Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 163845, 12 pages
http://dx.doi.org/10.1155/2015/163845

Research Article

A Practical Infrastructure for Real-Time Simulation across Timing Domains

Yao-fei Ma,[1] Xiao Song,[1] Jiang-yun Wang,[1] and Zhen Xiao[2]

[1]*School of Automation Science and Electrical Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China*
[2]*Beijing Institute of Nearspace Vehicle's Systems Engineering, China Aerospace Science and Technology Corporation, Beijing 100076, China*

Correspondence should be addressed to Xiao Song; songxiao@buaa.edu.cn

Received 6 June 2014; Revised 16 September 2014; Accepted 20 September 2014

Academic Editor: Minrui Fei

A real-time infrastructure, called MLRTI, is proposed in this paper to fulfill the requirement of real-time simulation in distributed environment. There are two novel contributions in this work. Firstly, a flexible timing mechanism is proposed to integrate external time source and local timer utility, enabling the distributed nodes to advance their timeline simultaneously at different speeds with high precision. A data transmission solution is also presented in which the reflective memory card (VMIC) is employed to provide fast data transmission with minimum delay. Secondly, a system partition schema is proposed in MLRTI to reduce the solution errors introduced by transforming a continuous system into distribution system, which is common in a class of control applications where the system is designed in centralized model but simulated in distributed environment for constrains on system structure or the need to balance computation load. Experiments are conducted and the results show this schema effectively reduces the possible errors by properly partitioning the system into parts that are suitable to be deployed in distributed environment.

1. Introduction

Real-time simulation has been applied in many domains like defense [1, 2], aero, and air space systems [3, 4], embedded automotive electronics [5], and so forth. These domains are often involved with hardware-in-the-loop or human-in-the-loop applications, where the response speed to control signals/commands is critical and the model equations must be solved within limited time intervals. For example, in a robotic control system, the vision system needs to respond with low latency because the produced image is required with the feedback loops of robotic application [6].

Sometimes a real-time system has to be deployed in distributed environment. One case is the system's structure which is scattered. For example, a training system is where the human operated device is located at one place while the controller is at another place. Another case is the scale of the system which is large and has to be distributed to average the computation load on each node [7]. As a result, a real-time infrastructure for distribution simulation must handle the following issues.

(i) It is to synchronize time advancements on distributed nodes. There are often multiple subsystems coexisting and each has different timing requirements. For example, in combat simulation, virtual entities created by computer may need not to interact with human operator directly. Thus their time advancement step can be longer (e.g., 50 milliseconds); occasionally delay on time advancement would not lead to fatal failure. On the other hand, nodes operated by human being (e.g., a manned flight simulator) need strict real-time performance in which the simulation step of 1~5 milliseconds or shorter is needed.

(ii) It is to ensure the data between subsystems can be exchanged timely. The performance on data transmission will be affected by multiple factors like bandwidth, data load, topology, and so forth. The unpredictable delays among simulation nodes are unacceptable in critical real-time applications.

(iii) A common but little concerned one is the transforming problem when constructing the distributed system. In the application of continuous system, the original system is normally designed and tested in centralized manner; that is, the system is constructed as a whole and tested in a standalone computer; the data transmission does not cross network. When such system is deployed in distributed environment, the solution errors, that is, the difference on states trajectories in two systems, can be introduced.

There does exist formal approach like QSS [8–10] to transform a continuous model/system into one that is suitable for distributed environment. However, the quantization operation, named as "hysteretic quantization," is necessary for QSS to hold the outputs of submodels until some predefined thresholds are crossed. This operation needs to modify system models, which will lead to problem when too many models exist or modifications are not allowed at all due to classified reason. Additionally, QSS approach did not discuss the situation where the model resolver step h is different from the distributed simulation step T, which is the common cases in real-world applications.

In this paper, a multiple layer real-time infrastructure (MLRTI) is proposed to address these problems. MLRTI highlight the following three characteristics: (i) high precision global timing capability achieved by integrating external timing source and local timer; (ii) low latency on data transmission and the publish/subscribe mechanism borrowed from High Level Architecture (HLA) (IEEE standard 1516) to specify data exchanging map between components; (iii) a novel partition schema to reduce the solution errors (for control system) incurred by system transformation from the centralized one to the distributed one.

The content of this paper is organized as follows. In Section 2, the structure of MLRTI is introduced. Two critical characteristics, the high precision timing and low latency data communication, are discussed in detail. And a publishing and subscribing mechanism is also presented to form the data exchanging map in distributed environment. In Section 3, the system partition schema is given to minimize the possible solution errors caused by transforming a centralized system into distributed system. Comparative experiments are conducted to verify its effectiveness.

2. System Structure

MLRTI is designed to allow the distributed parts of a system to work together. Each part can have different timing speeds. Normally, there are different requirements on timing performance within a system:

(a) the nonreal-time (NRT) tasks like resource management, deployment configuration, and so forth; these tasks are normally performed at presimulation period and need not to advance time with real-time manner;

(b) the soft real-time (SRT) parts, which advance their times at real-time but failures (e.g., the inconsistency caused by inaccurate timing or delays on data transmission) are allowed; examples include the displaying part or virtual entities that do not interact with human operators directly; the time interval here can be 1 ms–100 ms;

(c) the hard real-time (HRT) parts, including the human operated equipment or the models whose inputs must be sampled from external environment; the computation of these parts must be completed within specified interval (e.g., 1 μs–1 ms); their times need to be advanced with high precision.

To satisfy all these requirements, a layered infrastructure is proposed, as Figure 1 shows.

The NRT layer, SRT layer, and HRT layer are presented from the top to the bottom, respectively. The differences among layers are distinguished by two factors: the timing precision and the data transmission speed. They are often connected with each other: high data transmission speed helps to improve the timing precision, and high timing precision helps to align data updating interval, thus eliminating the data jitters.

In NRT layer, the network media can be common Ethernet. Currently, the bandwidth of Ethernet can reach 1 Gb/s or more. However, the underlying mechanism of detecting data collision on Ethernet (CSMA/CD according to IEEE802.3) could cause inevitable delays on data transmission. According to our test, the average transmission delay in a commonly configured Ethernet LAN is about 10~15 milliseconds [11].

To reduce this delay, one solution is to measure and compensate time-delay by special schedule mechanism [12]. However, it is not a complete approach. A better solution is to improve the collision detecting mechanism of Ethernet protocol. An excellent example is the PowerLink protocol [13] whose PowerLink stack can replace the TCP/IP and UDP/IP layer seamlessly, called PowerLink Layer. This layer realizes fast, real-time data transmission in which a collision prevention mechanism called "Slot Communication Network Management (SCNM)" is employed to synchronize each node in the polling way. In the best case, the minimum interval between data package sending is about 100 μs, which can meet the requirement of SRT.

As for HRT layer, the requirements on timing and data transmission are higher. Specialized hardware needs to be employed to meet such requirement. High precision time sources, like GPS and BDS (BeiDou Navigation Satellite System), can be introduced into the simulation for high precision synchronizing signal. Additionally, reflective memory card (VMIC) connected with optical fiber provides a mechanism for high speed, low latency data transmission capability (e.g., GE PCI-5565piorc's data rate $\geq$ 170 Mb/s; time delay can reach nanosecond level). The specialized hardware and protocol enable CPUs not to be involved in data sending/receiving process; the big bandwidth of optical fiber also contributes to the high performance on data communication.

2.1. Flexible Timing Solution. As mentioned previously, precise timing is one of the key factors of real-time system. Time synchronization is important to maintain correct timeline and causal relationship between nodes. Two issues need to be addressed [14]:

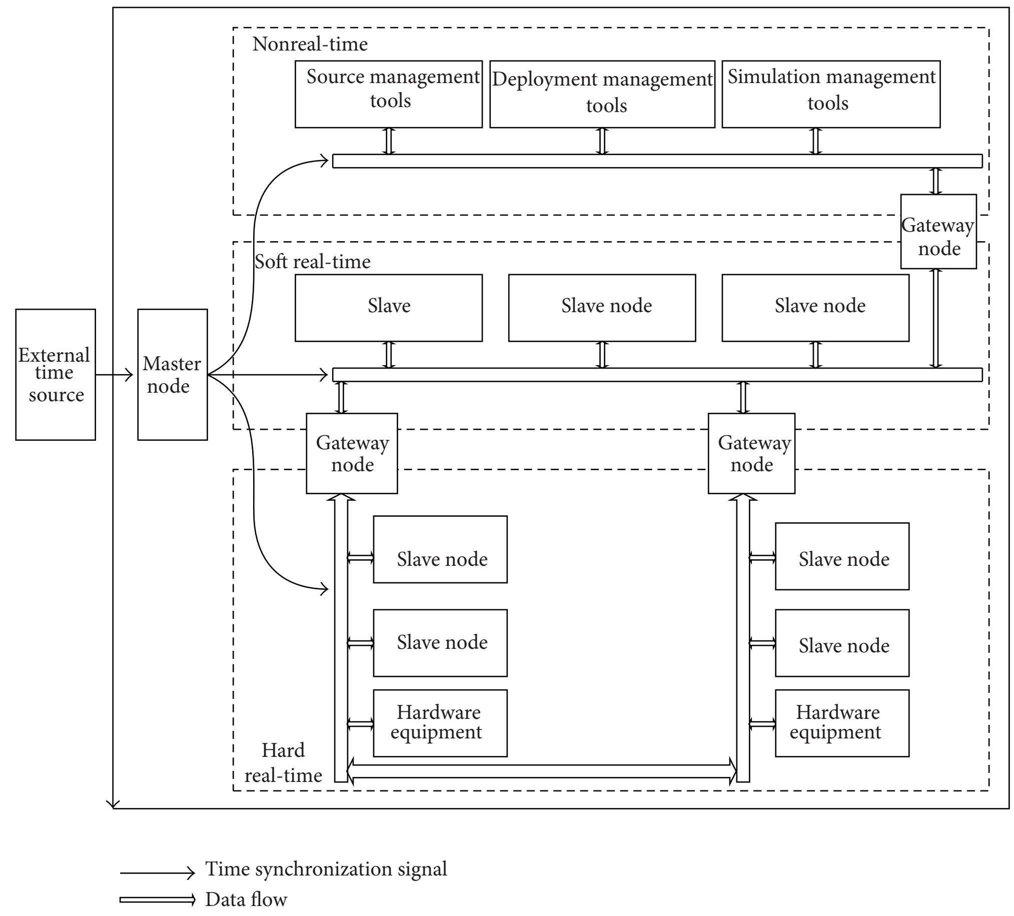


Figure 1: The structure of MLRTI.

(a) absolute synchronization: simulation time needs to align precisely with external time in real world, that is, the physical time; it is important in hybrid simulation like military drill STOW 99 held in US, where real military force (human and equipment) and virtual force interact with each other;

(b) relative synchronization: the distributed nodes also need to align their own times during simulation.

Relative synchronization is easier to realize. Timer utilities provided by operation system (OS) or other commercial software can be used to fulfill it. Four utilities are introduced here.

The first one is the timer utility provided by OS. For example, the multimedia time provided by Windows OS can provide periodic timing with accuracy of 1 ms; and its built-in error compensation mechanism can check and remove accumulated timing error.

The second one is the crystal oscillator clock embedded on computer's motherboard, which could provide more accurate time signal with nanoseconds interval. However, the precision of this signal will vary with motherboard type, manufacturing technology, or working temperature.

Commercial software RTX (real-time extension) is designed to overcome the lack of real-time capability of Windows operating system. RTX can provide time interval at 1 ms, 1 μs, or one-tick interval produced by computer hardware by giving its own clock signals.

The fourth one is the specialized hardware. VMIC is often used in HRT domain to get high precision timing capability. It can broadcast the interrupt signals to network and the perdefined interrupt function on each node will be invoked to process them.

The former two utilities cost low since they are easy to access. However, their performances are limited. The latter two provide improved performance, at the cost of expensive investment.

External time source needs to be imported into system when absolute synchronization is required. The most convenient way is to use timing signals from global navigation satellite system (GNSS). However, it is difficult to get high resolution time signals from such system due to cost or authorization reason. The common resolution is 1 s [15].

As a result, a "Master-Slave" structure is proposed here to manage the timing in distributed manner, as Figure 2 shows.

The master node is responsible for timing control throughout the system. Its time is pinned to external time

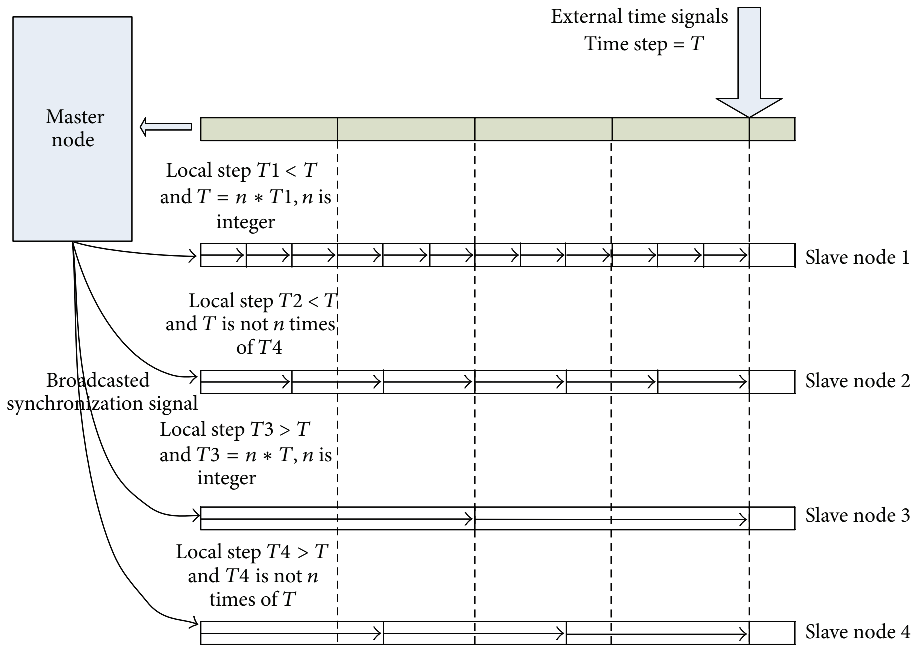

External time signals
Time step = T
Master node
Local step $T1 < T$ and $T = n * T1$, n is integer
Slave node 1
Local step $T2 < T$ and T is not n times of $T4$
Slave node 2
Broadcasted synchronization signal
Local step $T3 > T$ and $T3 = n * T$, n is integer
Slave node 3
Local step $T4 > T$ and $T4$ is not n times of T
Slave node 4

Figure 2: The Master-Slave structure for global timing in MLRTI.

source and produces synchronization signals. The signals are broadcasted to the whole network, as Figure 2 shows. The slave nodes keep listening to the signals. Each slave node will firstly check the received signal. If this signal is intended for it, the slave node will proceed the following process: (i) update its local time; (ii) read in the latest data from its source node (if any exists); and (iii) do computation and update its outputs. When using VMIC device, this process is done via interrupt functions. The timing precision is ensured by both external time source and local timer.

To maintain different time advancement speeds on simulation nodes, there are two information arrays being maintained in the master node: (i) the time advancement request array; request from each slave node specified the next time point it needs to go to; (ii) the priority array that records a predefined priority sequence, by which the distributed nodes are allowed to advance their time one by one. A complete procedure is described as follows.

(1) Master node polls the request array and pick up the nearest next time point T_n. The slave nodes (there could be multiple nodes asking for advancing to the same time point) who send this request are recorded in set variable N_q.

(2) Master node resorts N_q with descending priority order and then sends the interrupt signals when each item's T_n is reached.

(3) Slave nodes that are allowed to advance their time do model computation in specified time interval and then send a new time advancement request to the master node. The next time point is told by this way.

(4) Master node keeps waiting until all time advancing requests (from N_q) arrived; then the request array is updated. If some node does not respond in time, the master node would have three options: (a) ignore this delay and keep going; (b) warn this delay and ask user if to continue; (c) warn this delay and stop simulation. It is decided by user.

(5) Go back to step (1).

2.2. Data Exchanging Map between Nodes. The data exchanging between simulation nodes is defined before simulation start. The data is classified into two classes according to their importance: "property" and "message." "Property" refers to data periodically produced by models and exchanged during simulation, which are not critical and a small amount of loss cannot lead to fatal consequence to the stability and correctness of the whole system. On the other hand, "message" is important to notify the critical events and keeping causality correct. In Figure 1, the "messages" are kept and the "properties" are discarded when data exchanging happened between real-time domains with different advancing speed.

To describe the data exchanging between nodes, the concept of "publishing and subscribing" is borrowed from HLA in which "publishing" node means it can produce data to simulation space and "subscribing" node consumes data produced by others.

The publishing and subscribing map is defined in a tree, in which the parent node represents the parent publishing or subscribing over its son nodes. For example, a "vehicle" node may has son nodes of "fighter" and "tank"; subscribing "vehicle" data means subscribing both "fighter" and "tank" data. On the other hand, publishing or subscribing of a leaf node only triggers the corresponding nodes when that leaf node (data or event) updates itself. This mechanism provides extra flexible to the description of data exchanging.

2.3. Performance Experiments. A distributed HRT environment was built up to test the delays that may exist in timing and data exchange. In our experiment scenario, three computers equipped with VMICs (GE PCI-5565piorc) are connected with optical fiber. Time advancing mechanism takes "Master-Slave" structure, and the local timer of master node is enhanced by RTX middleware working with Windows XP.

The performance of relative synchronization is tested in this scenario. On each timing step, delay comes from three aspects: (i) timing aberration of RTX timer, (ii) data transmission delay caused by VMIC's ring network protocol, and (iii) responding latency of interrupt function of VMIC since receiving interrupt signal.

According to the product specification, data transmission latency between adjacent RMICs is about 0.4 μs. In Figure 3, 1000 measurements of RTX timer callback with interval of 1 ms are recorded. As we can see, the maximum deviation is less than 1.8 μs and the average deviation is 0.3758 μs.

In Figure 4, 1000 measurements of the responding latency of interrupt function of VMIC since receiving interrupt signal are recorded. The maximum latency is less than 23 μs; the average latency is about 13.228 μs.

The average time advancement error between simulation nodes is 0.3758 + 13.228 + 0.4 = 14.0038 μs. For HRT domain advancing at millisecond level, this error is less than 5% of time step and can be omitted most of time.

The experiment on timing performance with external time source is not conducted here, considering the fact that external time sources like GPS and BDS have been quite mature in their technological evolution and own stable timing precision, which can be added with time delay inside the simulation (Figures 3 and 4) to get the final performance.

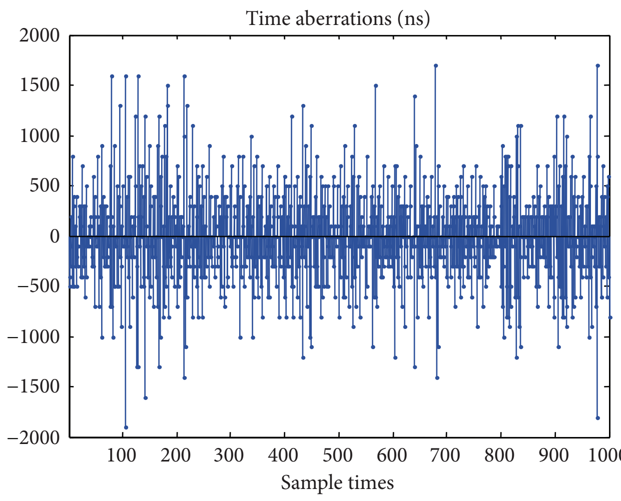


FIGURE 3: Variation of the length of time intervals triggered by timer of RTX.

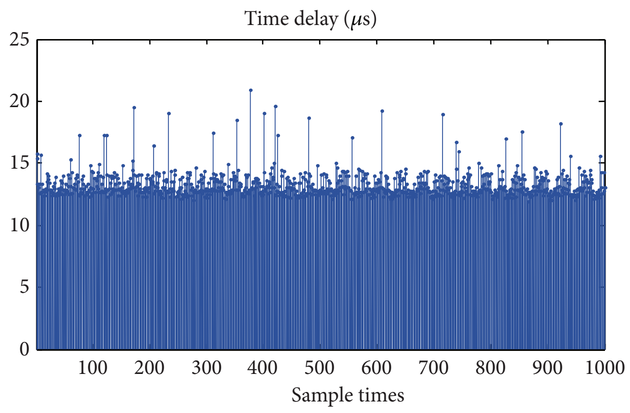


FIGURE 4: Time latency before interrupt service function starting since VMIC received the interrupt signal.

3. System Partition Schema

MLRTI guarantees that the distributed system can advance its global time synchronously with high precision and exchange data swiftly with low latency. However, for a class of continuous system application, the possible errors introduced by system transforming are not considered yet. It is a common case that a control system is designed in "centralized" manner; that is, the system is constructed as a whole and is tested in a standalone computer. When such a system is deployed in distributed environment, the new system has been different from the original one. In brief, the input/output sequence between each distributed part could be disordered. This can be explained with a simple example shown in Figure 5, an inverted pendulum control system.

3.1. Problem Description. The inverted pendulum system contains 9 submodels in it, as Figure 5 shows. In nondistributed simulation, all submodels need to be computed and updated on each step, with certain computing order. The principle to determine this order is not to violate data dependence among submodels. To achieve this, the models can be classified into two categories according to their input/output characteristics.

(a) Direct-Feed-Through (DFT) Model. DFT port is defined as a pair of <input, output> where the output is determined by current input value. A DFT model owns one or more DFT ports. Assuming a model can be described with three sets of variables: input set X, state set S, and output set Y; then the DFT model can be expressed as

$$\begin{aligned} \dot{S}_{n+1} &= I\left(S_n, X_{n+1}\right), \\ S_{n+1} &= T\left(\dot{S}_{n+1}, S_n, h\right), \\ Y_{n+1} &= O\left(S_{n+1}, X_{n+1}\right), \end{aligned} \qquad (1)$$

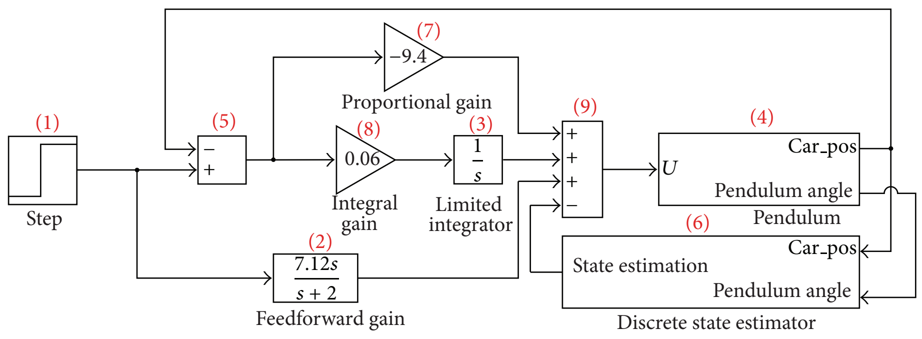


Figure 5: Inverted pendulum control system: a sorted order (represented with red, bracketed figures) needs to be determined to compute and update each model correctly in nondistributed environment.

where $I(*)$ is input function, $T(*)$ is state transition function, and $O(*)$ is output function. The latest output Y_{n+1} is determined by current input X_{n+1} and states S_{n+1}, which implies a sequent computing order existing between this DFT model and its preceding models (that produce X_{n+1}). $S = \Phi$ is a special case of DFT model, where Φ is the empty set. Common DFT models include gain, product, sum, derivative, and so forth.

(b) Non-Direct-Feed-Through (NDFT) Model. NDFT model has no DFT ports. A NDFT model can be expressed as

$$\begin{aligned}\dot{S}_{n+1} &= I\left(S_n, X_{n+1}\right),\\ S_{n+1} &= T\left(\dot{S}_n, S_n, h\right),\\ Y_{n+1} &= O\left(S_{n+1}\right).\end{aligned} \tag{2}$$

The output is associated with current state S_{n+1} rather than the current input X_{n+1}, which means this NDFT model can produce output without waiting for the latest input; thus the constrain on the computing order between it and its preceding models is relaxed. Common NDFT models include integrator, input signal, memory, and so forth.

Upon this classification, the rules to determine the computing order can be stated as follows.

(1) For a DFT model, the models which drive its DFT ports should be computed before it.

(2) For a NDFT model, it can be computed with any order as long as before the DFT models it drives.

There could be multiple feasible computing orders for a specific system according to the above rules; the red bracketed figures in Figure 5 indicate one of them. In centralized simulation, the system works well by this order; however, the case becomes complex when it is deployed in distributed environment.

An extreme scenario is to deploy each submodel to a separate node. Obviously, the system would still work well as long as the computation order is maintained. However, it is meaningless to maintain a "sequent" computation in a distributed environment. If we want to make full use of the advantage of distributed environment, that is, to compute in parallel, the input/output between distributed models may become disordered. To describe it, the output sequence of each submodel will be analyzed.

Table 1: Model classification.

NDFT model	DFT model
(1) Signal source (step) ①	(1) Sum operator ⑤
(2) Feedforward gain ②	(2) Discrete state estimator ⑥
(3) Integrator ③	(3) Proportional gain ⑦
(4) Pendulum ④	(4) Integral gain ⑧
	(5) Multiple-input sum ⑨

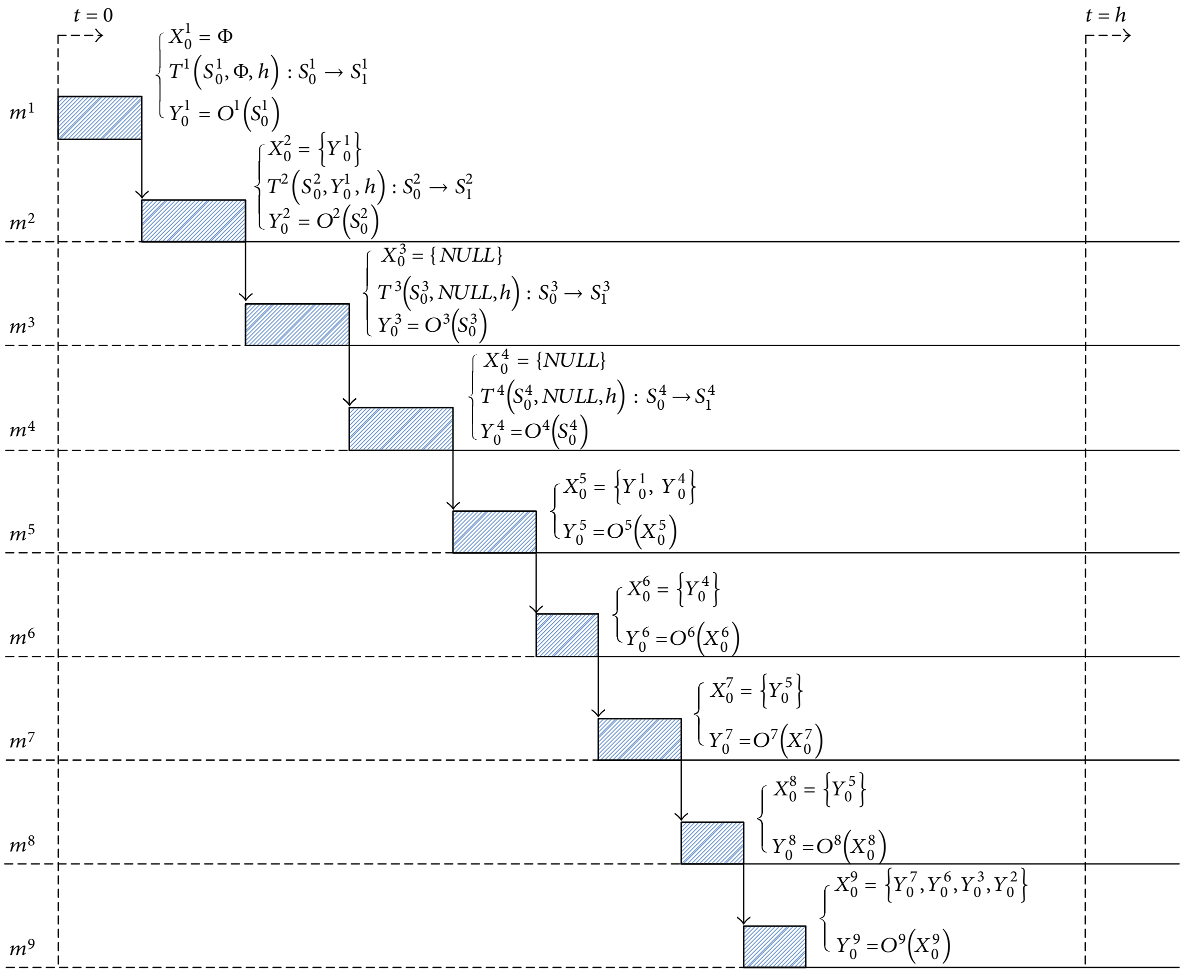
In the following analysis, two time symbols would be referred to: the time step h of model resolver and the simulation step T of distributed system. Actually, the continuous model is normally implemented as "discrete time model" with specific numerical simulation schema, and the respond numerical resolver (e.g., the *Euler* or *Runge-Kutta* resolver) is employed to compute it. The resolver can be fixed or variable step size. In this example, a fixed step of $h = 0.001$ s is used. Simulation step is the globally allocated time interval for each simulation node to compute the models deployed on it. Data exchanges are performed at the end of each simulation step. Without losing generality, we specify that $T = n \cdot h$, $n = 1, 2, \ldots$.

According to the types of model port, there are 4 NDFT and 5 DFT models in the inverted pendulum system as shown in Table 1.

Each model is described by three variable sets $\{X, S, Y\}$. The subscript of them refers to the index of time step, and the superscript refers to the index of submodel (denoted as m^n). The computation of each model is displayed in Figure 6. The expressions are briefly explained as follows:

(i) $X = \{*\}$: receiving current inputs "$*$"; it should be noted that there are two special representations on inputs: (a) $X = \{\Phi\}$ means this model has no inputs, and (b) $X = \{NULL\}$ means the input is not available at computing time;

(ii) $T(S, X, H) : S \rightarrow S'$: transiting states from old state S to new state S', with current input X and time interval



$t = 0$
$t = h$
m^1
m^2
m^3
m^4
m^5
m^6
m^7
m^8
m^9
$X_0^1 = \Phi$, $T^1(S_0^1, \Phi, h) : S_0^1 \to S_1^1$, $Y_0^1 = O^1(S_0^1)$
$X_0^2 = \{Y_0^1\}$, $T^2(S_0^2, Y_0^1, h) : S_0^2 \to S_1^2$, $Y_0^2 = O^2(S_0^2)$
$X_0^3 = \{NULL\}$, $T^3(S_0^3, NULL, h) : S_0^3 \to S_1^3$, $Y_0^3 = O^3(S_0^3)$
$X_0^4 = \{NULL\}$, $T^4(S_0^4, NULL, h) : S_0^4 \to S_1^4$, $Y_0^4 = O^4(S_0^4)$
$X_0^5 = \{Y_0^1, Y_0^4\}$, $Y_0^5 = O^5(X_0^5)$
$X_0^6 = \{Y_0^4\}$, $Y_0^6 = O^6(X_0^6)$
$X_0^7 = \{Y_0^5\}$, $Y_0^7 = O^7(X_0^7)$
$X_0^8 = \{Y_0^5\}$, $Y_0^8 = O^8(X_0^8)$
$X_0^9 = \{Y_0^7, Y_0^6, Y_0^3, Y_0^2\}$, $Y_0^9 = O^9(X_0^9)$

FIGURE 6: Model computation and data exchanging within one step in centralized simulation.

H. A brief convention is employed here; for example, $T(S = S_0, X = \{\Phi\}, H = h) : S_0 \to S'$ is represented as $T(S_0, \Phi, h) : S_0 \to S'$;

$T(S, NULL \mid \Phi, h)$ which represents that the state transition is triggered by time interval h, not by input; this case only appears when $X = \{\Phi\}$ or $X = \{NULL\}$; the model can be considered being out of control when $X = \{NULL\}$;

DFT models which have no internal states; thus their state transitions are omitted in the following analysis;

(iii) $Y = O(*)$: computing the outputs Y; symbol "$*$" represents (a) the internal states if this is a NDFT model or (b) the current inputs if this is a DFT model.

The outputs and state transitions at the first 3 steps are listed in Table 2.

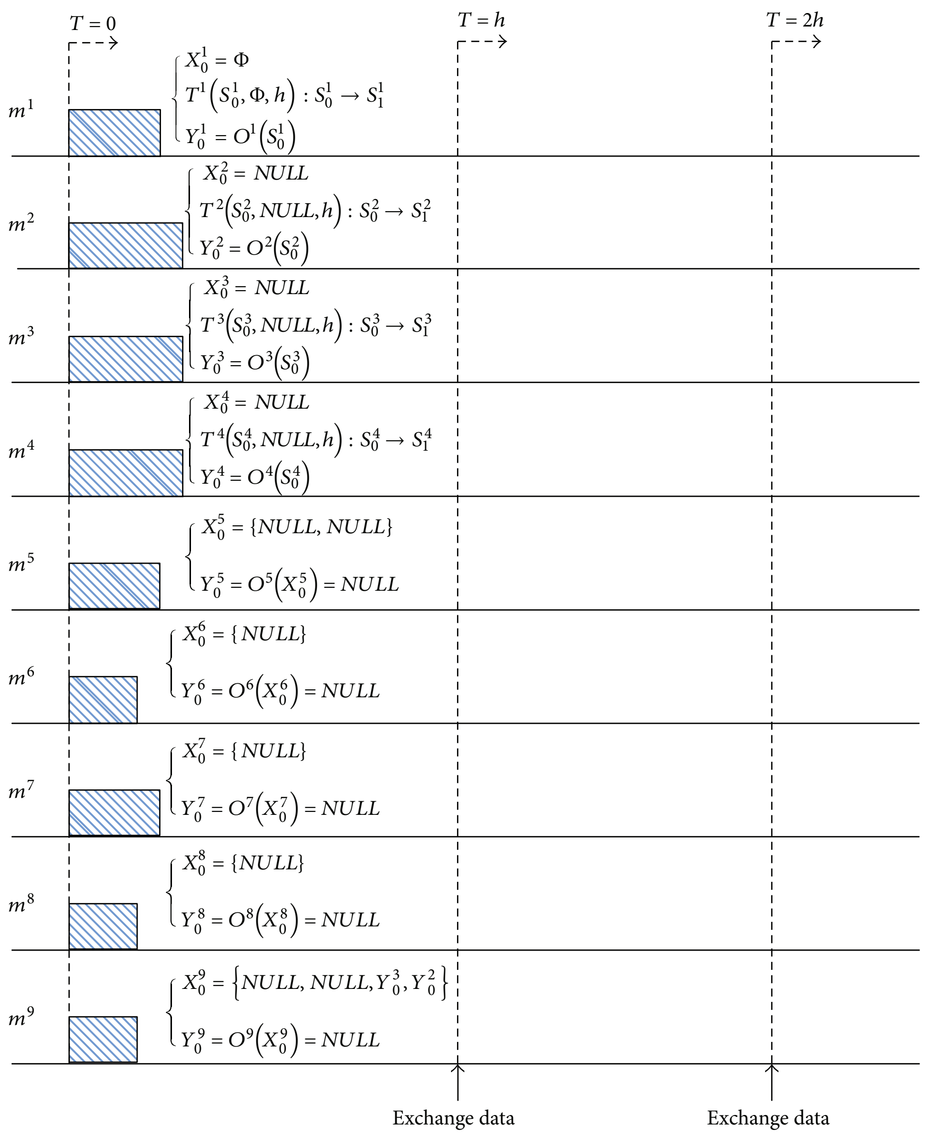
Figure 7 shows the simulation advancing in distributed environment. Each submodel is computed in parallel manner and data exchange happens at the end of each simulation step.

The outputs and states transition of the first 5 (or 6) steps are listed in Table 3.

The outputs of m^4 (NDFT model) and m^5 (DFT model) are compared here between centralized and distributed scenario, as Figure 8 shows. Firstly, m^4 and m^5's outputs are all delayed in distributed environment. The 3rd output of m^5 in centralized simulation, y_2^5, appears at the 7th step in distributed simulation. Similarly, the 3rd output of m^4 in centralized simulation, y_2^4, appears at the 6th step in distributed simulation. Secondly, the values of the corresponding outputs in centralized and distributed scenarios are also changed except the delayed time. For example, it is found that $y_2^5 \neq y_6^5$ by backtracking their data dependence.

The following facts on output delay can be concluded from the above comparison. All models' outputs would be delayed except the source model (m^1). Delays start from the first step since all nodes need to advance their computation simultaneously. The first input of each node is missing. For DFT model, the missing input would produce invalid output (denoted as "$NULL$"); for NDFT model, it is equivalent that the model's dynamics is changing with inertia rather than external stimulation.

When two or more DFT models are cascaded, the delay will accumulate along the cascading path, but NDFT do not.



$T = 0$
$T = h$
$T = 2h$
m^1
$X_0^1 = \Phi$
$T^1(S_0^1, \Phi, h) : S_0^1 \to S_1^1$
$Y_0^1 = O^1(S_0^1)$
m^2
$X_0^2 = NULL$
$T^2(S_0^2, NULL, h) : S_0^2 \to S_1^2$
$Y_0^2 = O^2(S_0^2)$
m^3
$X_0^3 = NULL$
$T^3(S_0^3, NULL, h) : S_0^3 \to S_1^3$
$Y_0^3 = O^3(S_0^3)$
m^4
$X_0^4 = NULL$
$T^4(S_0^4, NULL, h) : S_0^4 \to S_1^4$
$Y_0^4 = O^4(S_0^4)$
m^5
$X_0^5 = \{NULL, NULL\}$
$Y_0^5 = O^5(X_0^5) = NULL$
m^6
$X_0^6 = \{NULL\}$
$Y_0^6 = O^6(X_0^6) = NULL$
m^7
$X_0^7 = \{NULL\}$
$Y_0^7 = O^7(X_0^7) = NULL$
m^8
$X_0^8 = \{NULL\}$
$Y_0^8 = O^8(X_0^8) = NULL$
m^9
$X_0^9 = \{NULL, NULL, Y_0^3, Y_0^2\}$
$Y_0^9 = O^9(X_0^9) = NULL$
Exchange data
Exchange data

FIGURE 7: Model computation and data exchange in distributed simulation.

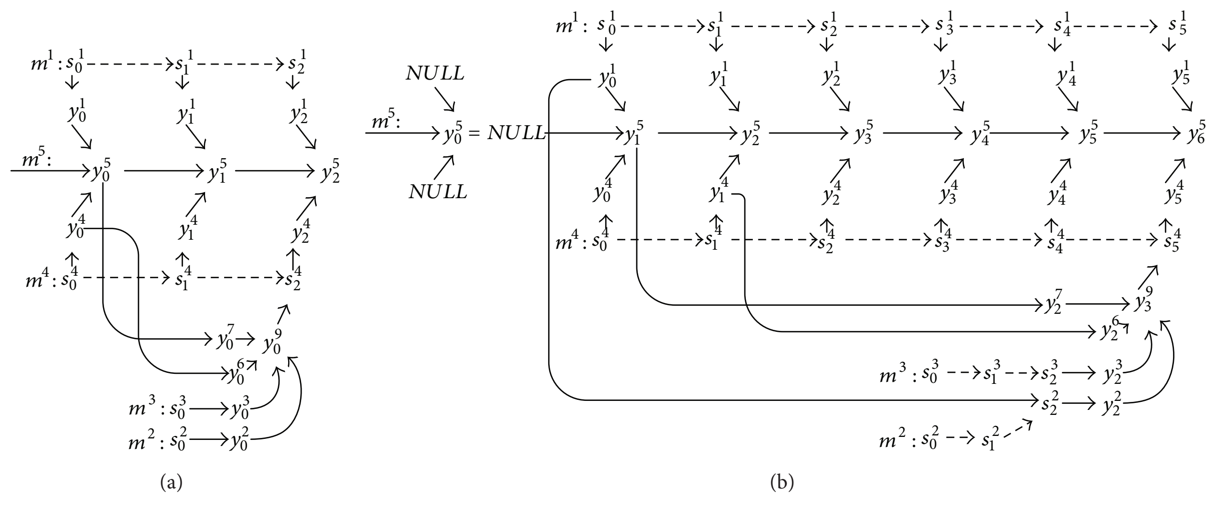


FIGURE 8: The output sequences of m^4 and m^5 in centralized (a) and distributed (b) scenario. Solid arrows indicate the data dependence for output generation; dotted arrows indicate the data dependence for state transition.

Table 2: The output sequence of submodels in centralized simulation.

Models	Outputs	States transition
m^1	$Y_0^1 = O^1(S_0^1)$ $Y_1^1 = O^1(S_1^1)$ $Y_2^1 = O^1(S_2^1)$	$S_1^1 = T^1(S_0^1, \Phi, h)$, $S_2^1 = T^1(S_1^1, \Phi, h)$
m^2	$Y_0^2 = O^2(S_0^2)$, $Y_1^2 = O^2(S_1^2)$, $Y_2^2 = O^2(S_2^2)$	$S_1^2 = T^2(S_0^2, Y_0^1, h)$, $S_2^2 = T^2(S_1^2, Y_1^1, h)$
m^3	$Y_0^3 = O^3(S_0^3)$, $Y_1^3 = O^3(S_1^3)$, $Y_2^3 = O^3(S_2^3)$	$S_1^3 = T^3(S_0^3, NULL, h)$, $S_2^3 = T^3(S_1^3, Y_0^8, h)$
m^4	$Y_0^4 = O^4(S_0^4)$, $Y_1^4 = O^4(S_1^4)$, $Y_2^4 = O^4(S_2^4)$	$S_1^4 = T^4(S_0^4, NULL, h)$, $S_2^4 = T^4(S_1^4, Y_0^9, h)$
m^5	$Y_0^5 = O^5(Y_0^1, Y_0^4)$, $Y_1^5 = O^5(Y_1^1, Y_1^4)$, $Y_2^5 = O^5(Y_2^1, Y_2^4)$	—
m^6	$Y_0^6 = O^6(Y_0^4)$, $Y_1^6 = O^6(Y_1^4)$, $Y_2^6 = O^6(Y_2^4)$	—
m^7	$Y_0^7 = O^7(Y_0^5)$, $Y_1^7 = O^7(Y_1^5)$, $Y_2^7 = O^7(Y_2^5)$	—
m^8	$Y_0^8 = O^8(Y_0^5)$, $Y_1^8 = O^8(Y_1^5)$, $Y_2^8 = O^8(Y_2^5)$	—
m^9	$Y_0^9 = O^9(Y_0^7, Y_0^6, Y_0^3, Y_0^2)$, $Y_1^9 = O^9(Y_1^7, Y_1^6, Y_1^3, Y_1^2)$, $Y_2^9 = O^9(Y_2^7, Y_2^6, Y_2^3, Y_2^2)$	—

In the situation where NDFT and DFT models are mixed cascaded, the delays can be counted as follows.

(a) Back-track NDFT model's preceding DFT models until to itself (loop) or another NDFT model. There could be multiple trace paths, among which the longest one determines the delays on the output of this NDFT. For example, there are four incoming paths before m^4: two go back to itself, one to m^2, and one to m^3, as Figure 1 shows. The longest one (DFT model cascaded) is the "$m^5 \rightarrow m^7 \rightarrow m^9$"; thus m^4's output delay is 3 simulation steps.

(b) For DFT model, count the maximum delay of its preceding models and then plus one delay produced by itself. For example, the maximum delay before m^5 is 3 (contributed by m^4); thus m^5's output delay is $3 + 1 = 4$.

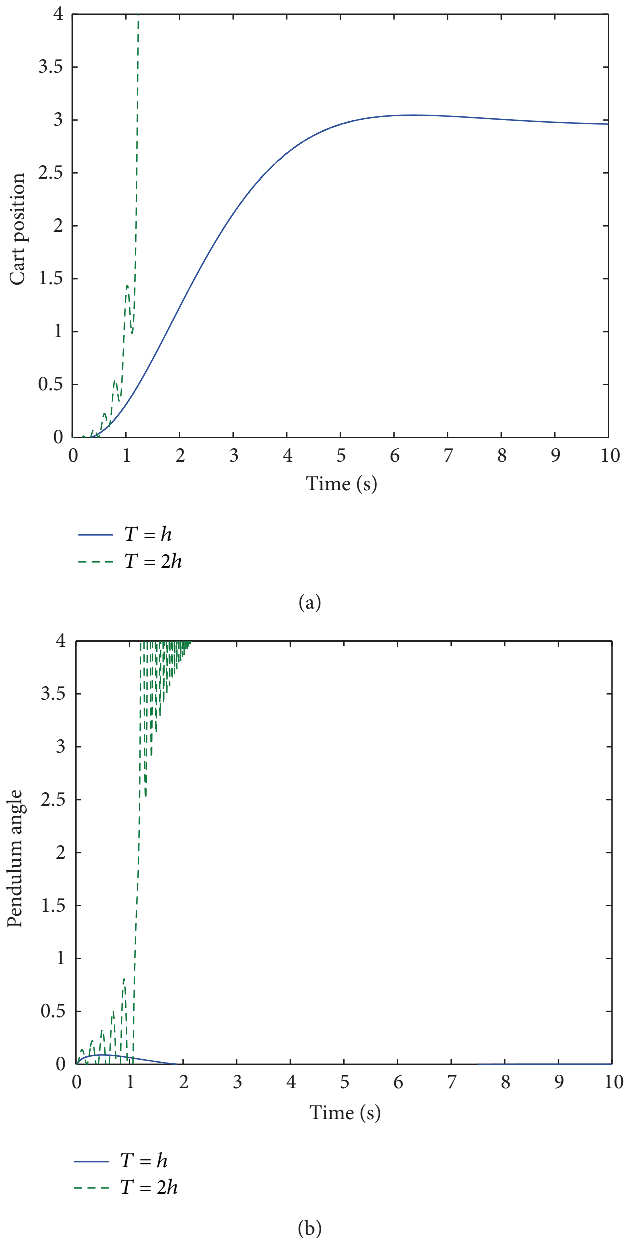


Figure 9: The models are computed with fixed-step resolver, where the step $h = 0.001$ s. When all nine submodels are totally distributed and deployed, the property of stability is volatile. As we can see, the cart position (a) and pendulum angle (b) cannot be stably controlled any more when distributed simulation step $T \geq 2h$.

3.2. Performance Improvement by Proper System Partition. Obviously, the accumulated delays are produced when distributed nodes only contain DFT models, which deteriorate the control quality. For example, in the inverted pendulum system, the pendulum is actually out of control during the time when the control signal is delayed by its preceding DFT models. Thus the system solution, that is, the state trajectories, could produce undesired errors. In some cases, it can produce instability, as Figure 9 shows.

Table 3: The output sequence of submodels in distributed simulation.

Models	Outputs	States transition
m^1	$Y_0^1 = O^1(S_0^1)$	$S_1^1 = T^1(S_0^1, \Phi, h)$,
	$Y_1^1 = O^1(S_1^1)$	$S_2^1 = T^1(S_1^1, \Phi, h)$,
	$Y_2^1 = O^1(S_2^1)$	$S_3^1 = T^1(S_2^1, \Phi, h)$,
	$Y_3^1 = O^1(S_3^1)$	$S_4^1 = T^1(S_3^1, \Phi, h)$
	$Y_4^1 = O^1(S_4^1)$	
m^2	$Y_0^2 = O^2(S_0^2)$,	$S_1^2 = T^2(S_0^2, NULL, h)$,
	$Y_1^2 = O^2(S_1^2)$,	$S_2^2 = T^2(S_1^2, Y_0^1, h)$,
	$Y_2^2 = O^2(S_2^2)$	$S_3^2 = T^2(S_2^2, Y_1^1, h)$,
	$Y_3^2 = O^2(S_3^2)$	$S_4^2 = T^2(S_3^2, Y_2^1, h)$
	$Y_4^2 = O^2(S_4^2)$	
m^3	$Y_0^3 = O^3(S_0^3)$,	$S_1^3 = T^3(S_0^3, NULL, h)$,
	$Y_1^3 = O^3(S_1^3)$,	$S_2^3 = T^3(S_1^3, NULL, h)$,
	$Y_2^3 = O^3(S_2^3)$	$S_3^3 = T^3(S_2^3, NULL, h)$,
	$Y_3^3 = O^3(S_3^3)$	$S_4^3 = T^3(S_3^3, Y_2^8, h)$
	$Y_4^3 = O^3(S_4^3)$	
m^4	$Y_0^4 = O^4(S_0^4)$,	$S_1^4 = T^4(S_0^4, NULL, h)$,
	$Y_1^4 = O^4(S_1^4)$,	$S_2^4 = T^4(S_1^4, NULL, h)$,
	$Y_2^4 = O^4(S_2^4)$,	$S_3^4 = T^4(S_2^4, NULL, h)$,
	$Y_3^4 = O^4(S_3^4)$,	$S_4^4 = T^4(S_3^4, NULL, h)$,
	$Y_4^4 = O^4(S_4^4)$,	$S_5^4 = T^4(S_4^4, Y_3^9, h)$
	$Y_5^4 = O^4(S_5^4)$	
m^5	$Y_0^5 = O^5(NULL, NULL) = NULL$,	—
	$Y_1^5 = O^5(Y_0^1, Y_0^4)$,	
	$Y_2^5 = O^5(Y_1^1, Y_1^4)$,	
	$Y_3^5 = O^5(Y_2^1, Y_2^4)$,	
	$Y_4^5 = O^5(Y_3^1, Y_3^4)$,	
	$Y_5^5 = O^5(Y_4^1, Y_4^4)$,	
	$Y_6^5 = O^5(Y_5^1, Y_5^4)$	
m^6	$Y_0^6 = O^6(X_0^6 = NULL) = NULL$,	—
	$Y_1^6 = O^6(Y_0^4)$,	
	$Y_2^6 = O^6(Y_1^4)$,	
	$Y_3^6 = O^6(Y_2^4)$,	
	$Y_4^6 = O^6(Y_3^4)$	
m^7	$Y_0^7 = O^7(X_0^7 = NULL) = NULL$,	—
	$Y_1^7 = O^7(Y_0^5 = NULL) = NULL$,	
	$Y_2^7 = O^7(Y_1^5)$,	
	$Y_3^7 = O^7(Y_2^5)$,	
	$Y_4^7 = O^7(Y_3^5)$	
m^8	$Y_0^8 = O^8(X_0^8 = NULL) = NULL$,	—
	$Y_1^8 = O^8(X_1^8 = Y_0^5 = NULL) = NULL$,	
	$Y_2^8 = O^8(Y_1^5)$,	
	$Y_3^8 = O^8(Y_2^5)$,	
	$Y_4^8 = O^8(Y_3^5)$	
m^9	$Y_0^9 = O^9(NULL, NULL, NULL, NULL) = NULL$,	—
	$Y_1^9 = O^9(NULL, NULL, Y_0^3, Y_0^2) = NULL$,	
	$Y_2^9 = O^9(NULL, Y_1^6, Y_1^3, Y_1^2) = NULL$,	
	$Y_3^9 = O^9(Y_2^7, Y_2^6, Y_2^3, Y_2^2)$,	
	$Y_4^9 = O^9(Y_3^7, Y_3^6, Y_3^3, Y_3^2)$	

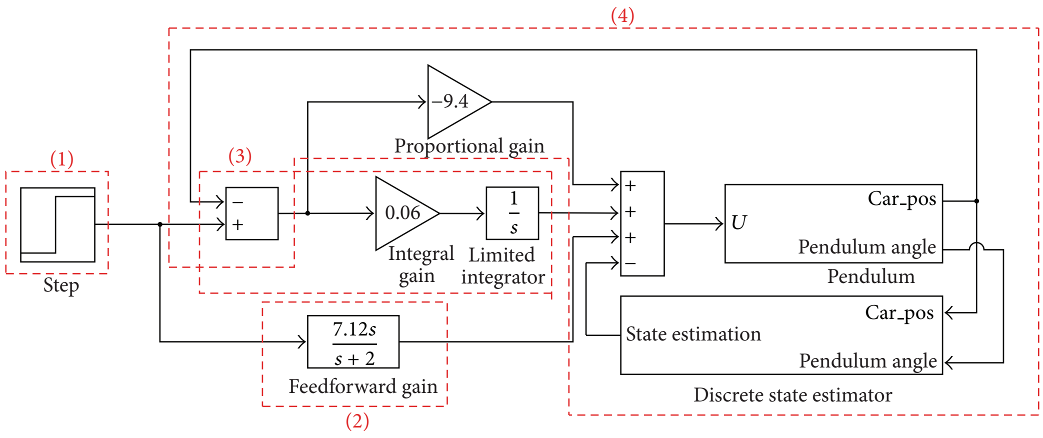


FIGURE 10: Four new submodels are formed by proposed partition schema, marked with dashed boxes and numbers. Separately deployed DFT models do not exist anymore and the accumulated delays incurred are eliminated.

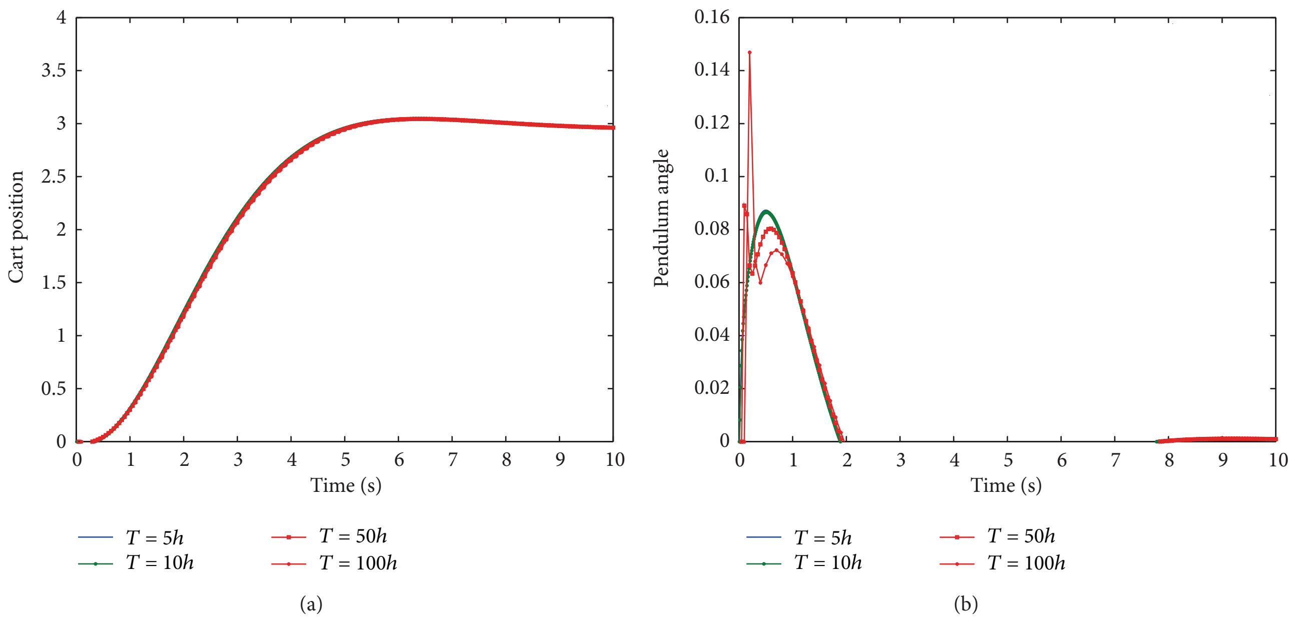


FIGURE 11: The control quality in distributed environment is improved greatly after repartition of the system. In (a), the dynamics of cart positions are almost unchanged with T increasing; in (b), although the pendulum angle is affected, it is still under control.

To reduce the delays caused by DFD models, a schema is proposed here to partition the system properly as follows.

(a) Pick up one of the NDFT models, denoted as m^{start}. Disconnect its outgoing connections and backtrack its preceding DFT models along each incoming path until a different NDFT model, denoted as m^{end}. Loop is enabled and kept as composition of this part. There could be multiple tracking paths; thus m^{end} is a set.

(b) Disconnect the connections between m^{end} set and their adjacent DFT models (along the tracking paths). This separated part forms a new integrated NDFT model.

(c) Repeat steps (a) and (b) until the system is partitioned completely.

Using this schema, the separately deployed DFT models are eliminated; as a result, the accumulated delays are eliminated either. In Figure 10, the inverted pendulum system is partitioned into 4 parts; the computing order inside each part is determined by the same way as introduced in Section 3.1.

With this partition schema, the control quality of the system in distributed environment is improved greatly, as we can see in Figure 11. The main benefit brought out by this schema is it enlarges the tolerance of simulation step T with which the distributed system can sustain the stability property of the original system. It relaxes the criterion under which the original system can be deployed into distributed environment without any modifications to models.

4. Conclusion

A simulation infrastructure, MLRTI, is proposed in this paper to address some practical issues related with real-time simulations. The integrating timing mechanism and high data transmission speed achieved with specialized hardware guarantee the performance of the infrastructure of real-time system. Additionally, the system partition schema successfully reduced the possible errors incurred by improper system distribution. With these characteristics, MLRTI is used in the following domains: (a) the virtual combat simulation domain, where computer generated entities act as friendly or rival forces and human pilots combat with or against them by simulators; (b) the distributed control domain, where different parts of the system residents on different nodes.

In the future, more theoretical work would be done to improve the system partition schema in two directions: (1) considering the load balance (computation balance and communication balance) requirement in the partition schema; (2) finding a formal approach to determine the upper bond of step size in distributed environment.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] P. Ryan, W. Oliver, and F. Hill, "Aerial refueling in a real-time distributed simulation environment," in *Proceedings of the SISO European Simulation Interoperability Workshop 2009 (EURO SIW '09)*, pp. 55–63, July 2009.

[2] M. Massei, A. Tremori, S. Poggi, and L. Nicoletti, "HLA-based real time distributed simulation of a marine port for training purposes," *International Journal of Simulation and Process Modelling*, vol. 8, no. 1, pp. 42–51, 2013.

[3] H. Zhao, Z. Wang, and L. Qiu, "Real-time distributed simulation of aircraft anti-skid control system," *Journal of Beijing University of Aeronautics and Astronautics*, vol. 26, no. 2, pp. 156–159, 2000.

[4] R. D. Lehmer and S. J. Malsom, "Distributed system architecture in VAST-RT for real-time airspace simulation," in *Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit*, vol. 2, pp. 870–879, Providence, RI, USA, 2004.

[5] J. Jatskevich, C. E. Lucas, E. A. Walters et al., "Real-time distributed simulation of DC Zonal electrical distribution system," in *Proceedings of the Power Systems Conference*, 2002.

[6] A. Russell, "A low-latency 60 Hz stereo vision system for real-time visual control," in *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, vol. 90, pp. 165–170, 1990.

[7] J. L. Bastos, J. Wu, N. Schulz, R. Liu, and A. Monti, "Distributed simulation using the Virtual Test Bed and its real-time extension," in *Proceedings of the Summer Computer Simulation Conference (SCSC '07)*, pp. 757–765, July 2007.

[8] E. Kofman, "Quantization-based simulation of differential Algebraic equation systems," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 79, no. 7, pp. 363–376, 2003.

[9] E. Kofman, "Discrete event simulation of hybrid systems," *SIAM Journal on Scientific Computing*, vol. 25, no. 5, pp. 1771–1797 (electronic), 2004.

[10] E. Kofman, J. S. Lee, and B. P. Zeigler, "DEVS representation of differential equation systems: review of recent advances," in *Proceedings of the 13th European Simulation Symposium*, pp. 591–595, 2001.

[11] G. Chan, *Key Technology Research of Distributed Flight Simulator System*, BeiHang University, Beijing, China, 2014.

[12] S. Li, T. Hui, and Z. Chen, "Research of schedule strategy in LAN-based distributed real-time simulation," in *Proceedings of The Asian Simulation Conference on System Simulation and Scientific Computing*, pp. 346–352, Shanghai, China, November 2002.

[13] J. Wang, Z. Liu, and J. Ma, "The study of the managing node redundancy of real-time industrial ethernet POWERLINK," in *Proceedings of the 2nd International Conference on Measurement, Instrumentation and Automation (ICMIA '13)*, Guilin, China, 2013.

[14] G. Li, X. Yao, and K. Huang, "Study on real-time distributed simulation management," in *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS '07)*, vol. 2 of *Lecture Notes in Engineering and Computer Science*, pp. 1557–1560, March 2007.

[15] C. Mensah-Bonsu and G. T. Heydt, "Real-time digital processing of GPS measurements for transmission engineering," *Power Engineering Review*, vol. 22, pp. P177–P182, 2003.