Transaction in Large-scale Device Collaborative System

Feng Chen, Xiaohui Rong, Pan Deng, Shilong Ma
State Key Laboratory of Software Development Environment
Beihang University
Beijing, P.R. China
e-mail:chenfeng@nlsde.buaa.edu.cn

***Abstract*—In the modern world, the large-scale area devices collaborative technology becomes the supporting technology in many areas. The demand on transaction for device collaborative system is high, but relevant research is scarce. In this paper, we propose a new transaction model, which loose the requirement of ACID. We also give device operations conflicts detection rules, and a mechanism to calculate compensation operation is proposed, which can generate compensation operation list and significantly reduce the number of device operations. The proposed models and algorithm are verified by testing results.**

***Index Terms*—transaction; large-scale; device collaborative system; compensation**

I. INTRODUCTION

In the modern world, in the area of AMS (area management system), and lighting control system, large-scale area devices collaborative technology become the supporting technology in these areas. Take the project that the author of this article and the research team participated for example, in the 2008 Olympic central area of landscape lighting control system, more than 500 lighting devices implements control and management of about 20,000 lights and lamps[1], and well-designed lighting effects has characteristics of aesthetic and artistic, by use of large-scale devices collaboration, landscape lighting has become an important part of the celebration in the Olympic central area.

In addition, with the widely used of wireless sensors in our life, the "internet of things" [2] has become the research hot spot. Lots of companies, research institutes put in a mass of human resources and material resources in this area, such as "Smarter Planet"[3] which is proposed by IBM. Lots of sensors connected with Internet, and a huge network is formed, so it makes devices easy to collaborate with other devices, and large-scale area devices collaboration becomes more and more universal.

A typical example is security system; it will describe the methods of different devices and wireless sensors collaboration: Sensors recognize somebody is crossing the warning line, they will active broadcast devices, and warning message is broadcast. If the suspect keeps moving closer, the alarm devices in the security room are active to inform security guards. If security guards confirm the alarm, the microphone is active so the guard can talk to the suspect and cameras around will turn to the suspect so that the guard can see more clearly. If the suspect leaves the warning line voluntary or security guards cancel the alarm, all devices will back to normal status.

It can be seen that, a large-scale area devices collaborative process requires a large number of devices to participate. Device operations will be done indivisibly; either fully implemented or compensated the operation impact. So, collaborative process has transactional characteristics.

Transaction processing technology is used to ensure reliability and consistency of information. Transaction is widely used in database system and workflow system. The transactional characteristic in large-scale area devices collaborative system is quite different from the one in database system or workflow system.

1. In the traditional transaction model, ACID[4] properties are too strict, for example, once a device operation is completed, the result is representing on equipment status, and the isolation cannot be achieved.

2. The duration of traditional transaction is a short, such as in database systems, a transaction is often the millisecond-level operations, and lock-mechanism is effective. The device collaborative process is long-lived transaction. Lock the device exclusively is inefficiencies.

3. In the traditional transaction model, compensation operation is done in reverse order of the execution order. At the same time, compensation operation is appointed by the user, and it is usually complicated. In device collaborative process, there is no need to compensate each operation; a convenient compensation is to set the device status according to the initial status. For example, in a lighting collaborative process, the loop is "ON" at the beginning. Then the transaction begins, "OFF, ON, OFF", a total of three operations is done on the loop. We can do an "ON" operation to achieve the compensation, rather than three operations of "ON, OFF, ON".

At present, the demand on transaction for device collaborative system is high, but relevant research is scarce. In order to meet the need for transaction in device collaborative system, in this paper, a new transaction model is proposed, which offers loose transaction properties. We propose a mechanism to calculated compensation operation, which will reduce the complexity of users. We also describe a scheduling policy for device collaborative task, which can support transaction characteristics and can compensate automatically when abort transaction.

doi:10.4304/jcp.5.8.1204-1212

This paper is organized as follows. Section 2 recalls related work about transaction model and transaction system. Section 3 presents the transaction model. Section 4 introduces a compensation operation calculated algorithm for device collaborative task. Section 5 gives an experiment and performance evaluation. We draw our conclusions in the Section 6.

II. Related Work

Transactions can provide reliable information processing in automated information systems. The amount of published research work on transactions is huge and a number of researches already exist, such as flat transactions, extended transaction and workflow transactions, etc.

Flat transactions have proven to be very useful in traditional database applications where the execution time is relatively short, the number of concurrent transactions is relatively small and transactions execute on only one database system. But it lacks the ability to support applications requiring long-living and/or complex transactions, and some advanced transaction models are proposed, such as sagas[5], WIDE[6].

Sagas were proposed which include a compensation mechanism to roll back already completed sub-transactions. Sagas divide a long lasting transaction into sequentially executed sub-transactions and each sub-transaction, except the last one, has a corresponding compensating sub-transaction. All these sub-transactions are atomic with ACID properties. The basic idea of the saga model is to allow a transaction to release resources before committing. A long lived transaction, or saga, is seen as a sequence of sub-transactions that can be interleaved in any way with other transactions. If the saga aborts then sub-transactions that have committed must be compensated. Thus, each sub-transaction has a compensating transaction associated with it, which undoes any changes introduced by the sub-transaction but does not necessarily return the database to the state it was before the sub-transaction was executed. More formally, let $T_1, T_2, ..., T_n$ be the sub-transactions of a saga T. Let $CT_1, CT_2, ..., CT_n$, be the corresponding compensating transactions. The system provides the following guarantee: either the sequence $T_1, T_2, ..., T_n$, is executed, or the sequence $T_1, T_2, ..., T_j, CT_j, ..., CT_2, CT_1$, for some $0 \le j \le n$, will be executed.

WIDE model combines the concept of safe point with Sagas, so that more flexibility is offered in compensation paths in case of exceptions. In the WIDE project, an extended workflow model and language are developed with advanced process primitives like multi-tasks, various join operators, exceptions, etc. A multi-level process model is used that allows for hierarchical decomposition of workflow processes with flexible transactional semantics. The top level of a process hierarchy is formed by a complete workflow process. The bottom level consists of individual tasks, i.e. process parts that are not further decomposed in the workflow specification. In this transaction model, the upper layer is formed by global transactions providing the relaxed transactional semantics of process layers above business transactions and the lower level by local transactions providing the strict transactional semantics of business transactions.

Workflow model supports a variety of business process logic for coordination of independent tasks [7]. An important point to note is that workflow models do not deal with the actual application semantics, i.e., the semantics of the activities is orthogonal to the workflow process. As a result, workflow models cannot be directly used to implement transaction models based on semantics or internal operations of the transactions. However, the semantics of such models are difficult to translate to workflow applications where most activities are not transactional in nature. Transactional workflows can be regarded as the unification of general workflows and transactions; it is motivated to support both transactional property and business process of an application [8].

ConTract model addresses the transactional challenge for long-lasting and complex computations in a formal basis and is aimed at the problem domain of large distributed applications. Although the ConTract model is not labeled as a workflow transaction model it is applicable in the workflow transaction area.[9][10]

There are also some researches in the area of pervasive computing [11] . The long-term goal of pervasive computing is to build large-scale smart environments that provide adequate services for users, and making computation invisible to us. Context-aware computing plays a key role to achieve this goal. Context-aware applications are driven by contexts which are collected from environments by sensors or other devices automatically.

However, most of these research are developed from database area, where preserving the data consistency of the shared database by transactional method is the main concern. These models are not suitable for device collaborative system because collaborative systems are far different from traditional ones simply based on databases.

III. Transaction Model and Definition

An example can be given to explain why we need to relax the ACID properties. A transaction in lighting control system is including 2 operations: turn on the 1st light and turn off the 2nd light. First, we turn on the 1st light, and when we turn on the 2nd light, something wrong happens, and we fail. The execution result is representing on lighting device's status: the 1st light is on. So what we can only do is to compensate the impact of the first step operation. We can turn off the 1st light, then the status of the 1st light and the 2nd light resume as nothing has happed.

Even a familiar scene is more persuasive: First, we turn on the 1st light, and when we turn on the 2nd light, somebody turn off the switch of the 1st light compulsorily, so, even we can turn off the 2nd light correctly, the result of the device status is not what the user want.

It can be seen that: in device collaborative system, strict ACID is not suitable, we need to relax the ACID

properties to meet the demand in device collaborative system.

The transaction model which offers loose ACID properties in device collaborative system is defined as follows:

1) **Atomic**

"All-or-none" atomicity is relaxed to the atomicity of "meet the needs of applications and users", meaning that, the transaction will end in the following 3 types:

a). the execution of collaborative processes complete normally;

b). the execution complete but some operations are redone;

c). the execution complete with compensation, all the impact of device operations is withdrawal.

2) **Isolation**

Because of the sharing and collaboration demands, the intermediate results of concurrent execution can be seen outside the transaction. And in order to increase the concurrency degree, access the device at the same time is valid. The strict isolation is relaxed to "the shared isolation".

3) **Consistency**

Relaxed isolation could undermine the data integrity and consistency, it does not require the system is in a consistent status all the time, but the concurrent scheduling can ensure that even if there is inconsistency, the status is still recoverable, and that "recoverable consistency".

4) **Durability**

When a device operation is successfully completed, the device status will be persistence changed, so durability is natural supported.

In this paper, "if and only if" is denoted by "⇔".

Definition 1: Device model

A device is composed of a number of terminals; a terminal has several kinds of status.

Let "d" a device; let "$c_1, c_2,\ldots,c_m$" are the terminals of device "d".

Let "cq_i" is the status of "c_i". $1\le i\le m$. $d=\{c_1,c_2,\ldots,c_m\}$.

Let $\sum$ the set of device operations.

Let Q the set of device status.

Definition 2: Device status

A device status "q" is defined as ordered couples of all the terminals status. $q=<cq_1,cq_2,\ldots,cq_m>, q\in Q$.

Definition 3: Basic terminal operation

A basic terminal operation "tp" is an operation that operates on a terminal. A basic terminal operation will only change one terminal status. $tp=< c_i ,cq_i >$, where "c_i" is a terminal of device "d", "cq_i" be the status of "c_i".

Definition 4: Device operation

A device operation "p" is defined as a set, $p\in\sum$. $p=\{tp_i,\ldots,tp_j\}=\{<c_i,cq_i>,\ldots,<c_j,cq_j>\}$, where tp_i is a basic terminal operation, c_i is a terminal, cq_j is the status of c_i.

If "p" is empty, it is defined as a read only operation. A read only operation will not change the status of the device.

If there is only one element in "p", it is a basic terminal operation.

If there is more than one element in "p", it is defined as a combined terminal operation. A combined terminal operation will change several terminals status concurrently.

A device operation is executed by the device itself, and it support strict ACID.

Definition 5: Device operation function

A device operation function is defined:

$$\delta := \sum \times Q \to Q \tag{1}$$

$q_n=\delta(q_{n-1}, p_n)$, where q_{n-1}, q_n are two status of a device, p_n is a device operation.

It predicates that the device change status from q_n to q_{n+1} because of device operation p_n.

Let α a device operation sequence, $\alpha=<p_1,\ldots,p_{n+1}>$, where $p_1,\ldots,p_{n+1}$ are n+1 device operations.

If $\alpha=<>$, it is defined as an empty sequence. An empty device operation sequence is denoted by ϕ.

Use (1),

$$\begin{aligned} q_{n+1} &= \delta(q_0,\alpha) \\ &= \delta(q_n, p_{n+1}) \\ &= \delta(\delta(q_{n-1}, p_n), p_{n+1}) \\ &= \underbrace{\delta(\delta\ldots(\delta}_{n+1}(q_0, p_1),p_2)\ldots, p_{n+1}) \end{aligned} \tag{2}$$

Definition 6: Equivalent device operation sequence

Let q_0 the initial status of a device, let q_{n+1} and q_{m+1}' two status of a device.

Let $\alpha=<p_1,\ldots,p_{n+1}>$, $\alpha'=<p_1',\ldots p_{m+1}'>$ are two device operation sequences.

$q_{n+1}=\delta(q_0,\alpha)$, $q_{m+1}'=\delta(q_0,\alpha')$, Use (2),

$$q_{n+1} = \underbrace{\delta(\delta\ldots(\delta}_{n+1}(q_0, p_1),p_2)\ldots, p_{n+1})$$

$$q'_{m+1} = \underbrace{\delta(\delta\ldots(\delta}_{m+1}(q_0, p_1'),p_2')\ldots, p'_{m+1})$$

$$\alpha \sim \alpha' \Leftrightarrow q_{n+1} = q'_{n+1}. \tag{3}$$

Definition 7: Compensation operation sequence

Let α, β are two device operation sequences.

For any device operation sequence $\theta\in\Sigma^*$, use (3), if $<\theta,\alpha,\beta>\sim<\theta>$, we call β is the compensation device operation sequence for α, denoted by

$$\beta = \alpha^{-1}. \tag{4}$$

IV. COMPENSATION OPERATION CALCULATED ALGORITHM

In traditional transaction system, if the user wants to compensate an operation, the user has to assign other operation. But in device collaborative system, device operations are too complicated, so the user is difficult to

know the compensation operations all the operations. When a transaction is abort abnormally, the transaction may not be compensated fully because of wrong compensation operations.

In this paper, we propose a mechanism to calculated compensation operation. With this mechanism, the user needn't to assign any compensation operations; the compensation procedure is done transparent.

Definition 8: Device operation conflict

When two device operations try to set a device to a different status, it will lead to a conflict. A conflict is denoted by "Θ".

A. Conflict detect algorithm

Rule 1: A read only operation will not conflict with other device operation.

Rule 2: Basic terminal operation conflict detect

Let tp be a basic terminal operation, tp={<c_i,cq_i>}

Let tp' be a basic terminal operation, tp'={<c_j,cq_i'>}

$$tp\Theta tp' \Leftrightarrow i = j \wedge cq_i \neq cq_j \quad (4)$$

Rule 3: Device operation conflict detect

Let tp be a basic terminal operation,t p={<c_i,cq_i>}

Let p' be a device operation, p'={ tp_1',…, tp_j'}

Use (4), rule is:

$$tp\Theta p' \Leftrightarrow \exists tp_j \in p', tp_j \Theta tp \quad (5)$$

Let p be a device operation, p={tp_1,…,tp_j}

Let p' be a device operation, p'={ tp_1',…, tp_j'}

Use (5), rule is:

$$p\Theta p' \Leftrightarrow (\exists tp_i \in p, tp_j \Theta p') \vee (\exists tp_j \in p', tp_j \Theta p) \quad (6)$$

Let d={c_1,c_2,…,c_m} be a device.

Let p_1={<c_i,tq_i>,…,<c_j,tq_j>} be a device operation of "d", 1≤i≤j≤n.

Let p_2={<c_x,tq_x>,…,<c_y,tq_y>} be a device operation of "d", 1≤x≤y≤n.

The conflict detection result (true or false) can be calculated by the following algorithm:

Algorithm 1: Device operation conflict detection algorithm

Input:

device operation: p_1

device operation: p_2

Output:

true(conflict) or false(not conflict)

Algorithm start

```
//Let p1=[<ci,tqi>,...,<cj,tqj>]
//Let p2=[<cx,tqx>,...,<cy,tqy>]
If( i>y or x>j)
  Return false;
//"min( )": get the minimum
e=min(j,y);
//"max( )": get the maximum
f=max(i,x);
For every elements <ca,tqa>( e≤a≤f )in A, do{
  For every elements <cb,tqb>( e≤b≤f )in B, do{
    If (ca == cb and tqa !=tqb)
      Return true;
  }
}
Return false;
```

Algorithm end

According to this algorithm, we assume that i≤x, and we consider 3 cases:

(1). 1≤i≤j≤x≤y≤n.

In this case, no operations are needed to be tested.

1	…	i	…	j	…	x	…	y	…	n

(2). 1≤i≤x≤j≤y≤n.

In this case, only terminal operations of c_x …c_j are needed to be tested.

1	…	i	…	x	…	j	…	y	…	n

(3). 1≤i≤x≤y≤j≤n.

In this case, only terminal operations of c_x …c_y are needed to be tested.

1	…	i	…	x	…	y	…	j	…	n

In order to speed up the conflict detection speed, the use of different devices to save the conflict detection matrix (CDM) to record the confliction.

A CDM is used to record a device confliction. Let "d" be a device, d={c_1,c_2,…,c_m}, p_1,p_2,…,p_n are device operations. CDM of d contains n rows, m columns. If $p_1 \Theta p_2$, CDM[i,j]=1, else CDM[i,j]=0.

An example of a CDM is shown in Table I.

B. Compensation operation calculated algorithm

Let d={c_1,c_2,…,c_m} be a device.

Let q_0={<c_1,tq_1>,…,<c_m,tq_m>} be the initial status of "d".

Let p_1 ,…,p_n be the device supported operations.

p_i =< tp_1 ,…, tp_{pi} >

Let α be an operation sequence. $q_1=\delta(q_0, \alpha)$,$\beta=\alpha^{-1}$.

β will be calculated according to the following algorithm:

TABLE I.
CONFLICT DETECTION MATRIX

	P_1	**P_2**	**P_3**	**P_4**	**P_5**	**P_6**	**P_7**	**P_8**
P_1={<1,1>,<2,1>,<3,1>}	0	1	0	1	0	1	1	0
P_2={<1,0>,<2,1>}	1	0	1	0	0	1	0	0
P_3={<1,1>}	0	1	0	1	0	1	0	0
P_4={<1,0>}	1	0	1	0	0	0	0	0
P_5={<2,1>}	0	0	0	0	0	1	0	0
P_6={<2,0>}	1	1	1	0	1	0	0	0
P_7={<3,0>}	1	0	0	0	0	0	0	1
P_8={<3,1>}	0	0	0	0	0	0	1	0

Algorithm 2: Compensation operation automatic calculated algorithm

Input:

initial device status: B

Output:

compensation operations list

Algorithm start

// Let "A" be the list of device operation.
A=[p_1 ,…,p_n];RA=[r_1 ,…,r_n]; r_1=…=r_n=0;
// Let "B" be the list of initial device status.
//B= [<c_1,tq_1>,…,<c_m,tq_m>];
//Let β be the compensation operations list.
β=[].
//Conflict detect
For every element p_x in A, do {
If (p_xΘ B){
A.remove(p_x);
RA.remove(r_x);
}
}
While B.length!=0 do{
//Calculate the number of elements of B occurs in A
For every element in B, do{
If (∃ <c_k,tq_k>∈ p_x and <c_k,tq_k>∈ B)
RA[x]= RA[x]+1;
}

//Remove useless elements in A
For every element p_x in A, do{
If (RA[x]==0){
A.remove(p_x);
RA.remove(r_x);
}
}

//Get an operation
Find Max r_y in RA;
//Only terminal operations left
If(max(r_y)==1){
β.Add(A);
Return β;
}

β.Add(p_y);
B.remove(every element in p_y);
RA=[0,…,0];
}
Return β;

Algorithm end

Because all basic terminal operations belong to A, the algorithm is terminable. In the worst case, β={<c_1,tq_1>,…,<c_m,tq_m>}.

C. Analysis of the Time Complexity of Algorithm 2

In the process of "*Conflict Detect*", it is necessary to detect every operation in A whether it is conflict with B. Assume device number is n, device operation number is m. The complexity of compute conflict is o(m*n). But if we use conflict detection matrix, the complexity is o(1). This fully indicates that CDM can significantly decrease the complexity.

In the process of "*Calculate the number of elements in B occurs in A*", because the number of terminal operation in a device operation is limited to less than a fix constant, in the best situation, we can finish the algorithm at the first loop, the computational complexity is o(m*n); in the worst situation, we can finish the algorithm after n loop, the computational complexity is o(m*n^2). Note that the list B is dynamic, so we can't decrease the complexity through a matrix like CDM.

In the process of "*Get a operation*", the computational complexity of the sort algorithm is o(m), so the computational complexity is o(m*n).

Thus, to sum up the points which we have just indicated, in the best situation, the total computational complexity is o(m*n); in the worst situation, the computational complexity is o(m*n^2).

Take the situation descript in Table.1 for example, assume that the initial status is: {<1,0>,<2,1><3,0>}.

At the beginning,:
A={ p_1 , p_2 , p_3 , p_4 , p_5 , p_6 , p_7 ,p_8 },
B={<1,0>,<2,1><3,0>}.
After conflict detection,
A={ p_2 , p_4 , p_5 , p_7 }.
First Cyc:
A={ p_4 , p_5 , p_7 };
B={<3,0>};
β={ p_2 }
Second Cyc:
A={ p_4 , p_5 };
B={};
β={ p_2 , p_7}
End
The compensation operation sequence is: <p_2 , p_7>

Using this algorithm, we can get an approximate solution, but it may not be an optimal solution. The algorithm is more efficient than exhaustive algorithm.

For example, the device operations P_1-P_9 are:
P_1={<1,1>,<3,1>,<4,1>,<5,1>}
P_2={<1,1>,<2,1>,<3,1>}
P_3={<4,1>,<5,1>,<6,1>}
P_4={<1,1>}
P_5={<2,1>}
P_6={<3,1>}
P_7={<4,1>}
P_8={<5,1>}
P_9={<6,1>}
The initial device status is: {<1,1>,<2,1>,<3,1>,<4,1>,<5,1>,<6,1>}, we use the algorithm 2:

At the beginning,:
A={ p_1 , p_2 , p_3 , p_4 , p_5 , p_6 , p_7 ,p_8, p_9 }.
B={ <1,1>,<2,1>,<3,1>,<4,1>,<4,1>,<5,1>,<6,1>}.
After conflict detection,
A={ p_1 , p_2 , p_3 , p_4 , p_5 , p_6 , p_7 ,p_8, p_9 }.
First Cyc:
A={ p_2 , p_3 , p_4 , p_5 , p_6 , p_7 ,p_8, p_9 };
B={<2,1>,<6,1>};
β={ p_1 }

Second Cyc:
A={ p_3 , p_5 , p_9};
B={<6,1>};
β={ p_1 , p_2 }
Third Cyc:
A={ p_9};
B={};
β={ p_1 , p_2 , p_3}
End

Actually, the operation list β'={p_2 , p_3} can compensation the initial device status. So the { p_2 , p_3} is the optimal solution, but if we want to get the β', the total computational complexity is enormous. By using of algorithm 2, lots of computation is reduced.

V. DEVICE OPERATIONS AND CONFLICT DETECT METHOD IN LIGHTING DEVICE COLLABORATIVE SYSTEM

A lighting device is composed of several lighting loops. A lighting loop command set includes turn-on and turn-off. For every loop, there are two kinds of status, ON and OFF. A lighting device supports three types' device control operations:

1. Loop control operation. A loop control operation is used to turn on or turn off a loop, and it is the basic terminal operation of a lighting device.
2. Group control operation. A group operation is used to turn on or turn off several loops concurrently.
3. Pattern control operation. A pattern operation is used to achieve a lighting scene. In lighting control system, a scene is implements by turn on several loops and turn off several loops.

In additional, there are three types of monitor operations, including loop monitor operation, pattern monitor operation and group monitor operation. The monitor operations are "read only" operations: these operations will not change devices' status, so in the following section, only control operations will be concerned, and no detail discussion about monitor operations will be given.

A. Lighting device model

The formal description of lighting device operations is given as follows:

Terminal of lighting device: lighting loop

Let "c" be a lighting loop, it has two statuses: ON and OFF, that is, $stat_c$=ON or OFF.

A loop supports two operations, loop on operation: on(c), and loop off operation: off(c).

Lighting device

Let "s" be a lighting device, s is expressed as a set of all the lighting loops, s={c_1 ,… , c_n},n≥1. Let c_i be a loop of s, 1≤i≤n. A lighting device is described as Fig.1:

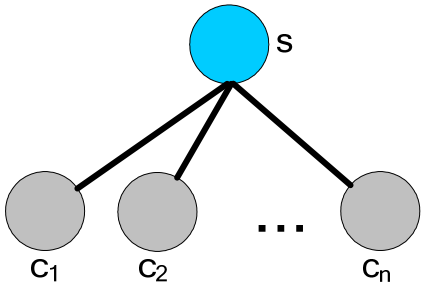


Figure 1. Lighting device model

The basic terminal operation of a lighting device

Loop control operation is the basic terminal operation of a lighting device, and loop control operation is denoted as *s.CCtrl.* It is described as:

$$s.CCtrl(c_i, onoff) = \begin{cases} on(c_i), onoff = ON; \\ off(c_i), onoff = OFF; \end{cases} \quad (7)$$

A loop control operation is described as Fig.2. It shows that c_2 will be turned on if *s.CCtrl* (c_2, *on)* is execute.

The group control operation of a lighting device

A lighting group "g" is defined as a series of loops which will be opened or closed simultaneously, g_m={c_i ,… , c_j},1≤ i,…,j ≤n.

A group control operation is denoted as *s.GCtrl*. It is defined as:

$$s.GCtrl(g_m, onoff) = \begin{cases} on(c_i) \| ... \| on(c_j), & onoff = ON; \\ off(c_i) \| ... \| off(c_j), & onoff = OFF; \end{cases} \quad (8)$$

In Table I, p_1 is considered as a group control operation.

Assume that group g_1={c_i ,… , c_n}.

A group control operation is described as Fig.2, it shows that c_i ,… , c_n will be turned on if *s.GCtrl* (g_1, *on)* is execute. This operation will turn on all lightings.

The pattern control operation of a lighting device

A lighting pattern is defined differently in different lighting control system, for example, in some systems a lighting pattern "p" is defined as the combination of all loops' statuses. But in some systems a lighting pattern "p" is defined as the combination of several loops' statuses, maybe not all the loops. In our experiments, we use the second definition. p={<c_x, $stat_{cx}$>,…,<c_y, $stat_{cy}$>}.

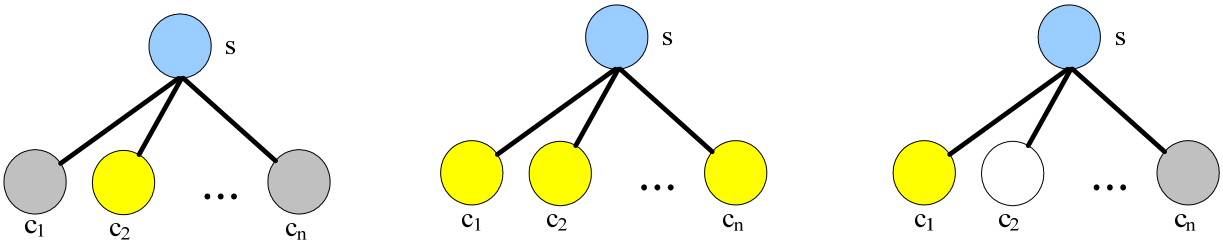
A lighting pattern control operation is denoted as *s.PCtrl.* It is defined as:

$$s.PCtrl = stat_{cx}(c_x) \| ... \| stat_{cy}(c_y); \quad stat_{cx}=ON \text{ or } OFF. \quad (9)$$

In Table I, p_2 is considered as a pattern control operation.

Assume that pattern p_1={<c_1, ON>,<c_2, OFF>}.

A pattern control operation is described as Fig.2. it shows that c_1 will be turned on and c_2 will be turned off and all the other loop will never be operated when *s.PCtrl* (p_1) is execute.



s.CCtrl (c_2, on) *s.GCtrl (g_1, on)* *s.PCtrl (p_1)*

Figure 2. Lighting device operation

B. The Conflict Detection method

According to definition 8, we can detect the conflict of lighting device operations by means of the following method.

An empty set is denoted by "ϕ".

1) Let op_1=*s.CCtrl (c_i, $onoff_1$)*.
Let op_2=*s.CCtrl (c_j, $onoff_2$)*.
Op_1 and Op_2 are terminal operation, so we use (4),(7):

$$op_1 \Theta op_2 \Leftrightarrow c_i = c_j \wedge onoff_1 \neq onoff_2$$

2) Let op_1=*s.CCtrl (c_i, $onoff_1$)*.
Let op_2=*s.GCtrl (g, $onoff_2$)*.
Op_1 is a terminal operation, and Op_2 is a device operation, so we use (5),(7),(8):

$$op_1 \Theta op_2 \Leftrightarrow onoff_1 \neq onoff_2 \wedge c_i \in g$$

3) Let op_1=*s.GCtrl (g_i, $onoff_1$)*.
Let op_2=*s.GCtrl (g_j, $onoff_2$)*.
Op_1 and Op_2 are device operations, so we use (6),(8):

$$op_1 \Theta op_2 \Leftrightarrow onoff_1 \neq onoff_2 \wedge g_i \cap g_j \neq \phi$$

4) Let op_1=*s.CCtrl (c_i, $onoff_1$)*.
Let op_2=*s.PCtrl (p)*.
Op_1 is a terminal operation, and Op_2 is a device operation, so we use (5),(7),(9):

$$op_1 \Theta op_2 \Leftrightarrow \exists < c_x, stat_{cx} > \in p \; c_x = c_i \wedge stat_{cx} \neq onoff_1$$

5) Let op_1=*s.GCtrl (g_i, $onoff_1$)*.
Let op_2=*s.PCtrl (p)*.
Op_1 and Op_2 are device operations, so we use (6),(8),(9):

$$op_1 \Theta op_2 \Leftrightarrow \exists < c_x, stat_{cx} > \in p, \; c_x \in g \wedge stat_{cx} \neq onoff_1$$

6) Let op_1= *s.PCtrl (p_i)*.
Let op_2=*s.PCtrl (p_j)*.
Op_1 and Op_2 are device operations, so we use (6),(9):

$$op_1 \Theta op_2 \Leftrightarrow \exists < c_i, stat_{cx} > \in p_i, < c_i, stat_{cy} > \in p_j, \wedge stat_{cx} \neq stat_{cy}$$

VI. Experiment and Performance Evaluation

In order to verify the performance of the algorithm, experiments are done. In the experiment, we take lighting device control system project as the research object, and we use lighting devices as the study case.

During the experiments, we use the following parameters on the system configuration:

1. TN: The number of terminals.
2. SN: The number of terminal status.
3. DN: The number of device operations.
4. PN: The number of parallel tasks.

The experiment is done on a PC, with: CPU Intel Core 2 2.0GHz, memory 1G; Windows XP Professional with Service Pack 3.

During the experiments, we adjust the parameters to evaluate the performance of the algorithm at different situations. The experiments are divided into three parts:

In the first part of experiments, we research the relation between terminal number and the time that calculates the compensation operations.

In the second part of experiments, we research the relationship between numbers at worst situation and the actual numbers.

In the third part of experiments, we research the performance in multi-task environment. In the figure, all experiment data are shown in hash point.

The first part of the experimental results is shown in Fig.3. It can be seen from the experimental data that, with the increase of terminal number, the calculation time of compensation operations rises gradually. The parameters in this experiment are:

SN=2,
PN=1,
DN=TN*4+TN*SN.

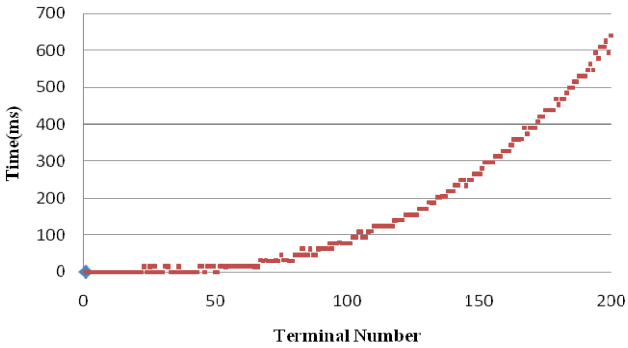


Figure 3. Terminal number vs Time

The second part of the experimental results is shown in Fig.4. The abscissa is the number of terminals. On worst situation, in order to finish the compensation, every terminal has to do a basic terminal operation. The longitudinal coordinates the number of actual operations. As can be seen from Fig.4, compensation operation automatic calculated algorithm can significantly reduce the number of device operations. The parameters in this experiment are:

SN=2,

PN=1,

DN=TN*4+TN*SN.

In the third experiment, calculate the performance of the algorithm through multiple tests. The parameters in this experiment are:

TN=100,

SN=2,

DN=TN*4+TN*SN.

Experimental data in Table II show that: In the case of multi-task, the algorithm performance degradation is not obvious.

VII. Conclusion

In this paper, we propose a new transaction model, which loose the requirement of ACID. We also give device operations conflicts detection rules, and a mechanism to calculate compensation operation is proposed. As can be seen from the experiment data, compensation operation automatic calculated algorithm can significantly reduce the number of device operations. And even when the number of terminal is 200, the calculation time is about 650ms, the algorithm performance is good. In the case of multi-task, such as 400 tasks concurrence running, the algorithm performance degradation is not obvious.

The proposed algorithm can effectively reduce the complexity of the user to design a device collaborative transaction, because users do not need to care about how to compensate for operation. With this mechanism, the compensation procedure is done transparent.

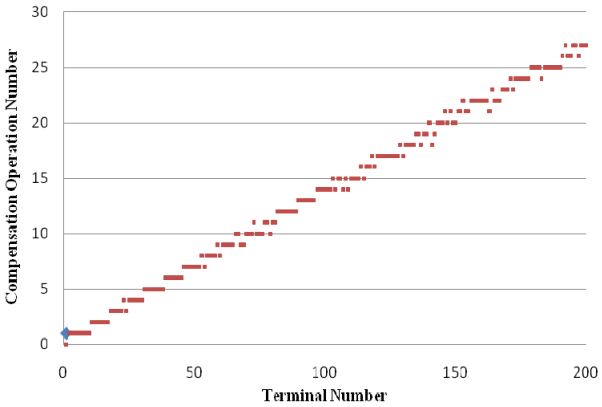


Figure 4. Operation number: worst vs actual

TABLE II.
Multi-task performance

PN	1	50	100	200	300	400
Time(ms)	78	112	128	145	155	164

Acknowledgment

The work was supported by National Key Technology R&D Program in China (NO. 2006BAK12B11).

References

[1] F. Chen, X. Rong and P. Deng, “A Large-Scale Device Collaborative Process Design Meta-Model and Case Study,” The 2nd International Conference on Advanced Computer Theory and Engineering (ICACTE 2009), Egypt, pp. 601-608, Oct 2009.

[2] N. Gershenfeld,R. Krikorian and D. Cohen, “The Internet of Things”, Sci Am, vol. 291, no. 4, 2004, pp. 76-81.

[3] S. Palmisano, “A Smarter planet: instrumented, interconnected, intelligent”, http://www.ibm.com/ibm/ideasfromibm/us/smartplanet/20081117/sjp_speech.shtml, 2008

[4] J. Gray and A. Reuter, “Transaction processing: concepts and techniques”, Morgan Kaufmann Pub, 1993.

[5] H. Garcia-Molina and K. Salem, “Sagas”, SIGMOD, ACM, 1987, pp. 249-259.

[6] P. Grefen,J. Vonk,E. Boertjes and P. Apers, “Two-layer transaction management for workflow management applications”, Lect Notes Comput Sci, 1997, pp. 430-439.

[7] D. Hollingsworth, “The workflow reference model”, Workflow Management Coalition, 1995.

[8] G. Alonso,D. Agrawal,A. El Abbadi,M. Kamath,R. Günth r and C. Mohan, “Advanced transaction models in workflow contexts”, institute of electrical and electronics engineers, 1996, pp. 574-583.

[9] T. Wang,J. Vonk,B. Kratz and P. Grefen, “A survey on the history of transaction management: from flat to grid transactions”, Distrib Parallel Dat, vol. 23, no. 3, 2008, pp. 235-270.

[10] H. W chter and A. Reuter, “The contract model”, Database transaction models for advanced applications, vol. 7, no. 4, 1992, pp. 219-263.

[11] S. Chen, J. Ge, X. Tao, et al. A Transaction Model for Context-Aware Applications [J]. Lecture Notes in Computer Science. 2007, 4459: 252.

Chen Feng (1981 -) is currently PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China.

His research interests include Grid, Workflow, Large-scale device collaboration.

Xiaohui Rong (1982 -) is currently PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China.

Her research interests include Grid, Large-scale device collaboration.

Deng Pan (1983 -) is currently PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China.

Her research interests include Grid, Uniform High-level Communication Protocol, Large-scale device collaboration.

Shilong Ma (1953 -) is Ph.D. professor, doctoral supervisor, standing deputy director of the State Key Laboratory of Software Development Environment in the school of computer science and engineering in Beihang University, member of the 10th expert appraisal panel under Department of Information Science of the National Natural Science Foundation Committee, member of executive Committee of Asian Software Fundamental Federation, standing director of China Artificial Intelligence Society.

His research interests include Grid, Computational Logic, Magnanimity Information Processing, Data Mining.