ do
9:          Drop the columns whose entities in the $j$-th
            row are 1
10:     end for
11:     $k \Leftarrow$ the number of columns of $\boldsymbol{M}'$
12:     if ($k > max$) then
13:         $max \Leftarrow k$
14:     end if
15: end for
16: Output($max$)
```

*Proof.* As we can see, Algorithm 3 is used to find out the largest submatrix $\boldsymbol{M}'$ whose entities are all 0. Suppose, $G' = (V', A')$ is the subgraph represented by $M'$, then $\forall \Theta_x, \Theta_y \in V'$, it follows that $(\Theta_x, \Theta_y) \notin A'$. This means there is no path between any two vertices in $V'$. Therefore, $G'$ is the largest subgraph of $G$ in which there does not exist any path. $\square$

## 6.2 Non-Preemption RSM Scheduling

Non-preemption RSM picks a processor whose number of cores is equal to the $\psi$-value of the transaction. If there are no such processors available, non-preemption RSM will pick an available processor with number of cores as close to the $\psi$-value as possible.

Suppose, $T = (E_1, E_2, \ldots, E_n)$ is a transaction. Let $S = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ be the corresponding resource pair sequence and $G = (V, A)$ be the RPD. We can get the $\psi$-value of $S$ by means of $\Psi(S) = |G|$, according to Algorithm 3. So we can assign a processor with at least $|G|$ cores to transaction $T$ if it is possible.

The Non-Preemption Algorithm is given in Algorithm 4. Suppose we have $k$ processors, denoted as $p[x]$ $(1 \leqslant x \leqslant k)$, organized as an ascending sequence by the number of cores. Notations $c[x]$ and $s[x]$ mean the number of cores and the current state of $p[x]$, respectively.

```
Algorithm 4. Non-Preemption-RSM($T$)
1:  $mindiff \Leftarrow$ MAXCORE
2:  $currentp \Leftarrow$ NOTAVAILABLE
3:  for $i = 1$ to $k$ do
4:      if ($|(c[i] - \Psi(T)| < mindiff$ and $s[i] =$ IDLE)
        then
5:          $mindiff \Leftarrow |(c[i] - \Psi(T)|$
6:          $currentp \Leftarrow i$
7:      end if
8:  end for
9:  Output($currentp$)
```

Fig.6 shows an example of intra-transaction scheduling. It shows the process of getting the $\psi$-value of a transaction. There are five events in Fig.6, and the $\psi$-value is 2. It means a processor with no less than two cores is enough to transaction $T$.

Fig.7 shows a more complicated example. There are nine events in transaction $T$, and the result shows that a processor with more than four cores allows transaction $T$ to finish running in the shortest duration.

## 6.3 Preemption RSM Scheduling

Preemption RSM has the same principle for picking a processor as non-preemption RSM. The difference is that a transaction can get a reassigned processor before

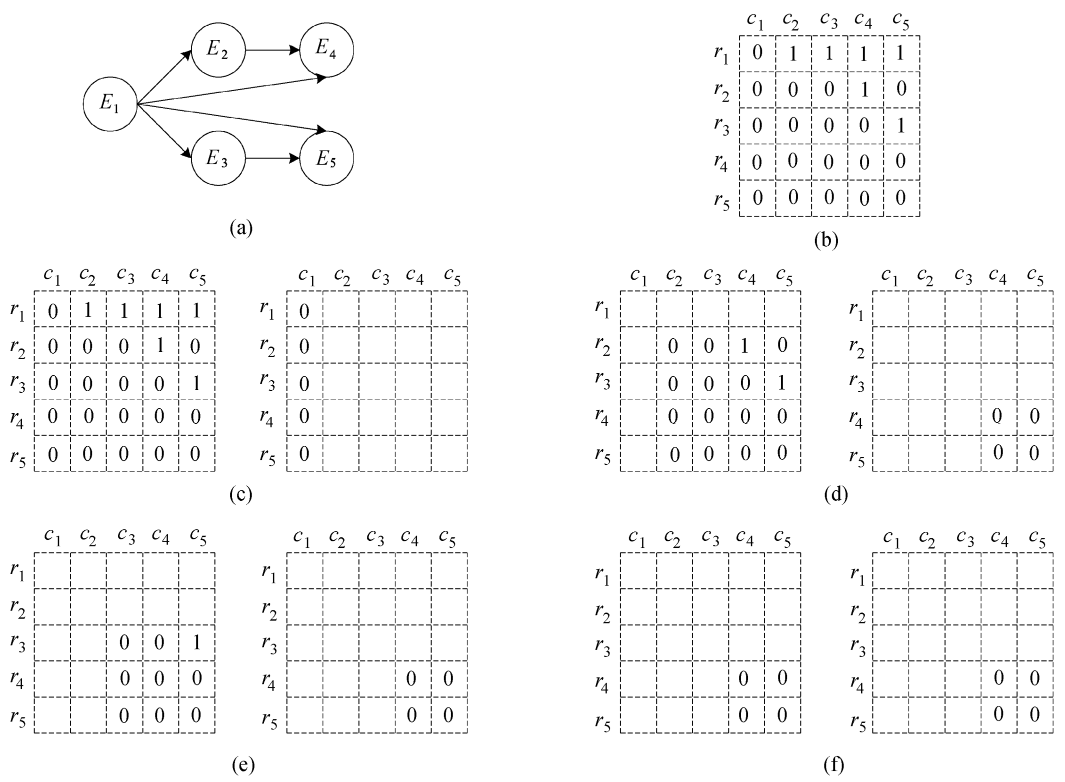


Fig.6. Illustration of intra-transaction scheduling. (a) RPD fo transaction $T$. (b) Initial $M$. (c) $i = 1$, $\psi$-value = 1. (d) $i = 2$, $\psi$-value = 2. (e) $i = 3$, $\psi$-value = 2. (f) $i = 4$, $\psi$-value = 2.

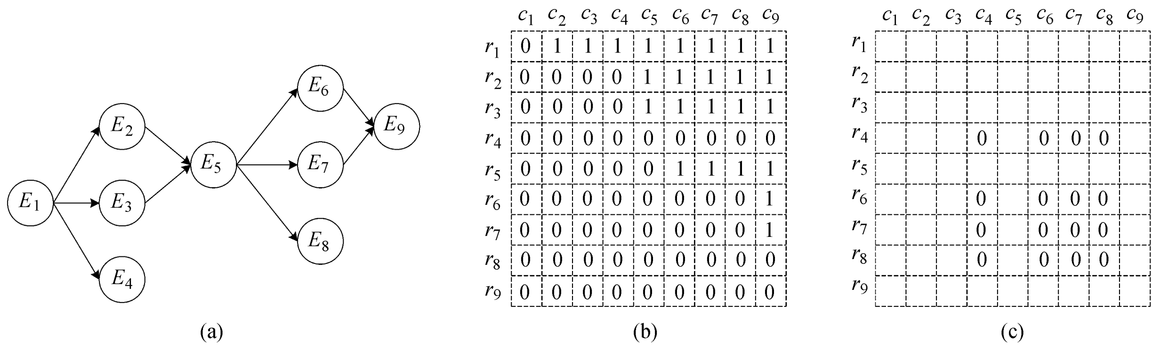


Fig.7. Illustration of complicated intra-transaction scheduling. (a) RPD of transaction $T$. (b) Initial $M$. (c) $i = 4$, $\psi$-value = 4.

it finishes. Algorithm 4 intends to assign the most appropriate processor to a transaction, but it does not always work. Preemption RSM keeps watch on the status of all processors. Once a more appropriate processor is released, the transaction will get it.

## 7 Experiments and Performance Study

We construct a simulation model to evaluate performance of RSM scheduling and compare the results with classic shared and exclusive locking scheduling.

### 7.1 Experimental Settings

We set up a PC cluster for experiments with 15 computers, including 14 computing nodes whose configurations are listed in Table 1, and one scheduling node. All nodes run under Redhat Enterprise Linux

**Table 1.** Hardware Configuration

| CPU | Cores | CLK (GHz) | PCs |
|---|---|---|---|
| Pentium-IV | 1 | 2.8 | 4 |
| Pentium G840 | 2 | 2.8 | 4 |
| AMD A6-3670 K | 4 | 2.7 | 4 |
| AMD FX6100 | 6 | 3.3 | 1 |
| Core i7-3770 K | 4(8) | 3.5 | 1 |

Note: Intel Core i7 has 4 physical cores, but the Hyper-Threading technology makes it powerful enough to work like an 8-core processor. In our experiments, we treat Core i7 as an 8-core processor.

Server, and GCC+MPI is adopted as coding environment.

Since the RPD is very similar to task graph in parallel systems, it can be deployed by borrowing the parallel task scheduling method to solve the scheduling problem. We choose a well-studied parallel task scheduling of DAG, DLS (Dynamic Level Scheduling)[33], as a baseline in our experiments. We just make a little modification on the ranking of levels.

## 7.2 Benchmark

The purpose of a benchmark is to reduce the diversity of operations found in a production application, while retaining the application's essential performance characteristics so that the workload can be representative of a production system. We study the performance of RSM by means of the benchmark of TPC-E (TPC Benchmark$^{\text{TM}}$ E[①].

TPC-E is a new on-line transaction processing (OLTP) workload developed by the TPC. TPC-E is composed of a set of transactional operations designed to exercise system functionalities in a manner representative of complex OLTP application environments. These transactional operations have been given a life-like context, portraying the activity of a brokerage firm, to help users relate intuitively to the components of the benchmark. The workload is centered on the activity of processing brokerage trades and uses a schema, which is logically divided into four sets of tables.

The testing data packages were generated by EGen[①]. EGen is a TPC-provided software package designed to facilitate the implementation of TPC-E. EGen provides consistent data generation independent of the underlying environment.

## 7.3 Simulation Model

Our simulation model consists of a transaction model which captures the characteristics of transactions, and a scheduling model which captures the behaviors of scheduling algorithms. Details are as follows.

1) We assume each transaction has up to eight events and each event requests up to two resources.

2) We simulate two types of schedule algorithms. One is RSM, and the other is the classic locking scheduler. Shared and Exclusive Locking Schema (S/E Locking) is the most widely used algorithm.

3) We implement four algorithms totally, two for S/E Locking and two for RSM. After S/E Locking finishes inter-transaction schedule, Optimism or Pessimism continues to finish intra-transaction schedule. Optimism always picks a processor with maximum number of cores out of all available processors, while Pessimism always picks a processor with minimum number of cores out of all available processors. So these two S/E Locking schedulings are named S/E LOCK-OPTI and S/E LOCK-PESS, respectively. RSM has two phases, one for inter-transaction schedule and the other for intra-transaction schedule. After the first phase of RSM, the second phase has two alternatives, Preemption and Non-preemption. So these two RSM schedulings are named PREE-RSM and NONP-RSM, respectively.

4) Two main factors which are possible to influence the performance of scheduling are considered: average duration of a transaction, average number of resources requested.

5) Two criteria relevant to scheduling performance are introduced: average turnaround index (ATI) for transactions and average busy index (ABI) for processors.

Suppose, $D_i$ is the duration of transaction executing in serial manner, and $d_i$ is the actual duration during a scheduling. Suppose there are totally $N$ transactions. The average turnaround index is defined as

$$\text{ATI} = -100 \times \ln \sum_{i=1}^{N} \frac{d_i}{D_i}.$$

Suppose, there is $P$ processors, each has $p_i$ cores, and $C(p_i|t_j)$ represents the number of working cores of $p_i$ at timeslice $t_j$. Suppose, it costs totally $S$ timeslices to execute some transactions. The average busy index is defined as

$$\text{ABI} = \sum_{j=1}^{S} \sum_{k=1}^{P} C(p_k|t_j) \Big/ \Big( S \times \sum_{i=1}^{P} p_i \Big).$$

## 7.4 Results and Discussion

Fig.8 and Fig.9 show that the ATI and ABI are stable while the average duration of transactions ranges from 50 to 100. Average duration changes the life span

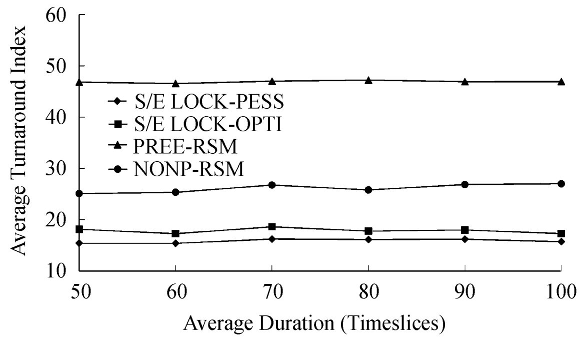


Fig.8. ATI vs average duration.

[①]http://www.tpc.org/tpce/default.asp, June 2001.

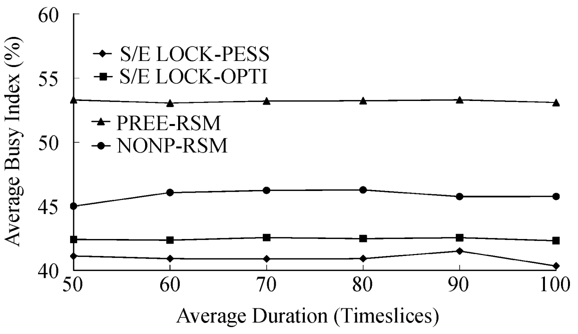


Fig.9. ABI vs average duration.

of transactions. But it does not change the probability of resource racing among the transactions. It just amplifies the interval between two conflicts. Preemption RSM is capable of switching transactions to appropriate processors. It makes the ABI also higher than that of the other three.

Fig.10 and Fig.11 show that the ATI and ABI are stable while the average number of requested resources ranges from 2 to 16. Average number of requested resources changes the possibility of conflicts. But its influence on any schedulers are equivalent. So it cannot make the performance of any of these four schedulers drop faster. But when the average number of requested resources gets greater than 14, the charts appear somewhat abnormal. Actually, too many conflicts occur. It is difficult for any scheduling policy to keep away from them.

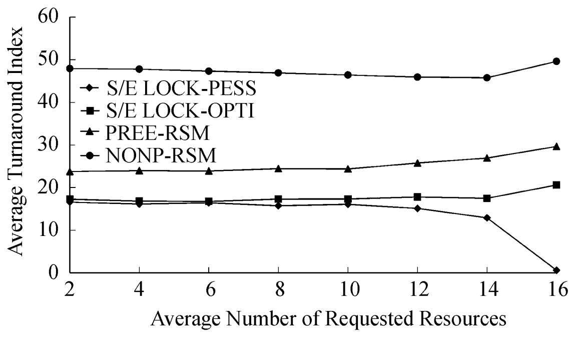


Fig.10. ATI vs average number of requested resources.

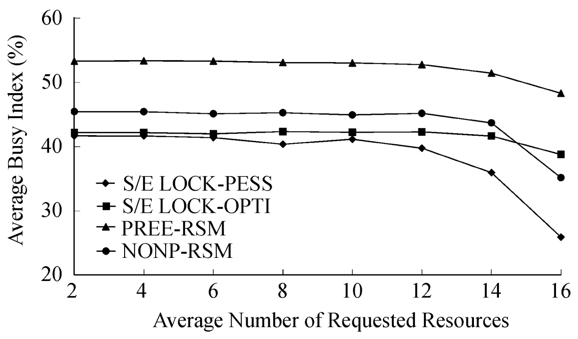


Fig.11. ABI vs average number of requested resources.

## 8 Conclusions and Future Work

We proposed a model, named RSM, which represents the resource requirements and occupations of a transaction stream at one moment. RSM learns the dependency on two transaction by presenting the $\gamma$-function and uses RPD to study the dependency in a transaction stream. RSM supports the research on parallelism of transactions in both inter-transaction level and intra-transaction level. It provides a non-restart and non-locking scheduling method in inter-transaction level by means of $\delta$-set. It also provides a processor assignment method under asymmetric multi-processor situations in intra-transaction level by means of $\psi$-value.

We conducted simulations for various situations of transaction streams in order to examine the performance of two non-restart schedulings, Non-Preemption and Preemption RSM scheduling. We found that the RSM schedulings are more efficient than some other schedulings in asymmetric multi-processor system and Preemption RSM is more efficient than Non-Preemption RSM.

A lot of work still remains. The condition of non-restart scheduling is sufficient, but no necessary. It guarantees that no restarting is needed and loses some parallelism in the mean time. So the RSM schedulings are feasible schedulings, but perhaps not the best schedulings which lead to the maximum average throughput and the minimum average waiting time. Another issue is the assumption that transactions' resource requirements are known beforehand limits the applicability of the work. It would be more useful to extend the algorithms to operate in a context where transactions' resource requirements only become known as the transactions dynamically execute. In the future, we would like to do more research on RSM and find out more efficient and practical non-restart scheduling methods.

## References

[1] Ansari M, Rusinkiewicz M, Ness L *et al.* Executing multidatabase transactions. In *Proc. the 25th Hawaii International Conference on System Sciences*, Jan. 1992, pp.335-346.

[2] Chrysanthis P K, Ramamritham K. Acta: A comprehensive transaction framework for extended transactions. In *Proc. the 2nd Int. Workshop on Transaction and Query Processing*, Feb. 1992, p.221.

[3] Sharaf M A, Guirguis S, Labrinidis A *et al.* Asets: A self-managing transaction scheduler. In *Proc. the 24th Int. Conf. Data Engineering Workshop* (*Poster*), Apr. 2008, pp.56-62.

[4] Biliris A, Dar S, Gehani N *et al.* Asset: A system for supporting extended transactions. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, May 1994, pp.44-54.

[5] Garcia-Molina H, Ullman J D, Widom J. Database Systems: The Complete Book (2nd edition). Prentice-Hall, 2008.

[6] Li G, Xiang J, Yang B *et al.* Scheduling algorithm of update transactions and quality of service management based on derived data in real-time and mobile database systems. In *Proc. Japan-China Joint Workshop on Frontier of Computer Science and Technology*, Nov. 2007, pp.131-138.

[7] Gruenwald L, Bernedo P, Padmanabhan P. Petranet: A power efficient transaction management technique for real-time mobile ad-hoc network databases. In *Proc. the 22nd International Conference on Data Engineering*, Apr. 2006, p.172.

[8] Alshorman R, Hussak W. Multi-step transactions specification and verification in a mobile database community. In *Proc. the 3rd Int. Conf. Information and Communication Technologies: From Theory to Applications*, Apr. 2008, pp.1-6.

[9] Chung I, Bhargava B, Mahoui M *et al.* Autonomous transaction processing using data dependency in mobile environments. In *Proc. the 9th IEEE Workshop on Future Trends of Distributed Computing Systems*, May 2003, pp.138-144.

[10] Lim J B, Hurson A R. Transaction processing in mobile, heterogeneous database systems. *IEEE Transactions on Knowledge and Data Engineering*, 2002, 14(6): 1330-1346.

[11] Han J, Li Q. A transaction scheduling algorithm with temporal constraints in real-time database systems. In *Proc. the 4th International Conference on Computer and Information Technology*, Sept. 2004, pp.940-945.

[12] Fernandes Y M P, Perkusich A, Neto P F R *et al.* Implementation of transactions scheduling for real-time database management. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2004, pp.5136-5141.

[13] Chen H, Chin Y H, Tseng S. Scheduling value-based transactions in distributed real-time database systems. In *Proc. the 15th Int. Conf. Parallel and Distributed*, 2001, pp.978-984.

[14] Neto P F R, Perkusich A, Perkusich M L B *et al.* Analysis of periodic transactions and semantic concurrency control for real-time databases using colored petri nets. In *Proc. IEEE Int. Conf. Systems Man and Cybernetics*, Oct. 2001, pp.2723-2728.

[15] Lam K, Kuo T, Lee T S H. Designing inter-class concurrency control strategies for real-time database systems with mixed transactions. In *Proc. the 12th Euromicro Conference on Real-Time Systems*, Jan. 2000, pp.47-54.

[16] Lee V C S, Lam K, Hung S. Concurrency control for mixed transactions in real-time databases. *IEEE Transactions on Computers*, 2002, 51(7): 821-834.

[17] Han Y, Jiang C, Luo X. Priority based transaction scheduling model and concurrency control in grid database. In *Proc. the 7th International Conference on Grid and Cooperative Computing*, Oct. 2008, pp.235-241.

[18] Zhang Q, Sui S, Li J. Research and realization of transaction concurrency control in grid database. In *Proc. the 6th Int. Conf. Grid and Cooperative Computing*, Aug. 2007, pp.168-172.

[19] Fujiyama K, Nakamura N, Hiraike R. Database transaction management for high-availability cluster system. In *Proc. the 12th Pacific Rim International Symposium on Dependable Computing*, Dec. 2006, pp.139-146.

[20] Chrysanthis P K, Ramamritham K. A formalism for extended transaction model. In *Proc. the 17th International Conference on Very Large Data Bases*, Sept. 1991, pp.103-112.

[21] Chrysanthis P K, Ramamritham K. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, 1994, 19(3): 450-491.

[22] Schwarz K, Turker C, Saake G. Execution dependencies in transaction closures. In *Proc. the 3rd Int. Conf. Cooperative Information Systems*, Aug. 1998, pp.122-131.

[23] Schwarz K, Turker C, Saake G. Transitive dependencies in transaction closures. In *Proc. the Int. Database Engineering and Applications Symposium*, Jul. 1998, pp.34-43.

[24] Taniar D, Goel S. Concurrency control issues in grid database. *Future Generation Computer Systems*, 2007, 23(1): 154-162.

[25] Xing Z, Gruenwald L, Phang K K. SODA: An algorithm to guarantee correctness of concurrent transaction execution in mobile p2p databases. In *Proc. of the 19th Int. Conf. Database and Expert Systems Application*, Sept. 2008, pp.337-341.

[26] Deng Y, Frankl P, Chays D. Testing database transactions with agenda. In *Proc. the 27th International Conference on Software Engineering*, May 2005, pp.78-87.

[27] Deng Y, Frankl P, Chen Z. Testing database transaction concurrency. In *Proc. the 18th IEEE International Conference on Automated Software Engineering*, Oct. 2003, pp.184-193.

[28] Xin T, Ray I. Detection for conflicts of dependencies in advanced transaction models. In *Proc. the 9th Int. Database Engineering and Application Symp.*, Jul. 2005, pp.17-26.

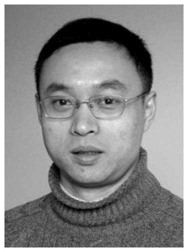
[29] Vijaykumar T N, Gopal S, Smith J E *et al.* Speculative versioning cache. *IEEE Transactions on Parallel and Distributed Systems*, 2001, 12(12): 1305-1317.

[30] Gopal S, Vijaykumar T N, Smith J E *et al.* Speculative versioning cache. In *Proc. the 4th Int. Symp. High-Performance Computer Architecture*, Feb. 1998, pp.195-205.

[31] Hammond L, Hubbert B A, Siu M *et al.* The Stanford hydra CMP. *IEEE Micro*, 2000, 20(2): 71-84.

[32] Christopher B C, Ailamaki A, Steffan J G *et al.* Incrementally parallelizing database transactions with thread-level speculation. *ACM Trans. Computer Systems*, 2008, 26(1), Article No.2.

[33] Sih G C, Lee E A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *Trans. Parallel and Distributed Systems*, 1993, 4(2): 75-87.



**Lei Zhao** received the Ph.D. degree in computer science in 2006 from Soochow University, Suzhou, China. He has been a faculty member of the School of Computer Science and Technology of Soochow University since 1998. He is now an associate professor at the Department of Network Engineering, Soochow University. His research interests include distributed data processing, data mining, parallel and distributed computing.

**Ji-Wen Yang** received the B.S. degree in mathematics in 1984 from Nanjing Normal University, China. He has been a faculty member of the School of Computer Science and Technology of Soochow University, China, since 1984. He is now a professor at the Department of Information Management. His research interests include distributed data processing, management information system, parallel and distributed computing.