A New Concurrent Mechanism for Distributed Hardware-In-The-Loop Simulation Test System

Mao Ye¹, Wei Dong², Yindong Ji³, Shuai Wang³
Tsinghua National Laboratory for Information Science and Technology
Department of Automation; Tsinghua University
Beijing, 100084, China
¹xiem03@mails.tsinghua.edu.cn; ²{weidong,jyd}@tsinghua.edu.cn; ³wangshuai04@mails.tsinghua.edu.cn

Keywords: Hardware-in-the-loop systems, System concurrency, Resource decomposition, Sharing locks.

Abstract
By simulating the operation environment, many device characteristics can be tested in Hardware-In-The-Loop (HIL) simulation systems and time cost determines the value of such project. Conventional task decomposition, scheduling and resource sharing mechanisms could no longer fully exploit the concurrency in complex HIL simulation systems since complex and stateful devices are integrated. A novel concurrent mechanism is proposed in this paper to solve this problem. In this mechanism, by decomposing the testing resources of complex devices in the simulation environment hierarchically, modeling and managing the resource working and occupation status, (1) test operations can work at finer granularities; (2) independent tasks can hold sharing resource access locks and execute on same devices simultaneously; (3) the logic correctness of real devices and simulation system can be maintained. By using this new mechanism, the effect of prevailing task scheduling algorithm can be extended. A Simple simulation experiment demonstrates the new mechanism improves the time cost about 20% under same test conditions.

1. INTRODUCTION
Hardware-in-the-loop (HIL) simulation test system is widely used in the verification and validation of both hardware devices and software. For example, a distributed Simulation Test System for Chinese Train Control System (SimCTCS) integrates many railway devices and software models (Hereafter they are not distinguished since they appear nothing different for the exterior simulation environment) and executes the required test cases to verify the correctness and robustness of devices, communication protocols, scheduling mechanisms, and etc [1].

Since HIL systems like SimCTCS are designed for test, the time and cost determines the system quality. Now, the application of concurrent test is proven [2][3][4] to be a useful tool to significantly save test cost in HIL test systems. However, prevailing concurrent mechanisms, which are mainly focusing on scheduling tasks on different resources or memory blocks, are no longer powerful enough for the more and more complex systems. For example, Interlock (IL) in SimCTCS manages a set of working and occupation status of tracks in the station. It could regulate the status of multiple track pairs simultaneously. If common test cases would only order one IL to regulate just one track pair at one time, system concurrency is wasted. In other words, manipulating independent tasks execute on same resource concurrently becomes a new requirement.

This “new” requirement in HIL test scenario is not new in the area of computer science. It is often regarded as “task decomposition” and “resource partitioning”. However, current solutions are not applicable for HIL test systems due to their uniqueness.

Take the SimCTCS as an example. A critical problem of supporting concurrent test in SimCTCS is that many of the SimCTCS resources are “stateful” rather than “stateless”. This means that whether or not a certain test action is eligible to be executed depends on not only the occupation of required resources, but also the possible current working status of these resources. Therefore, the concurrent mechanism should be able to model and maintain the resource working status effectively. Based on the working status and resource occupation constrains, the concurrent mechanism could decide how to allocate resources to specific test actions.

To achieve this goal, this paper proposes a novel concurrent mechanism which sets up a new concurrent operation model in HIL simulation test systems. In this mechanism, resources of each under test units (UTUs) are modeled and decomposed hierarchically and corresponding working status transferring matrix is represented; test actions and requirements are defined; specific algorithms are designed to effectively maintain the resource status. By doing this, independent tasks can be executed in the same device while maintaining the logic correctness.

The rest of this paper is organized as follows. Section 2 introduces some research results about resource partitioning; section 3 discusses how to model the HIL system; section 4 presents the basic design of the new mechanism including critical algorithms and constrains; section 5 takes a very simple SimCTCS abstract model as the example to illustrate the effectiveness and correctness of this new mechanism; in section 6, we conclude our work and point out some possible research directions.
2. RELATED WORK

Researchers in computer science area have carried out lots of excellent results about the requirement of resource sharing. Half of them focus on decomposing tasks to achieve finer operation granularities; and the other half focus on partitioning resources so that more parallel actions can be executed simultaneously.

There are several ways to do task decomposition [5], such as flat decomposition, hierarchical decomposition and so on. These ways share a same characteristic, i.e., they all focus on operate the test actions at a finer granularity. By doing this, the required amount of test resources is decreased, thus the overall resource confliction is decreased and boost the general performance. It is worthy to note that what is decomposed by these methods is the task, rather than the resource. In other words, these methods ask the test designers to modify the tasks to boost the test performance. This direction is valuable, especially when such methods are combined with certain task scheduling algorithms [6], such as Simulated Annealing, Tabu, and so on. But they are not capable to fulfill the potential concurrency of complex HIL simulation since they do not regard the concurrent operation capability of real devices.

Works about resource partition did focus on splitting general resources. Mature results are common in the area of simultaneous multithread testing (SMT) [7]. However, the partitioned resources are homogenous and stateless, such as cache, processor, pipelines and so on. There is no special algorithm or inner components to maintain the resource working status. In addition, the occupation of resources in these mechanisms cannot be partially occupied, thus the operation granularity of tasks has great impact on the general system performance.

To solve these problems, this paper demonstrates the way to model the HIL test actions, resources, and constrains; in addition, it shows the decomposition scheme of resources and how to maintain occupation and working status of resources. Details about the implementation of system architecture or action merge methods are not discussed since they highly depend on the specific application.

3. SYSTEM MODELLING

In this section, we present the mathematical models of testing actions and resources in HIL simulation test systems. Commonly, testing actions are some sets of code that are sent to the specific processors for execution. There must be some test controller to verify whether the actions are eligible to be executed. Also, a corresponding mechanism is needed to carry out the predetermined execution.

The following part is only the fundamental concepts related to testing actions and resources are modeled in this paper, because the detailed implementation depends on the specific application requirement.

Table 1. Terminology

<table>
<thead>
<tr>
<th>No.</th>
<th>Notation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A&lt;sub&gt;j&lt;/sub&gt;</td>
<td>Action j.</td>
</tr>
<tr>
<td>2</td>
<td>R&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Resource i.</td>
</tr>
<tr>
<td>3</td>
<td>R&lt;sub&gt;i&lt;/sub&gt;.occ</td>
<td>Present occupation status of resource R&lt;sub&gt;i&lt;/sub&gt;.</td>
</tr>
<tr>
<td>4</td>
<td>R&lt;sub&gt;i&lt;/sub&gt;.sta</td>
<td>Present working status of resource R&lt;sub&gt;i&lt;/sub&gt;.</td>
</tr>
<tr>
<td>5</td>
<td>A&lt;sub&gt;j&lt;/sub&gt;.pri</td>
<td>The non-negative priority value of Action j.</td>
</tr>
<tr>
<td>6</td>
<td>Null</td>
<td>Nothing</td>
</tr>
<tr>
<td>7</td>
<td>A&lt;sub&gt;j&lt;/sub&gt;.Q</td>
<td>The necessary resources to execute action A&lt;sub&gt;j&lt;/sub&gt;.</td>
</tr>
<tr>
<td>8</td>
<td>LE&lt;sub&gt;j&lt;/sub&gt;</td>
<td>The life expectation of action A&lt;sub&gt;j&lt;/sub&gt;.</td>
</tr>
<tr>
<td>9</td>
<td>Rd</td>
<td>Some action performed “Read” operation to a resource.</td>
</tr>
<tr>
<td>10</td>
<td>Wr</td>
<td>Some action performed “Write” operation to a resource.</td>
</tr>
<tr>
<td>11</td>
<td>Part_Rd</td>
<td>Some action performed “Read” action to PART of this resource.</td>
</tr>
<tr>
<td>12</td>
<td>Part_WR</td>
<td>Some action performed “Write” action to PART of this resource.</td>
</tr>
<tr>
<td>13</td>
<td>ε</td>
<td>A time parameter that determines period time length.</td>
</tr>
<tr>
<td>14</td>
<td>A ← B</td>
<td>(Assign) The value of B is given to A.</td>
</tr>
<tr>
<td>15</td>
<td>A = B</td>
<td>(Equal) The value of A equals to B.</td>
</tr>
</tbody>
</table>

Definition 1 (Action). Action j in the test system are presented by the symbol A<sub>j</sub>. Symbol “null” is designed to represent that this symbol stands for nothing.

\[
A_j = \begin{cases} 
Null & \text{if } A_j \text{ stands for nothing} \\
\text{Else} & \text{ if } A_j \text{ stands for a certain test action}
\end{cases}
\]

Definition 2.1 (Resource Occupation). If resource R<sub>i</sub> is already “Occupied” by action A<sub>j</sub>, we call the occupation status of resource R<sub>i</sub> “Busy” until this action A<sub>j</sub> releases it. Otherwise, we call the occupation status of resource R<sub>i</sub> “Idle”.

\[
(R_i.\text{occ} = A_j) \cap (A_j \neq \text{Null}) \quad R_i \text{ is busy} \\
(R_i.\text{occ} = \text{Null}) \quad R_i \text{ is idle}
\]

Define 2.2 (Resource Working Status). All possible

Figure 1. Graph of Resource Decomposition Scheme
current working status of each resource is concluded into different sets of working modes. For any specific device, all its possible working status is designed as follows.

\[ \text{Status}_i = \{\text{mode}_{i1}, \text{mode}_{i2}, ..., \text{mode}_{ik}\} \]

\[ \forall \text{mode}_k \in \text{Status}_i, R_i. \text{sta} = \text{mode}_k . \] This means currently, resource i is working as mode k.

**Definition 3.1 (Resource Requirement).** The implementation of action j might need the occupation of a series of resources. This relation is defined as follows.

\[ Q_j = A_j; Q = \{R_1, R_2, R_3, ..., R_n\} \]

In A_j, Q, each member R_i stands that this resource is required by the execution of action A_j.

Some actions might have constrainings on the working status of resources. We define the set of eligible working status as p_i, thus we can write this relation as follows.

\[ \text{Precond}_j = A_j; \text{Precond} = \{p_1, p_2, ..., p_n\} \]

In this equation, \( \forall p_i \in \text{Precond}_j \), \( p_i \in \text{Status}_A_j[i] \).

**Definition 3.2 (Operation Type).** The outcome of execution of action A_j can be modeled as the change to the working status of some resources. The types of specific operations this action A_j will perform on each of its required resource are defined and then form a set T_j.

\[ T_j = A_j; T = \{\text{type}_1, \text{type}_2, ..., \text{type}_n\} \]

In this equation, \( \forall \text{type}_i \in T_j, \text{type}_i \)

\[ \{R_d \text{ if } A_j \text{ will perform a Read operation on resource } Q_j[i] \]  
\[ \{W_r \text{ if } A_j \text{ will perform a Write operation on resource } Q_j[i] \]  

**Definition 4 (Status Transition).** Based on the definition of resource working status and actions, we may define the status transition function of resource R_i as follows.

\[ R_i. H(A_j, \text{mode}_{in}) = \text{mode}_{out} \]

This transition function has the following characters.

**Characteristic 4.1 (Closeness).** For any action A_j and any resource R_i, as long as R_i \( \in A_j \), Q, after the execution of A_j, the working status of resource R_i still belongs to the set Status_j.

\[ \forall A_j, \forall R_i \in A_j, Q, R_i, H(A_j, R_i, \text{sta}) \in \text{Status}_i \]

**Characteristic 4.2 (Uniqueness).** For any action A_j, and any eligible working status \( \text{mode}_{in} \) of resource R_i, as long as R_i \( \in A_j \), Q, there is one and only one \( \text{mode}_{out} \) equals to the transition function result.

\[ \forall A_j, \forall \text{mode}_{in} \in A_j; \text{Precond}[i], \exists \text{only } \text{mode}_{out} \]

\[ \in \text{Status}_i, R_i, H(A_j, \text{mode}_{in}) = \text{mode}_{out} \]

The construction of specific mapping between actions and working status of resources depends on the specific application. Therefore, the sufficient condition for an action A_j to be executed immediately is defined as follows.

**Definition 5 (Firability).** If all expected resources of a certain action A_j are not occupied or occupied by this action itself, and the corresponding working status meets the requirement of this action, this action A_j can be executed immediately. In other words, this action A_j fully meets the action compatibility and resource requirement constrains. Thus we mark the status of A_j in this condition as A_j is “Firable”.

\( \text{(Firable}(A_j) = \text{true}) \]

\[ \iff (\forall R_i \in A_j, Q, (R_i. \text{occ} = A_j) \cup (R_i. \text{occ} = \text{Null}) \) “OC” \]

\[ \iff (\forall R_i \in A_j, Q[i], R_i. \text{sta} = A_j. \text{Precond}[i]) “WC” \]

Here “OC” means “Occupation Constrains”, while “WC” means “Working Status Constrains”. Both the “OC” and “WC” need to be satisfied.

### 4. RESOURCE DECOMPOSITION AND STATUS MANTAINANCE

Usually, a certain test script has its focus and range. Thus the following two cases are so common: (1) not all hardware devices will take part in a same test simultaneously; (2) not all concurrent operation capability of a certain device is fully applied in a same test simultaneously. These facts contribute to the main concurrency capability of the HIL system. For example, in SimCTCS, IL operates independently. It receives commands or requests from a fixed command buffer and process them every 250 milliseconds. It is critical that the command buffer actually is capable of storing multiple compatible commands. Therefore, making the IL only deal with one command in every period is a kind of wasting its concurrent operation capability.

However, making the same device participate in independent tasks will often lead to resource conflictions, and then cause logic failure to the system. Making test operations execute at a finer granularity is a better way to

![Figure 2. SimCTCS Distributed Architecture](image-url)
4.1. Resource Decomposition

Since the actions in SimCTCS will operate at various granularities, the system should support sharing lock at various granularities. In this paper, the term “hierarchical” is applied to represent the decomposition and administration architecture of resources or sub-resources belonging to a same device. By organizing the fine granularities hierarchically, the system could easily get the action working status and relative parameter information. See the Fig. 1 for an illustration.

First of all, for any device which could support concurrent operation, its resource is split into multiple non-overlapping resource cells. These cells are also called the “leaf” resource unit. The granularity of the resource cell is the finest one so that no resource cell can be partly occupied.

Second, according to the specific device characters, some cells are grouped into a higher level resource unit. Also, these units are non-overlapping. These resource units permit partial occupation. Meanwhile, they also support the occupation on this level.

Third, the process of resource grouping continues until there is only one resource unit in the highest level. From up to down, the levels are named as “level-1”, “level-2”, and etc. The only one “level-1” resource unit is also called the “Root”. In other words, the resource of any device is decomposed and organized as a “Device Resource Tree (DRT)”.

Any none “leaf” resource units maintain the “Abstract” occupation status of all their individual children units. Meanwhile, the “Detailed” occupation status of any unit is maintained by this unit itself. Since the working status of each device is unable to decompose, the working status of each device, including that of each decomposed resource unit, is maintained by the “Root” node of each DRT.

Table 2. Occupation Status Maintenance

<table>
<thead>
<tr>
<th>Occupation Description</th>
<th>Occupation Status</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>All “None Occupation” (NO) in the same level</td>
<td>Parent: Self:</td>
<td>Null: Null</td>
</tr>
<tr>
<td>Some “Read” or “Part_Rd” in the same level</td>
<td>Parent: Self:</td>
<td>Part_Rd: Rd,Part_Rd</td>
</tr>
<tr>
<td>Some “Write” or “Part_Wr” in the same level</td>
<td>Parent: Self:</td>
<td>Part_Wr: Wr,Part_Wr</td>
</tr>
</tbody>
</table>

4.2. System Architecture

The decision of resource access lock assignment is made by a centralized unit. The simplicity of the centralized coordinator-based architecture could help save a lot of time in preventing deadlock and designing complex distributed algorithms, thus it is applied in this paper. When the coordinator (hereafter also named “Brain”) has made its decision, it sends the result to distributed resources. When the action is finished in each resource place, a “release” command is sent back to the “Brain”. The system architecture is illustrated as Fig.2, and the meanings of each number in Fig. 2 are defined in Table 3.

Table 3. Definition of Numbers in SimCTCS Architecture

<table>
<thead>
<tr>
<th>No.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The test controller sends a new command for (a) new action(s).</td>
</tr>
<tr>
<td>2</td>
<td>The coordinator buffers this command. Based on the coming status, the coordinator maintains the global resource occupation status.</td>
</tr>
<tr>
<td>3</td>
<td>When some buffered action can be executed, the coordinator sends this action to the best location with resource lock.</td>
</tr>
<tr>
<td>4</td>
<td>The local processor deals with the coming action and its resource lock. After all test commands for this device is merged, this combined action will be executed in the next period.</td>
</tr>
<tr>
<td>5</td>
<td>After the action is finished, all occupied resource is freed. Also, a “Release” command is sent back to the coordinator.</td>
</tr>
</tbody>
</table>

4.3. Status Maintenance Algorithms Design

This whole mechanism consists of three main modules, each of which has its own special algorithm. Fig. 3 illustrates the working relationship of each module.

The first module is named “Request Switch”, which deals with the all input requests/commands for resource access lock. It plays as a “Switch”, executes the commands which are “Firable” while puts those which are NOT “Firable” into the queue.

The second module is named “Resource Releaser”, which is responsible for the release work of resource access locks. It also maintains the working status of each device.

The third module is named “Watch Dog”, which operates periodically (T = c) and helps maintain the system inner status consistency and correctness. It pops out selected actions from the queue according to certain task scheduling algorithm and send them to the “Request Switch”.

4.3.1. “Request Switch” Design

**Place:** This module locates in the centralized “Brain”.

**Algorithm Driven Pattern:** Event-Drive

**Condition:** When any a request comes for resource access lock, this algorithm is activated to make its decision.

**Objective:** Executes the commands which are “Firable” while puts those which are NOT “Firable” into the queue.

**Algorithm Details**

```
Brain ← A₁ //Distributed agent passes the action A₁ to the “Brain”
Qtest ← A₁, Q: // Get the requirement information
```
Figure 3. Working Flowchart of modules in the New Mechanism

Boolean result $\leftarrow$ Check(A$_i$); // check whether this action is "Firable".  
if (result = true){    // if action A$_i$ is firable
    Occupy(Q$_{test}$); // Occupy the resources
    Execute(A$_i$); // execute action A$_i$
    Release(A$_i$); // invoke the "Resource Releaser" to release all occupied resources.
} else {  // not all requirements are meet
    pri $\leftarrow$ A$_i$.pri;  // get its non-negative priority value
    Inqueue(pri, A$_i$); // put the action into the queue A$_{queue}$
}

a) Function Check(A$_i$) Design
/** Check whether action A$_i$ is "Firable". */
P$_{test}$ $\leftarrow$ A$_i$.Q; //Get the list of all required resources
StatusReq $\leftarrow$ A$_i$.precond; // get its required resource working status/modes
TypeReq $\leftarrow$ A$_i$.T; // get its command type to each required resource
Boolean result $\leftarrow$ true;
    for (i $\leftarrow$ 0; i $\lt$ Q$_{test}$.length; i $+$ $+$){
        P$_{n,m}^q$ $\leftarrow$ P$_{test}^q$[i]; // P$_{n,m}^q$ means the No. m resource unit in level n of the DRT of R$_q$
        if ((P$_{1,1}^q$.sta = StatusReq[i])){
            if (P$_{n,m}^q$.occ = Null)
                break; // no occupation
            else if ((P$_{n,m}^q$.occ = Rd U Part_Rd)
                $\land$ (TypeReq[i] = Rd))
                break; // also legal if no write confliction will happen.
    }
}

b) Function Occupy(Q$_{test}$) Design
/** Occupy all required resources based on the list of Q$_{test}$ */
for (i $\leftarrow$ 0; i $\lt$ Q$_{test}$.length; i $+$ $+$){
    R$_{ik}^q$ $\leftarrow$ Q$_{test}$[i];
    Based on the rule described in table 2, update all the occupation information for this resource unit and its direct higher node, such as parent, grandfather, grandfather’s parent, and etc.
    This process can be broke if it encountered some node that does not need any change to its occupation status although one of its children’s is changed.
    R$_{1,1}^q$.sta $\leftarrow$ R$_{ik}^q$.H(A$_i$, R$_{1,1}^q$.sta); // update the working status of the “root” resource unit of the DRT of this resource.
}

c) Function Inqueue(pri, A$_i$) Design
/** Put the action A$_i$ into the queue. */
int i $\leftarrow$ 0;
while (i $\lt$ A$_{queue}$.length){
    pri$_{base}^q$ $\leftarrow$ A$_{queue}$[i].pri;
    if (pri$_{base}^q$ $\lt$ pri)
        i $+$ $+$;
    else {
        A$_{queue}$.insert(i, A$_i$);  // put this action into the queue. Its order is determined based on its privilege value.
    }
}

4.3.2. “Resource Releaser” Design

Place: This module locates in the centralized “Brain”.

Algorithm Driven Pattern: Event-Drive

Condition: When any a reminder (Re$_i$) comes for resource release of certain action, this algorithm is activated to release all occupied resource by this action. Release reminder is actually a kind of action.

Objective: Release the occupied resources of this action; also, maintains the working status of each device.

Algorithm Details

- Brain $\leftarrow$ Re$_i$ (Distributed agent passes the resource release Re$_i$ to the “Brain”)
- Q$_{test}$ $\leftarrow$ Re$_i$.Q
- for (w $\leftarrow$ 0; w $\lt$ Q$_{test}$.getLength; w $+$ $+$){
      Q$_{test}$[w].occ $\leftarrow$ Null; //release this resource unit
      // here we assume that R$_{ij,k}^q$ = Q$_{test}$[w]
Based on the rule described in Table 2, update all the occupation information for this resource unit and its direct higher node, such as parent, grandfather, grandfather’s parent, and etc.

This process can be broke if it encountered some node that does not need any change to its occupation status although one of its children’s is changed.

\[ R^1_{1,1} \text{ sta} \leftarrow R^1_i, H(R^i_{1,1} \text{ sta}); \] // update the working status of the “root” resource unit of the DRT of this resource. It is important to note that \( R^1_i, H(R^i_{1,1} \text{ sta}) \) does not necessarily change the value of \( R^1_{1,1} \text{ sta}. \]

}\} WatchDog() ; // invoke the “watch dog” module.

4.3.3. “Watch Dog” Design

Place: This module locates in the centralized “Brain”.

Algorithm Driven Pattern: Event-Drive and Periodical Invocation

Condition: When a command comes for the invocation of the Watchdog, or the system time passes certain threshold, this algorithm is activated to clear system status and invokes eligible actions.

Objective: It maintains the system inner status consistency and correctness. It also picks out eligible commands from the queue according to certain task scheduling algorithm.

Algorithm Details

Flush Eligible Queue;

for each \( A_i \in \text{Aqueue} \)

if (check(Ai) = true) // action Ai can be executed right now

\( \text{Aqueue}. \text{remove}(A_i); // \text{pop out the eligible action} \)

Eligible Queue. add(Ai); // remove this action and put it into the eligible queue.

Sort(Eligible Queue) // the implementation of this method depends on the specific task scheduling algorithm applied in the system.

else { \n
\( \text{A}_i, \text{pri}++; \)

Sort(Aqueue ); } // increase the priority of stored actions and then based on this priority to resort the members. Thus starvation can be prevented.

5. EXAMPLE

5.1. Definition of resources and actions

In this part, we simply analyze some typical devices in the SimCTCS and constructed corresponding resource maps and access lock mapping relationship. Based on the decomposition method demonstrated in Fig.1, we finish the resource decomposition work in the example as Table 4. Here \( R^1_{1,k} \) means the No. k resource unit in level j of DRT of resource \( R^1_i \).

The interlock (IL) is responsible for the management of railway tracts. Thus each track of a IL forms the resource cell. ATP is responsible for the operation of a train object, thus any train-related actions would read or change the working status of its corresponding ATP. Balis is a kind of fixed device placed in the railway. A train could read or change its working status.

Table 4. Resource Hierarchical Decomposition Scheme

<table>
<thead>
<tr>
<th>Name</th>
<th>Resource Hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interloc 1</td>
<td>R^1_i (Interlock in Station_1)</td>
</tr>
<tr>
<td>Interloc 2</td>
<td>R^2_i (Interlock in Station_2)</td>
</tr>
<tr>
<td>ATP1</td>
<td>R^1_i (Train_A; Train_C)</td>
</tr>
<tr>
<td>ATP2</td>
<td>R^1_i (Train_B; Train_D)</td>
</tr>
<tr>
<td>Balis1</td>
<td>R^1_i (Balis near Station_1)</td>
</tr>
<tr>
<td>Balis2</td>
<td>R^1_i (Balis near Station_2)</td>
</tr>
</tbody>
</table>

The life expectation (LE) of each action in assumed randomly. The mean value (5 Sec.) and deviation (2 Sec.) is determined based on the field test result.

Table 5. Testing Action Configuration Scheme

<table>
<thead>
<tr>
<th>Action</th>
<th>Note</th>
<th>Precond.</th>
<th>Resource Req. &amp; Exp</th>
<th>LE (Sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action_01</td>
<td>Create Train_A at Station B on Track_B_1</td>
<td>Initial</td>
<td>Wr1; Wr01</td>
<td>4.94</td>
</tr>
<tr>
<td>Action_02</td>
<td>Create Train_B at Station A on Track_A_3</td>
<td>Initial</td>
<td>Wr02</td>
<td>5.54</td>
</tr>
<tr>
<td>Action_03</td>
<td>Create Train_C at Station B on Track_B_3</td>
<td>Initial</td>
<td>Wr03</td>
<td>5.01</td>
</tr>
<tr>
<td>Action_04</td>
<td>Create Train_D at Station A on Track_A_5</td>
<td>Initial</td>
<td>Wr04</td>
<td>5.75</td>
</tr>
<tr>
<td>Action_11</td>
<td>Pass Station A through Tracks (A_1 + A_2)</td>
<td>R^1_i,sta</td>
<td>R^1_1,sta</td>
<td>5.81</td>
</tr>
<tr>
<td>Action_12</td>
<td>Pass Station A through Tracks (A_3 + A_4)</td>
<td>R^1_i,sta</td>
<td>R^1_2,sta</td>
<td>6.09</td>
</tr>
<tr>
<td>Action_13</td>
<td>Pass Station A through Tracks (A_5 + A_6)</td>
<td>R^1_i,sta</td>
<td>R^1_3,sta</td>
<td>6.17</td>
</tr>
<tr>
<td>Action_21</td>
<td>Pass Station B through Tracks (B_1 + B_2)</td>
<td>R^1_i,sta</td>
<td>R^1_4,sta</td>
<td>6.65</td>
</tr>
<tr>
<td>Action_22</td>
<td>Pass Station B through Tracks (B_3 + B_4)</td>
<td>R^1_i,sta</td>
<td>R^1_5,sta</td>
<td>6.20</td>
</tr>
</tbody>
</table>
5.2. Computing Complexity

In the Function Check(), the process has to check the occupation status for all the necessary resources required by the action; meanwhile, if its required resources are not “root” in the individual DRT, corresponding “higher level” resource status have to be checked. In addition, the working status of the target resource has to be checked, too.

In the Function Release(), the system has to reset the occupation status of current resource, meanwhile, the working status and occupation information of all its fellow resource unit, which share the same “Parent” node, are gathered to generate the new “Parent” node status. If this new value is different from the old value, the release process has to repeat once at the parent level. This cycle process will not stop until there is no need to continue this process or it reaches the “level 1”.

In the hierarchical structure by which the working status and resource occupation information is maintained and managed, we assume there is one “Root”, and on average n “Level-2” nodes, and n² “Level-3” nodes, and etc. There are N levels in all. The action A_i requires m resources, on average, they are all “level-k” (1 ≤ k ≤ N) resources. Thus the computing complexity for the function Check() and Release() for this action can be calculated.

5.3. Simulation Experiment

We make four simple scripts (see Table 9 for details), which includes multiple actions, to test the system working effectiveness with common concurrent method in HIL system, and the proposed concurrent method. The start time of scripts is randomly chosen.

Table 7. Computing Complex of the New Mechanism

<table>
<thead>
<tr>
<th>Function</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check()</td>
<td>O(m * k)</td>
</tr>
<tr>
<td>Release()</td>
<td>O(m * k * (n - 1))</td>
</tr>
</tbody>
</table>

Table 8. Computing Complex of Comparison Mechanism

<table>
<thead>
<tr>
<th>Function</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check()</td>
<td>O(m * n^k)</td>
</tr>
<tr>
<td>Release()</td>
<td>O(m * n^k)</td>
</tr>
</tbody>
</table>

It is obvious that when N > k and n is not a small number, the new method is much more effective.

5.3. Simulation Experiment

We make four simple scripts (see Table 9 for details), which includes multiple actions, to test the system working effectiveness with common concurrent method in HIL system, and the proposed concurrent method. The start time of scripts is randomly chosen.

Table 9. Script Content and Life Expectation

<table>
<thead>
<tr>
<th>Script</th>
<th>Action_01</th>
<th>Action_21</th>
<th>Action_32</th>
<th>Action_11</th>
<th>Action_41</th>
<th>t_start</th>
<th>LE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE</td>
<td>4.94</td>
<td>6.65</td>
<td>7.22</td>
<td>5.81</td>
<td>3.19</td>
<td>3.5</td>
<td>27</td>
</tr>
</tbody>
</table>


In this paper, a new kind of concurrent mechanism applies to improve the general HIL system concurrency while maintain logic correctness.

From the experiment result, it is clear that the effectiveness of this new concurrent mechanism lies in the capability of exploiting the inner concurrent operation ability of complex devices. Study on other kinds of railway complex devices and finish modeling them into the simulation system is one of our future directions.

5.4. Experiment Result Analysis

From the experiment result demonstrated in Table 10 and Fig. 4, we can see the hierarchical sharing lock mechanism (“New Method”) performs better (about 20%) than pure scheduling-based method under same task scheduling algorithm, system settings and test scripts. Especially in complex environment, sharing lock mechanism makes the system more flexible.

6. CONCLUSION AND FUTURE WORK

In this paper, a new kind of concurrent mechanism was proposed and experimentally verified. From the experiment result, it is clear that the effectiveness of this new concurrent mechanism lies in the capability of exploiting the inner concurrent operation ability of complex devices.

Acknowledgment

This work was supported in part by the National Key Technology R&D Program under Grant 2009BAG12A08, the Research Foundation of the Ministry of Railways and Tsinghua University (RFMOR&THU) under Grant J2009Z28 and the Research Foundation of Beijing National Railway Research and Design Institute of Signal and Communication.

Table 10. Comparison Experiment Result

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
<th>GTTL (Sec.)</th>
<th>Improve (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>Four scripts are executed simultaneously. However, any (Level-1) resource could operate only exclusively.</td>
<td>61.3 (bench mark)</td>
<td></td>
</tr>
</tbody>
</table>
| New Mechanism | System would automatically change the granularity of resources; hierarchically maintain the working status of better resources and occupation information. | 49.3 | 19.65%

The General Test Time Length (GTTL) in Table 10 is applied to quantify the efficiency of each test scheme. The “SIMPLE” method means the common method, i.e., any resource could support the execution of at least one action at any time. The “NEW” method is the new designed concurrent method demonstrated in this paper.

5.4. Experiment Result Analysis

From the experiment result demonstrated in Table 10 and Fig. 4, we can see the hierarchical sharing lock mechanism (“New Method”) performs better (about 20%) than pure scheduling-based method under same task scheduling algorithm, system settings and test scripts. Especially in complex environment, sharing lock mechanism makes the system more flexible.

6. CONCLUSION AND FUTURE WORK

In this paper, a new kind of concurrent mechanism applies to improve the general HIL system concurrency while maintain logic correctness.

From the experiment result, it is clear that the effectiveness of this new concurrent mechanism lies in the capability of exploiting the inner concurrent operation ability of complex devices. Study on other kinds of railway complex devices and finish modeling them into the simulation system is one of our future directions.

Acknowledgment

This work was supported in part by the National Key Technology R&D Program under Grant 2009BAG12A08, the Research Foundation of the Ministry of Railways and Tsinghua University (RFMOR&THU) under Grant J2009Z28 and the Research Foundation of Beijing National Railway Research and Design Institute of Signal and Communication.

References