Cluster Comput
DOI 10.1007/s10586-015-0498-9

An efficient approach to collaborative simulation of variable structure systems on multi-core machines

Chen Yang[1] · Peng Chi[2] · Xiao Song[3] · Ting Yu Lin[2] · Bo Hu Li[2] · Xudong Chai[2]

Received: 19 April 2015 / Revised: 30 September 2015 / Accepted: 3 October 2015

Abstract Complex variable-structure systems (CVSSs) are a common type of complex systems that exhibit changes both at structural and behavior levels. Simulations of CVSSs challenge current collaborative execution methods with increasingly big and complex models. The emergence of multi-core paradigm presents an exciting opportunity to address such challenge, so an advanced parallel simulator under multi-core environments is proposed. The simulator: (1) provides thread simulation kernels and five kinds of management services to support dynamic model structure flexibly; (2) can explore both inherent and dynamic parallelism among models based on interaction relations, and employ the multi-thread paradigm to gain good speedup; (3) adopts an efficient dynamic load-balancing method, which can migrate models among cores with very low cost and support dynamic core allocation on demand, to address evident load-imbalance problems brought by variable-structure. The experiments show that structure changes can be supported while up to 23 % performance increase can be gained.

Keywords Multi-core · Parallel discrete event simulation · Variable structure · Dynamic load balancing · Collaborative simulation

✉ Chen Yang
cyang337@uwo.ca

Xiao Song
songxiao@buaa.edu.cn

[1] Department of Electrical and Computer Engineering, University of Western Ontario, London, ON, Canada

[2] Beijing Simulation Center, Beijing, People's Republic of China

[3] School of Automation Science and Electrical Engineering, Beihang University, Beijing, People's Republic of China

1 Introduction

Many complex systems consist of a large number of components which adapt or learn as they interact [1], so that the systems exhibit eminent overall properties, such as emergence, nonlinearity, self-organization, chaos or gaming [2,3]. Among them, complex variable-structure systems (CVSSs) are systems that have variable constituents and interaction structure during the execution. In other words, CVSSs exhibit changes simultaneously at structural and behavior levels [4,5], when evolving over time. Typical CVSSs are widely distributed in the areas of complex engineering, sociology, and ecology. Commonly, where there are autonomous and interactive entities, system changes in interactions and constituents occur frequently.

Parallel discrete event simulation (PDES) involves the use of multiple processors to collaboratively simulate and analyze large-scale systems in shorter time. However, using common PDES methods, a CVSS is mostly modeled as the "complete" and static-structure system, in which all possible models, interaction ports and connections exist and participate in the simulation since the beginning stage and could not be changed easily and flexibly during the execution period. Furthermore, models need to be designed to only deal with related events during the right intervals and discard unrelated events caused by redundant components, ports and connections. Two problems will arise. First, the efficiency of simulation executions would be severely degraded, if the structure changes a lot (especially for large-scale simulation systems), as all the components and the "complete" interaction structure need to be loaded before the simulation starts and a lot of redundant events flow in the simulation system. Secondly, users may be unable to exactly predict all the possible models, ports and connections at the beginning, especially for the evolving CVSSs.

Published online: 20 October 2015

Researchers try to employ variable-structure theory and technology in PDES [4,6–8] to address the above problems. Variable structure PDES (VSP) refers to the PDES that supports the change of structure (constituent models, model ports or connections between models) and parameter changes to well simulate CVSSs' behavior. However, current PDES methods mostly have two drawbacks to deal with increasingly big and complex models of CVSSs [16].

(1) *Current VSP methods largely overlook the efficiency issues, which can lead to unbearably long simulation time.*

The extended formalisms of discrete event system specification (DEVS [9]) aiming to improve the expressiveness, such as DYNDEVS [4], DSDE [6], ρ-DEVS [8] have been proposed to support the modeling of variable structure systems. Most of these work focuses on the theoretical aspect of VSP. Hu et al. [7] further introduced the concept of variable "ports" and enabled structure changes (reconfiguration of simulation systems) by adding extra supporting services to the DEVS-based simulation environment. However, these work did not adopt popular methods, such as flattened simulation structure [10,11] and load balancing [12,13] to improve the efficiency. On the other hand, the emergence of multi-core paradigm provides an exciting opportunity to improve the efficiency of VSP, by enabling fine-grained parallelism of code execution and supporting low-latency communication [16]. However, very few research results of VSP under multi-core environments have been done.

(2) *Current PDES methods adopting multi-core technology mostly could not support the VSP.*

Most cutting-edge simulation applications are trying to employ high performance computing to solve challenging problems, such as the aerospace vehicle analysis and design [14] on the Columbia supercomputer in NASA, the earth simulator project [15] in Japan, the human brain project in European Union. Thus it is of great importance to improve the capacity of simulating large-scale CVSSs by utilizing the latest technologies of high-performance computing. The latest advances in the PDES community such as [16–19] attempt to exploit multi-core computers, but mostly, they cannot support the simulation systems featuring dynamic structure, as they cannot both allow new models and connections to be created and added to the existing network of models dynamically, and eliminate possible straggler events that flow through newly-added connections.

Overall, there is a lack of flexible, efficient and systematic methods for the PDES of CVSSs to use multi-core technology.

We only discuss conservative simulation in this paper, because it is usually hard or costly to save the whole state of models for rollbacks [20,21] that are common in optimistic simulation, e.g. large continuous models, models built by using commercial software, legacy models or other models containing irrevocable operations. The algorithms and methods presented in this paper can also be applied to common PDES for static-structure models, which can be deemed as special cases of variable structure models. It is pervasive that a high performance computer equips with tens or hundreds of cores, which means that a computer alone can provide powerful computing power for some large-scale simulation applications, so we only discuss the methods on multi-core machines, though the proposed methods can be adapted to multi-core clusters.

The paper is organized as follows: Sect. 2 reviews related work and existing problems. Section 3 briefly introduces the parallel simulator-Ivy which our methods are based on. Sections 4 and 5 elaborate on the detailed methods. Section 6 presents the experiment result. Finally, Sect. 7 gives the conclusion.

2 Related work and existing problems

The physical system is viewed as comprising some number of physical processes (PPs) that interact in some fashion [21]. Each PP is modeled by a logical process (LP) and interactions between PPs are modeled by exchanging time-stamped messages between the corresponding LPs. Thus a PDES of the physical system is typically composed of a collection of collaborative LPs. Each LP performs computations to process events, leading to the modification of state variables and/or the scheduling of new events for itself or other LPs. If a LP at simulation time T can only schedule new events with time stamp of at least $T + L$, then L is referred to as the *lookahead* for that LP. The lower bound time stamp (LBTS) of a LP is essentially the smallest timestamp of events that may be received in the future. If events within each LP have been processed in time stamp order, then it is sufficient to ensure that the parallel simulation will produce exactly the same results as the corresponding sequential simulation [21]. This is the theoretical basis of PDES.

2.1 Variable structure PDES

DYNDEVS and its simulator [4,22] can only support the creation and the removal of models and connections between models. Uhrmacher et al. [8] proposed a new formalism ρ-DEVS by extending DYNDEVS with variable ports. Muzy and Zeigler [23] proposed a coherent framework for common dynamic structure formalisms. Barros [24] compared the advantages of centralized and distributed policies to

enforce dynamic structures. However, these works did not discuss the efficiency aspect of their hierarchical simulation approaches [10,11], such as flattened execution structure and load balancing. Moreover, they cannot take full advantage of multi-core architectures. To the best of our knowledge, our work-Ivy [25] is the first high performance simulator that is capable of VSP on multi-core machines.

Thus, an efficient method, supporting comprehensive structure changes in a distributed way, a centralized way, or a hybrid way, should be developed.

2.2 High performance simulator

2.2.1 Execution architecture

ThreadedWarped [26] adopted a master-slave architecture: a manager thread, a global event queue of the simulation objects and several worker threads. Vitali et al. [27] presented a load-sharing approach by allocating the computing power to multi-threads dynamically. Chen et al. [28] proposed a global scheduling mechanism using several event lists and several active worker threads, each of which selects and processes the earliest event in current event lists repeatedly. Tang et al. [19] proposed a hierarchical parallel simulator (for multi-core clusters) which schedules LPs to process safe events in parallel with multi-threaded operating system processes (OSPs). The hierarchical architecture consists of the OSP level and the thread level. Wang et al. [16] proposed a thread-based simulator ROSS-MT that can avoid multiple message copying in the same OSP. Lin et al. [17] introduced an optimistic and thread-based simulator which creates several priority queues within one OSP and maps a subset of the threads to a single queue to decrease the contention and improve performance. Bauer et al. [18] proposed a technique to control optimism of PDES in which each LP continuously communicates time estimates of its next k outgoing inter-LP events to its neighboring LPs and uses time estimates from its neighbours as bounds for its time advance. However, most of these simulators [16–18,26–28] adopts optimistic synchronization algorithms and we could not find evidence that these simulators allow new LPs and connections to be established and added to the existing network of LPs during the execution and contain proper mechanisms to deal with possible straggler messages sent through new connections.

2.2.2 Time management algorithm

Conservative simulation strictly avoids any occurrence of causality errors, so time management algorithms need to determine whether the events are safe to process. There have been many conservative algorithms [29], most of which are based on lookahead and LBTS. However, common conservative algorithms are not applicable to the VSP, as when new directed connections are added, the downstream LPs may receive events with past time stamps. This should be addressed to guarantee all the events of each LP are processed in time stamp order.

As for the efficiency aspect, more related researches are [19,30]. Peng et al. [30] adopted a multi-threaded architecture for each federate, but the information of all sub-models in a federate are needed to compute unified LBTS for all these sub-models. Tang et al. [19] proposed a similar algorithm to compute approximate LBTS in an asynchronous way. The thread that has finished the processing of current safe events for its models will seek to acquire the unique handle to initiate the computation of new approximate LBTS and then share the LBTS inside the federate. However, these two methods largely ignore the model structure (could not capture structure changes) and thus constrain the extraction of inherent and dynamic parallelism among models, especially when the lookahead of models differ greatly and the models are sparsely connected. Some work uses application-specific information, such as model structure, to improve the performance of conservative algorithms, but they cannot cope with minor changes to the model [29]. We have not yet seen application-independent algorithms that are capable of using model structure to extract the parallelism of models.

Thus a multi-threaded simulator with good time management algorithms is preferred in order to gain good performance.

2.3 Load balancing

Load imbalance on multi-cores can lead to severe degradation of the system performance [31], as the risk of blocking fast LPs for a long time to wait slow LPs greatly increases in conservative simulation. Research on load balance of PDES can be broadly divided into two categories: (1) metrics for detecting load imbalances and deciding about LP movements; (2) protocols or mechanisms to support load migration.

For the first set, there have been many algorithms [13]. Glazer and Tropper [32] proposed the simulation advance rate, which is calculated using the CPU allocation and the virtual time advance of processors. Jiang et al. [33] generalized Glazer's algorithm by considering heterogeneous processors and background load. Peschlow's metric of computation load [31] depended on the number of events processed and the effective time advance. The model structure/interaction can be used to build model groups to facilitate load balancing [36,37]. Overall, these work largely has an underlying assumption that the structure of simulation systems does not change. Structure changes require that loads of new models are considered and evaluated, and loads of removed models are neglected, to get a precise load distribution, so current metrics need to be extended.

For the second set, comparing with current load balancing methods among nodes or CPUs, balancing load among cores allocated to an OSP does not necessarily mean copying and transferring of model state, as LPs can be created as shared models inside an OSP. Then the efficiency of load balancing can be substantially improved with proper algorithms. A more related work is [30]. Peng et al. [30] proposed a preliminary load balancing method by transferring the state of models between threads in an OSP, but this can be time-consuming when a large number of LPs need to be migrated.

Besides, PDES of CVSSs needs the capability of dynamical addition and release of computing resources. Basically, the quantity of processor time required to process an event for a LP may change during the simulation, and so does the event population. Dynamic addition (or removal) of models to (from) the simulation system for CVSSs further makes such functions more significant, because of load changes. Vitali et al. [27] and Carothers et al. [38] did some useful work on this aspect, but they did not address the problems caused by structure changes, for example, how to evaluate the load of new models and how to detect internal workload imbalances caused by a sharp change in the simulation communication pattern.

To address the above problems, a parallel simulator-Ivy and a high efficient load balancing method on multi-core machines are proposed, based on our initial work in [25,39]. Adopting our method, Ivy can support dynamic structure in a distributed, a centralized, or a hybrid way; deep extraction of the parallelism between models; and fine-grained parallel execution of models. Moreover it can migrate unbalanced load between cores without pausing the whole system and without copying the state of LPs, and can add or release cores on demand. The comprehensive experiment shows that our simulator can achieve better performance.

3 Introduction of Ivy under multi-core environments

3.1 Basic principles of Ivy

We have proposed a parallel simulator-Ivy under multi-core environments to enhance the ability to naturally and effectively simulate CVSSs which are normally large in scale [25]. Component-based models in Ivy's model base are named as simulation component models (SCMs), which can be instantiated with initial parameters into different Component Model Instances (CMIs). A standard framework of the SCM is defined in [25] to guide the implementation of formal SCMs. A standard SCM mainly includes a unique ID, a user model, common attributes, and management interface. The user model can be any domain model built by users for their specified business. The common attributes include input/output ports, simulation time, lookahead, and input/output/waiting event lists. The management interface contains the common interface that is necessary for Ivy to schedule CMIs.

Variable structure of a simulation system includes the change of model ports, constituent models and connections between models. Basic principles of structure changes are as the following.

Different data protocols that a CMI uses to communicate with others are abstracted as ports, in order to (a) increase the simulation capability for variable-structure systems, and (b) loose the coupling between CMIs and improve model reusability. The port lists and the corresponding operation methods (*addPorts* and *removePorts* in model's unified management interface *MgrInterface* [25]) are formally included in the model implementation to support variable ports. Via variable ports, significant changes inside the models can be signalized to the external world, which is particularly important in the molecular biological domain [8].

The interaction between CMIs can be described by oriented connections between ports of CMIs. Directed connections are independent from model implementations, but it should be guaranteed that the type of source ports match that of destination ports. The change of connection relationships between models can be used to naturally simulate the change of the system network, which can help to study complex systems with variable interaction structure.

After all connections related to a model are removed, the model can be removed to change the component parts of the simulation system. Similarly, models can join the simulation if they are created, and the related connections and the model references are added to the connection management and Simulation Engine Instances (SEIs, see Sect. 3.2) respectively. However, when necessary, some inactive models can be kept in the heap and later initiated as new models to deal with the frequent change of models in the system, in order to improve the performance of dynamic structure simulation.

Example Airplanes will normally fly across different air-traffic-control areas and need to interact with different control centers to guarantee a safe flight. This can be naturally modeled by changes of connections between airplane models and control center models (cf. Fig. 1). Airplanes will fly over an air-traffic-control area when it becomes far enough. If the attention is paid to the operation of only one air-traffic-control center, then the airplane models should be added to or removed from the simulation system.

3.2 Multi-thread execution architecture and life-cycle

The multi-thread execution architecture of Ivy is shown in Fig. 2. Ivy and models-CMIs compose the simulation system. Ivy executes as an OSP, which creates a collection of thread-level SEIs. One or more CMIs, which are initiated

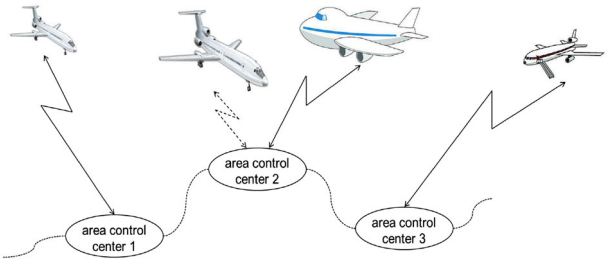

Fig. 1 Variable interaction structure

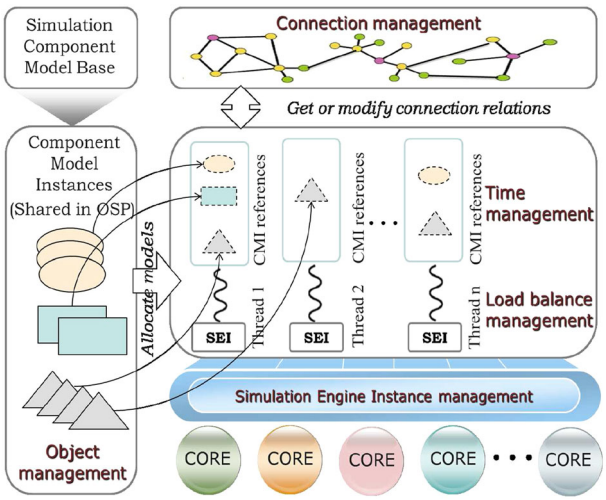

Fig. 2 Multi-thread Execution architecture of Ivy

from the SCM base and allocated to a SEI, form the CMI-reference list of that SEI. The SEIs are created in the form of threads to schedule CMIs on their own CMI-reference list. To efficiently support simulation of variable structure systems, Ivy provides five kinds of core services: object management, connection management, simulation engine instance management, time management and load balance management, by making good use of the multi-threaded paradigm and the communication mechanism based on shared variables. The main thread controller-actually the main thread of the OSP is designed to respond to user requests, and to configure and control the simulation experiment using the above core services of Ivy.

The life-cycle of Ivy, represented mainly by the main thread controller and the SEIs, is discussed as follows:

3.2.1 Main thread controller

The main thread controller initiates Ivy's core services, which play their roles as the following:

(a) Object management loads the SCMs and instantiates them as the CMIs according to the application demand. Object management can create or delete the CMIs dynamically, when the components of the simulation system needs to be changed.
(b) Connection management initiates the network of the simulation system by loading the interaction model of the system and maintains directed connections between ports of the CMIs.
(c) Simulation engine instance management creates, initializes, starts, pauses and terminates the simulation engine instances (SEIs). The SEIs schedule the event-processing of the CMIs in parallel and are responsible for passing events according to directed connections. Time management is utilized to synchronize the CMIs in the simulation system.
(d) Load balance management migrates the CMIs between the SEIs, or even employs SEI management to release certain in-use cores or add additional cores on demand, in order to improve the efficiency of the collaborative scheduling of the LPs on cores.

3.2.2 Simulation engine instance (SEI)

Once created, a SEI will schedule the CMIs on its CMI-reference list repeatedly. The SEI will:

(a) Schedule the next CMI on the list to calculate the LBTS of the CMI and read safe events from the CMI's *inputList* to *waitingList* under the control of the LBTS,
(b) Sort events in *waitingList* of the CMI in time stamp order,
(c) Schedule the processing of the earliest event in *waitingList*,
(d) Advance CMI's simulation time to the timestamp of the processed event,
(e) Send newly scheduled events caused by event processing to *inputList* of the destination CMIs, according to connections.
(f) Repeat step (c)–(e) until no events exist in *waitingList*, otherwise go to step (a).

4 Efficient structure change and time management algorithm

Section 4.1 introduces how dynamic and distributed structure changes of models are achieved. Section 4.2 presents the synchronization algorithm for VSP, but we only elaborate on how it correctly synchronizes the models, exploits the fine-grained inherent parallelism between the models and achieve the deadlock avoidance, in the static-structure simulation system. Then based on the results in Section 4.2, Section 4.3 further discusses how the proposed algorithm and extra mechanism work in VSP, for example how the deadlock is avoided in VSP.

Table 1 Locks used in Ivy

Lock name	Locked object	Affiliated to	Contention by
CRL_RW_lock	CMI-reference list	SEI	MTC, SEIs, LB
IPL_RW_lock	input port list	CMI	MTC, SEIs
OPL_RW_lock	output port list	CMI	MTC, SEIs
IEL_RW_lock	input (event) list	CMI	MTC, SEIs
TV_RW_lock	time variable	CMI	MTC, SEIs
CC_RW_lock	CMI connections	CM	MTC, SEIs

CM connection management, *MTC* main thread controller, *LB* load balance management

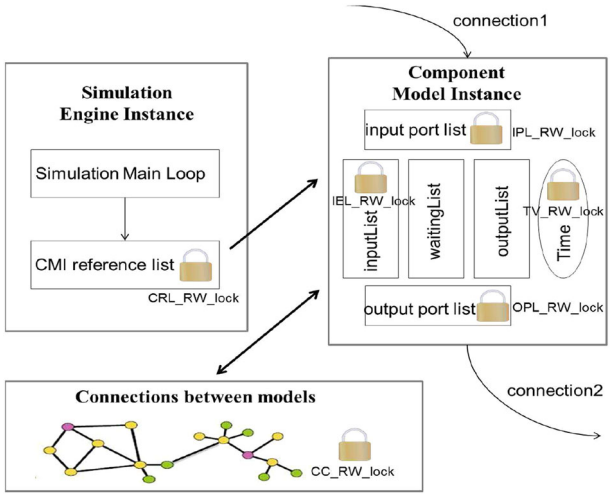

Fig. 3 Locks used in Ivy

4.1 Dynamic and distributed structure change

Due to dynamic structure, multiple threads may try to read and modify the same information of the simulation system simultaneously, so locks are used to guarantee safe concurrent access to the shared resources among threads [25]. We made some optimizations by redesigning some data structures and removing some locks based on [25] (Table 1).

The locks (cf. Fig. 3) are used to avoid resource contention. All these locks are readers-writer locks. A readers-writer lock allows concurrent access for read-only operations but requires exclusive access for write operations.

4.1.1 Control of structure change

As shown in Fig. 4, the models in a simulation system can be divided into several groups, in each of which a control model of variable structure is responsible for issuing orders of structure changes. Each SEI acting as a scheduling center for its CMIs, can execute structure changes under the regulation of related locks, so the structure can be changed in a distributed and concurrent way. If only the models (or even only one model) scheduled by a certain SEI are programmed with

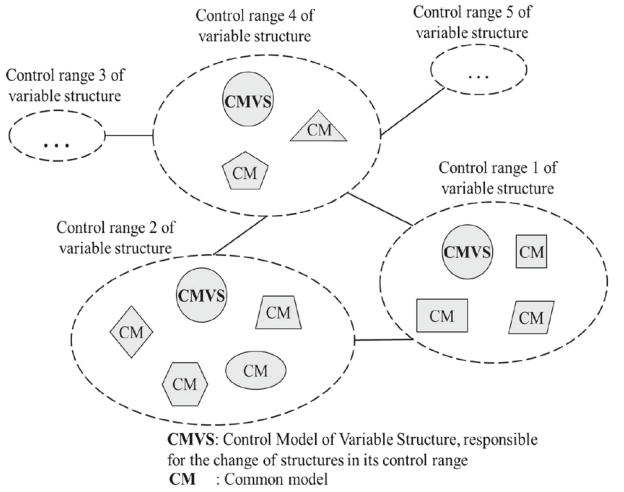

Fig. 4 Distributed control of variable structure

the structure-change ability, then the structure change will be performed in a centralized way. Another extreme would be that each model is only responsible for structure changes related to itself, like an agent in complex systems.

IPL_RW_lock and *OPL_RW_lock* can ensure safe concurrent access to the input port list and the output port list respectively. The addition and removal of model ports are used to simulate the evolving phenomenon of subjects with new output/input. Port changes can be initiated by the CMI itself to exhibit internal state changes by different ports (i.e. reflection [4,8]), or even by other CMIs to exert external influence.

Connection management uses *CC_RW_lock* to control reading and writing operations of connections among models. Current connections to a CMI are utilized to acquire the simulation time and lookahead of CMIs on the other side of those connections to compute the LBTS of that CMI. With *CC_RW_lock*, connections can be changed without interrupting the execution of unrelated CMIs, while the potential risk of access violations is eliminated. When connections are being changed, read operation will be suspended until the alteration has been finished and vice versa, but multiple read operations on connections are allowed.

CRL_RW_lock and certain procedures are proposed to guarantee the safe removal and addition of CMIs. Three steps to remove a CMI include: 1) delete the reference of this CMI from the CMI-reference list of a certain SEI, 2) delete related connections of the CMI, and 3) delete the CMI. The process to add a CMI is largely reversed except some special treatment on time management is needed (see Sect. 4.2).

4.1.2 Efficiency and flexibility of variable structure

Our proposed method can achieve smooth and safe structure changes with little intervention, instead of pausing the whole

simulation. For large-scale simulation running on tens or hundreds of computing nodes, this can substantially improve the efficiency, due to that any unrelated CMI can be scheduled normally. As stated in [40], variable structure model can make the simulation more efficient, due to the focus only on active models without the burden of all models always active in the system. However our work is a starting point, more research effort should be paid to gain higher efficiency.

The method also exhibits good flexibility for users to simulate the complex system: support both the distributed way and the centralized way (autonomy and control [41]) to change model structures. The way and algorithms to control structure changes are leaved to users. Comparatively, Barros [6] defined the dynamic structure system network using the DEVS formalism, but the work based on the vision of an executive that resides as a kind of all-mighty atomic model in the coupled model [42], showed limited capability to deal with complex systems that consist of autonomy entities and is essentially a centralized mechanism of controlling structure changes [43].

4.2 Time management algorithm

Only the CMIs that have connections to CMI i currently are considered to compute $LBTS(i)$, as only the CMIs that send events to CMI i can affect the time advance of CMI i. The influence of structure changes on time management is ultimately reflected on the change of connection relationships. Thus $LBTS(i)$ can be computed, according to the current connection relationship, as

$$LBTS(i) = min\{T(j) + LA(j)\} \tag{1}$$

in which CMI i receives messages from CMI j, $LA(j)$ is the lookahead of CMI j, and $T(j)$ is subject to the constraint ($T_c(j)$ is the current simulation time of CMI j):

$$T(j) = \begin{cases} T_c(j) & \text{if processing safe events} \\ LBTS(j) & \text{if no safe events} \end{cases} \tag{2}$$

The transient messages do not exist in Ivy, because the scheduled events (actually the references to these events) are directly written into *inputList* of destination CMIs and the SEI thread in charge of sending an event will never return until the write operation is finished.

4.2.1 Correct synchronization of the static-structure model

We will first check whether the above algorithm can correctly synchronize all CMIs in the static-structure system. To facilitate the proof, we define the following variables:

T_a: the timestamp of event a sent to CMI i

$T_b(i, j)$: the timestamp of event b sent from CMI i to CMI j

$T_c(i)$, $T_c(j)$: the current time of CMI i, CMI j

$LA(i)$, $LA(j)$: the *lookahead* time of CMI i, CMI j

Event b is scheduled by CMI i, after the process of any event a. Assume b should be sent to CMI j, in other words CMI j receives messages from CMI i. so

$$T_c(i) = T_a \tag{3}$$

$$T_b(i, j) \geq T_c(i) + LA(i) \tag{4}$$

$$\begin{aligned} T_c(j) &\leq LBTS(j) \\ &= min\{T(m) + LA(m)\} \\ &\leq T_c(i) + LA(i) \end{aligned} \tag{5}$$

CMI i advance its current time to $T_c(i)$, after event a is processed, i.e. Eq. (3). CMI i can only schedule events with time stamp not less than $T_c(i) + LA(i)$, i.e. Eq. (4).

Because there are no transient messages that arrive with the past time stamp, and the scheduling of events is constrained by Eqs. 1 and 2, so Eq. 5 is correct. m refers to any model that has connections to model j. So $T_b(i, j) \geq T_c(j)$, i.e. CMI j will not receive straggler messages. Then by processing received events in time order, CMI j conforms to the local causality constraint [21]. Event a is scheduled to CMI i, so it can be proofed that CMI i conforms to the local causality constraint similarly. Thus all CMIs are synchronized in the static-structure system.

4.2.2 Fine-grained inherent parallelism

This algorithm could exploit the parallelism by only taking into account of related models through acquiring current connection relations (by connection management) to compute the LBTS of a model. Traditional conservative algorithms base on global reductions to derive a unified LBTS, so its $LBTS_G$ equals $min\{T(j) + LA(j)\}$, for each model j in the simulation. Time advance of each model is constraint by $LBTS_G$, i.e. $LBTS(i) = LBTS_G$. In our method, the SEIs schedule the LBTS computation for their own models in parallel. $LBTS_{Ivy}(i) = min\{T(j) + LA(j)\}$, for each model j which will send messages to model i. Thus we can derive that $LBTS_{Ivy}(i) \geq LBTS(i) = LBTS_G$, i.e. our algorithm can get the fine-grained inherent parallelism to facilitate parallel scheduling of models.

4.2.3 Deadlock avoidance in the static-structure system

Conservative time management algorithms may lead to a deadlock, manifesting that some CMIs cannot advance their local time anymore. During the period between the times of two consecutive structure changes, the simulation system

will not change its connections and models. We will first see the properties of such static-structure system.

Theorem 1 *If the system encounters a deadlock, there must exist cycles.*

Proof We assume that there is no cycle in the deadlocked simulation system. Then the system can be abstracted as a directed acyclic graph (DAG), in which nodes represent CMIs and directed connections indicate interaction relationship. The DAG of the simulation system can be topologically sorted as a linear array, so that if the array is aligned in a row, all connections (edges) are directed from the left nodes (vertices) to the right ones. The nodes in the n-th position are denoted as CMI n. Intuitively if the upstream nodes (actually CMIs) do not stop the time advance, the downstream nodes will not encounter a deadlock in a DAG using our proposed algorithm, because the upstream senders (nodes) that eventually advance their local time make LBTS of the downstream nodes become larger and larger, so that the downstream nodes can also advance to anywhere in the future in a finite amount of time. We will proof it strictly as the following. □

Lemma 1 *Any CMI n can advance to anywhere in the future in a finite amount of time, if the simulation is not over.*

The lemma can be translated into the mathematic description as:

Lemma 2 *Given any $M > 0$, a T_M, subjected to $T_M > 0$, can be found, so that after the wall clock time T_M, the virtual time of CMI n, $T(n) > M$.*

Proof Strong induction is used to proof this lemma. □

Basis: $n = 1$, CMI 1 has no senders, so $LBTS(1) = +\infty$. If there are unprocessed events, it is safe to process them without a deadlock and eventually advance its local time to anywhere in the future if the simulation is not over. If no events exist, $T(1) = LBTS(1) = +\infty$. Thus Lemma 2 holds for $n = 1$.

Induction step: assume Lemma 2 holds for $n \leq m$.

When $n = m + 1$, given any $M(m+1) > 0$, $LBTS(m+1) = min\{T(h) + LA(h)\}$, for any upstream CMI h $(0 < h \leq m)$ that connects to CMI $m + 1$.

According to the induction step, for any $\delta > 0$, $M(m+1) + \delta > 0$, a $T_M(k) > 0$ for each CMI $k(0 < k \leq m)$ can be found, so that after $T_M(k)$, $T(k) > M(m+1) + \delta$.

We set $T'_M = Max\{T_M(k)|0 < k \leq m\}$, so after T'_M, $T(l) > M(m+1) + \delta$, for any CMI l that $0 < l \leq m$. And we assume that CMI p has the smallest $T(p) + LA(p)$ among CMI l that $0 < l \leq m$. Then

$$\begin{aligned} LBTS(m+1) &= min\{T(h) + LA(h)\} \\ &\geqslant min\{T(l) + LA(l)\} \\ &= T(p) + LA(p) \\ &> M(m+1) + \delta \end{aligned} \tag{6}$$

so CMI $m + 1$ can process any events with timestamp less than $M(m+1) + \delta$. When no events exist, according to Eq. 2 and Ineq. 6,

$$\begin{aligned} T(m+1) &= LBTS(m+1) \\ &> M(m+1) + \delta \\ &> M(m+1) \end{aligned}$$

Thus for $n = m + 1$, Lemma 2 holds.

Thus using strong induction, we can infer that the lemma holds. So for any CMI, if any event exists, it will be processed after a finite amount of time. This contradicts to the deadlock assumption. So alternatively there must be cycles in the deadlocked system.

Lemma 3 *Upstream nodes not on cycles can advance to anywhere in the future in a finite amount of time.*

Proof These nodes and their upstream nodes consist of a DAG, so the lemma can be deduced from lemma 1. □

Theorem 2 *With our time algorithm there is no deadlock on cycles.*

Proof Assuming that there are unprocessed events, and one of the earliest events is contained in CMI i. Due to the constraint of LBTS, no event can be safely processed. So $T(k) = LBTS(k) = min\{T(l) + LA(l)\}$, for any CMI k on cycles. When deadlocking,

$$\begin{aligned} LBTS(i) &= min\{T(j) + LA(j)\}_{j \to i} \\ &= T(k) + LA(k) \\ &= min\{T(l) + LA(l)\}_{l \to k} + LA(k) \\ &= \cdots \\ &= LBTS(i)_{latest} + LA(m) + \cdots + LA(k) \end{aligned} \tag{7}$$

Because all the upstream nodes can have enough big LBTS after some time, so when deadlocking, all CMI j in Eq. 7 belongs to the cycles. Due to finite nodes on the cycles, the node that determines the time advance of its direct downstream nodes can be ultimately traced to CMI i itself. If any CMI on the cycle has $LA(j) > 0$, $LBTS(i)$ can increase gradually and its unprocessed events will certainly be processed after some time. Thus the earliest time of events on the cycles increases. This contradicts with the assumption that the system is deadlocked. So no deadlock exists on the cycles. We can infer from the process of proof that this efficient algorithm has the same restriction as the null message algorithm: there should be no zero-lookahead cycle [21]. □

Theorem 3 *Downstream nodes will not encounter a deadlock.*

Proof Due to the incremental simulation time of nodes on the upstream and cycles, it can be easily inferred that with enough time advance of nodes on the upstream and cycles, the downstream nodes can process any future event. So Theorem 3 holds. □

If all the nodes do not have events to be processed, then the simulation ends. Otherwise all the nodes can process any future event after a finite amount of time, i.e. the simulation system adopting our time management algorithm can avoid the deadlock.

4.3 Relations between time management and dynamic structure

Connection relations are the core part of the time management shown in Sect. 4.2, as the algorithm acquires the set of models-CMI j only through directed connections. The set of models-CMI j in the Eq. 1 may change, due to the change of connections.

4.3.1 Dynamic parallelism

When the interaction structure changes, new parallelism (we call it dynamic parallelism) may emerge. For example, when an airplane flies over an air control area, the interaction between the airplane model and the control center model does not exist anymore, and thus the two models can be scheduled totally in parallel. Our algorithm deals with such situation in the way that if some connections to CMI i are removed, the CMIs on the other side of these connections are no longer considered to compute $LBTS(i)$, so that such dynamic parallelism between CMI i and other CMIs can be naturally acquired. The effects of removing the CMIs on time synchronization are indirectly exerted by the removal of related connections. The removal of senders (CMIs) or connections only makes time constraint on recipients (CMIs) relaxed, so the normal scheduling of recipients is not influenced.

4.3.2 Special treatment on new connections

The removal of connections or models increases the parallelism between models, while the addition of connections or models might bring straggler events or lead to sudden decreases of the LBTS of the downstream models. Actually, an optimistic algorithm can deal with straggler events by the rollback mechanism to guarantee strict synchronization, but it is impractical for many reasons (see Sect. 1). Normally, in conservative simulation either receivers or senders are constrained to eliminate straggler events [21]. In the first approach, receivers are prevented from advancing too far ahead of all potential sending models, so that receivers will not receive messages in their past. However, the assumption of this method is that we know all potential senders for any receiver in advance. Even if such assumption holds, receivers will advance slowly by taking account of all potential senders. The window-based method described in [21] is an extreme of this approach by introducing a time window of size T_w to prevent any LP from advancing more than T_w units of time ahead of any other LP. The second approach allows LPs with greater flexibility to advance further ahead of others but provides less control due to, in general, no limit on how large $T_R - T_S$ (the initial *lookahead* on the new connection) can be [21], but *lookahead* has to be changed to the normal value after the initial transition period. Rajaei et al. [44] proposed an idea by delaying the timestamp of straggler messages (to be not smaller than the current time of the receivers-CMIs) without the modification of *lookahead*.

Comparatively, we define two kinds of events: the essential and the non-essential events. The essential events can exert significant effects on other models, while the non-essential events are kinds of events that can be discarded like in the parallel discrete event simulation of continuous systems [20], for example the periodical state-reporting events. Events with time stamp less than the simulation time of a CMI can either be adopted by delaying its time stamp like in [44] (or even NOTIME, NOTIME which accepts straggler events as correct is another extreme algorithm without any synchronization [45].) or simply be discarded according to the tag that denotes the significance of the event. Our algorithm leaves this kind of decisions to users and Ivy can automatically dispose straggler events when reading input events of a model.

A more recent method [20] only stores the straggler event with the latest timestamp for each input port of a LP and the arrival of a new message will override the event no matter if the LP was able to read it or not. Ivy can also support this method simply by removing straggler input events except the latest one when reading the input events from *inputList* for a LP.

The effect caused by a sudden decrease of LBTS of downstream models can be eliminated by discarding the decreased LBTS and keeping the old one. When the smaller LBTS is not accepted, the corresponding CMI will not be able to advance and will wait until its new senders advance to proper time (under the constraint of Eq. 1). Time advance of the new CMIs is driven by the processing of received events.

4.3.3 Deadlock avoidance in the dynamic-structure simulation system

Section 4.2 presents that our time management algorithm can guarantee the deadlock avoidance of static-structure

simulation systems, which can actually be considered as variable-structure systems during the period between two consecutive structure changes. After new connections or models are added, the immediate downstream models may be affected, not being able to get a bigger LBTS to advance their simulation time by processing future events. But once the structure has changed and before the next structure change happens, no matter where the earliest events exist (on the circle or not), the minimal simulation time (or exactly LBTS) of all models in the simulation system, behaving like a static-structure system, will continue to advance in order to process the earliest events, so the variable structure system adopting our time management algorithm can avoid the deadlock naturally.

5 Dynamic load balancing of variable structure simulation systems

5.1 Dynamic load balancing mechanism

Two mechanisms are proposed to achieve high efficient load balancing (as shown in Fig. 5): an efficient mechanism for load migration and a mechanism that supports dynamic allocation of computing resources. The latter one is mostly neglected by researchers, but it can achieve high utility of computing resources and substantially increase the performance of simulation systems with obvious structure changes, as dynamic structure (changes of constituent models, interaction structure and model ports) can cause changes of the communication or/and computation load on computing facilities, and allocating fewer cores may greatly decrease the synchronization cost when there exist fewer models, or allocating more cores may increase the speed to simulate more models.

Our basic idea of the first mechanism is to separate the necessary elements to schedule a model from the threads, so that load migration can be easily realized when the model as a whole is migrated. When a model is scheduled, four kinds

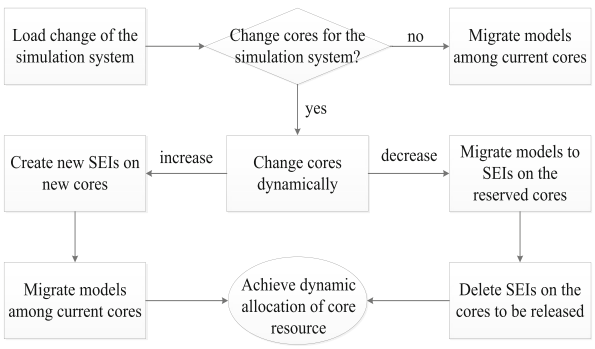

Fig. 5 High efficient load balancing strategy

of elements are tightly related, (1) input events, (2) current simulation time, (3) LBTS, and (4) output destinations of events. The input event queue *inputList*, current simulation time and LBTS of the model are maintained inside the model, so the first three kinds of the related elements can be migrated with models.

The CMIs and connections are shared among all threads (SEIs) in the OSP (*Ivy*). Each SEI maintains a list of CMI references (cf. Figs. 2, 3). The CMIs can be accessed using these references, so that the CMIs can be traversed by a SEI to schedule LBTS computing, processing of safe events and event output in each simulation cycle. The LBTS computation for a model is decided by the CMIs that connect to this model. With the help of shared connections, the information of the sender CMIs can be acquired to compute the LBTS and the events are output by directly inserting event references into the input event queue of the destination CMIs. Thus we can see that the scheduling of migrated models is not affected, except for a short pause of their execution. So the migration of models is simplified as the removal of CMI references from the source SEIs and the addition of CMI references to the destination SEIs.

Operations on the list of model references are controlled by the readers-writer lock *CRL_RW_lock*. Once the next model reference on the list is got, the SEI will release the reader lock of *CRL_RW_lock* to schedule the model, and then load balance management can get the writer lock and modify the list before the SEI returns to get the reader lock of *CRL_RW_lock*. So it is possible that the migration of models can be done without interrupting the normal scheduling of models. During the migrating process, there is a short interval when models are removed from the source SEI and not added to the destination SEI. However, due to the shared connections, messages are normally received without the loss of message and the computation of LBTS for receivers is not affected. Thus the proposed load migration mechanism can be a high efficient way for migration of models without pausing the whole system, and without copying and transferring of the model state.

Our method to achieve dynamic resource allocation is to create new SEI threads or delete old SEI threads on cores. As for decreasing the occupied computing resource, the first step is to migrate models from SEIs to be deleted, and then the redundant SEIs can be safely removed. In order to allocate more computing resource, SEI management need create new SEI threads first and then load balance management would migrate models to the new SEI threads. Such adjusting processes can be executed dynamically, which would raise the efficiency of the simulation execution with little intervention. Our method enables good scalability and can cope effectively with the changing demand for the computing resource.

5.2 Load balancing algorithm

In the multi-core era, the inter-thread communication is extraordinarily fast (the delay is in the nanosecond range [46]), thus the computation speed of CMIs becomes a primary affecting factor and the communication requirement is taken into consideration as a secondary optional factor.

Peschlow et al. [31] and Jiang et al. [33] did some useful work on load balancing in common PDES. Inspired by their work, a metric for fine-grained component models is proposed by considering the processing time of events. During a monitoring interval, T_{Mtr}, the event set $evtSet(m)$ for CMI m is

$$evtSet(m) = \{evt_i | timestamp(evt_i) \in T_Mtr, \\ evt_i \in safeEventList(m)\}. \tag{8}$$

Events in $evtSet(m)$ are processed and $Advance_m$ is the simulation advance of the simulation time of CMI m. The load of CMI m is a measure of the amount of the CPU time it needs to advance its local simulation clock one unit. We consider only homogeneous cores in the multi-core machines, because the metric for heterogeneous cores can be easily deduced like the method in [33].

$$Load_m = \frac{CPU_m}{Advance_m} \\ = \frac{\sum_{evt_i \in evtSet(m)} CPU_m(evt_i)}{Advance_m} \tag{9}$$

The load metric of LPs in [31] can be deduced by setting $CPU_m(evt_i)$ as one to get the number of events a LP has executed within the measurement interval. However, such metric in [31] cannot reflect the real LP load when the processing of individual events needs very different CPU time. Assuming that there exist LPs whose total number is M in the monitoring interval, and M' in the next interval, the mean load of the LPs and the future total load in the next interval are defined as:

$$MEAN = \frac{\sum_{m=1}^{M} Load_m}{M} \tag{10}$$

$$LPsLoad = \sum_{m=1}^{M'} Load_m \tag{11}$$

The load of a new CMI is evaluated as the same as that of a CMI that have the same template SCM, or else is evaluated as $MEAN$. So the future load is the sum of all CMIs in the system as Eq. 11. In order to reduce the time cost of dynamic instantiation of the SCMs, certain number of CMIs can be kept in the system (but not in the simulation) and reused in the simulation after re-initialized. This leaves as the future work.

The algorithm can be configured, according to the following two situations.

5.2.1 Resource-constrained dynamic load balancing

It is highly possible that the available cores are limited, due to the background load of other applications, when the computational infrastructure is not dedicated. Due to fluctuations of the background load, the available cores change over time, indicated in Eq. 12. The load distribution of the simulation also changes because of the variation of the system constituents, reflected in Eq. 11, so the load balancing should also proactively eliminate such possible imbalance.

The ratio of the CPU allocation by processor n to the total CPU allocation is

$$FRAC_n = \frac{EffCPU_n}{EffCPU} = \frac{EffCPU_n}{\sum_{j=1}^{N} EffCPU_j} \tag{12}$$

Thus the load undertaken by processor n is defined as

$$CoreAlloc_n = FRAC_n \times LPsLoad. \tag{13}$$

This is a classical bin-packing problem that many bin packing algorithms can be employed, such as [33–35]. We have implemented one bin-packing algorithm described in [33].

5.2.2 Dynamic load balancing with ample resources

For the multi-core machine with enough available cores, cores can be allocated exclusively to the SEI threads. However, too many cores may lead to costly communication overhead, as the wide distribution of LPs (extreme: each LP allocated to one core exclusively) make the inter-core communication increase greatly and the inter-core communication brings higher latency and overhead than the intra-core communication. More cores also mean it takes less average time to process an event, so the communication delay and overhead become obvious. Thus communication should not be neglected any more.

With the help of connection management, a SEI will schedule its CMIs to pass events to destination CMIs. During this process, the number of events sent between LPs can be naturally acquired to compute Eqs. 16 and 17. These data are used to reduce the communication cost on the basis of computation load balance. The size of events is generally overlooked, because the communication delays are more or less the same using the shared-variable communication.

$$RatioSyncCost = \frac{CommCPU}{EffCPU} \\ = \frac{EffCPU - \sum_{m=1}^{M} CPU_m}{EffCPU} \tag{14}$$

where *RatioSyncCost* indicates the ratio of the communication cost to the whole cost of CPU.

$$minCost = w_{comm} \times \frac{Comm'}{Comm} \times CommCPU + w_{comp} \times \frac{Load'}{Load} \times \frac{CPU}{CPU'} \quad (15)$$

$$Comm_{m \to n} = \frac{LpEvents_{m \to n}}{Advance_m} \quad (16)$$

$$Comm_{m,n} = Comm_{m \to n} + Comm_{n \to m} \quad (17)$$

$$Comm = \sum_{CMI_m \in SEI_i, CMI_n \in SEI_j}^{(SEI_i, SEI_j)} Comm_{m,n} \quad (18)$$

$\{Comm, Load, CPU\}$ and $\{Comm', Load', CPU'\}$ are the states of the simulation system before and after dynamic balancing. $Comm$ and $Comm'$ denote the total inter-thread communication by considering interactions between LPs in different SEI threads. Because of only one single thread for each SEI, little communication cost between LPs in one SEI can be achieved.

RatioSynCost can be set before the simulation to indicate the acceptable threshold of the communication cost. When the threshold is exceeded, the load balancing service considering both computation and communication is started up. A large number of modern computational intelligence methods can be employed to achieve minimal cost in Eq. 15. The values of weights and depend on the simulation application type (computation or communication intensive) and can be assigned using experimental methods or machine learning methods.

6 Experiment and analysis

According to the standard interface defined in [25], the port alteration methods are encapsulated inside the model as the unique entry to modify ports, so naturally variable ports can be supported. Our initial application (Fig. 6) [25], in which a tanker aircraft and several normal planes fly in formation to one remote place has validated our methods on the removal and addition of connections and models. The situation is that: when a normal plane does not receive oil refueling in time, it would land emergently on a nearby airport (exit from the formation fly) and should be removed from the simulation system; before it lands, it will request for reinforce-a new normal plane nearby will join. We do not explain the detailed support of structure changes in this section any more.

The following section demonstrates another application and its experiment results, based on Ivy. Ivy along with the load balancing strategies has been tested to acquire its performance on a common simulation system and a variable-structure simulation system using a typical scenario of the air traffic control system.

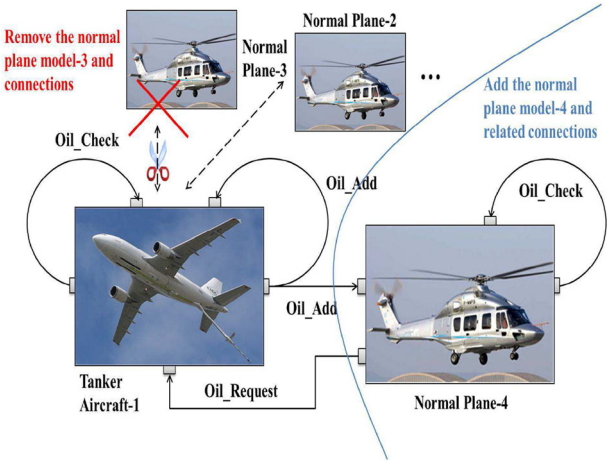

Fig. 6 An example application of the formation flight

6.1 Simulation models

The simulation models are built to simulate the scenario that multiple teams of controllers in a major airport direct hundreds of airplanes to land or take off. Each team is in charge of airplanes in certain sector airspace that can be scanned with radars. The airplanes enter or depart from the controlled airspace when their distances with the control tower are less or more than 80km. The simulation for this scenario includes the change of airplane models and related connections in the simulation system. The simulation system consists of one scheduling center, several teams of controllers and hundreds of airplanes. The landing or taking-off plan of airplanes are made by the scheduling center. Multiple teams of controllers are responsible for coordination between airplanes and the scheduling center. The airplanes are randomly generated and they join the simulation to simulate the entrance of airplanes into the controlled airspace. After landing, the airplane will stay in the airport for different time intervals, take off and ultimately depart from the controlled airspace, which is simulated by the removal of airplane models. The time interval is drawn from the same normal distribution with a standard deviation of 10 %. The constituents and the interaction structure of the whole simulation system can be abstracted as Fig. 7.

6.2 Testing environment and results

All experiments were performed on a high performance multi-core machine with four way 3.07 GHz Intel Xeon CPU X5657, 24G RAM to test our work on the capability of dynamic structure and load balancing among cores. Each CPU contains six cores, so 24 independent threads at most can be created to execute exclusively on different cores. We have created 500 entities including one scheduling center model, four team models and 495 airplanes. The

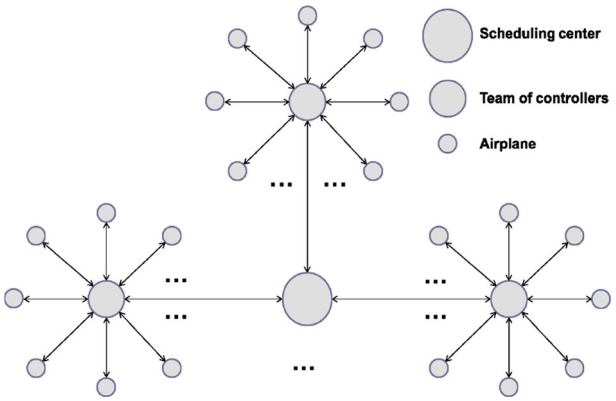

Fig. 7 Model of the air traffic control in an airport

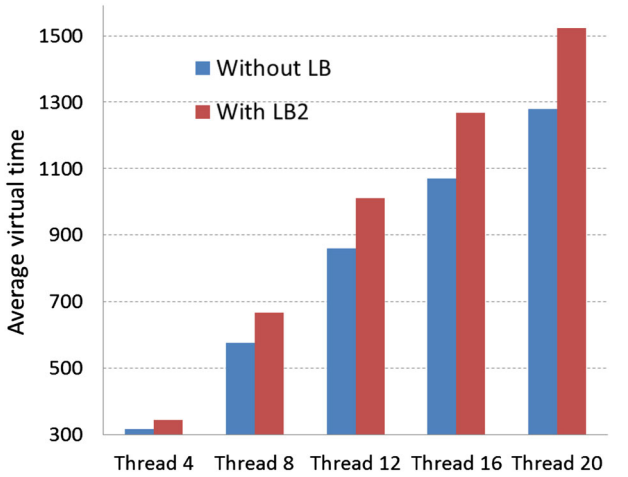

Fig. 8 Experiment results of common simulation

event-processing time for models was randomly generated among [1, 100 ms] and saved to simulate different computation requirements of event processing. Ivy and 500 models executed as an OSP holding 4, 8, 12, 16 and 20 threads respectively on the dedicated multi-core machine. The system ran for the wall-clock time 10 h each. Two kinds of conditions are set, (1) the system constituents of the system is do not changed during the system execution; (2) the system constituents of the system is changed dynamically. Because there are enough available cores, the second method is adopted and the values of weights w_{comm} and w_{comp} are firstly set as 1 and 10 respectively according to our empirical analysis. The cores allocated to the simulation are not changed in this test. The comprehensive experiment covering more applications with the function of dynamic core allocation is the next focus of our work.

6.2.1 Load balancing of the normal simulation system

At the beginning, models are scattered evenly among threads on cores. We set the radius of the controlled airspace as an extraordinarily big value. The airplanes are generated at the beginning and randomly distributed in the controlled airspace, so that during the experiment, the constituents and interaction structure of the simulation system were not changed. The results of the experiments with/without load balancing are shown in Fig. 8.

Comparing with the results of the experiments without load balancing, the average performance by adopting our load balancing method has been improved by 16.05 % (cf. Table 2). With the increase of the thread number, our algorithm can achieve better performance, because load balancing is increasingly important when there exist many collaborative working threads and the efficiency can be extremely low when the load discrepancy is large. Thus we can see that adopting our balancing method can upgrade the performance of the common simulation system.

Table 2 The performance improvement

System configuration	Thd 4	Thd 8	Thd 12	Thd 16	Thd 20	Aver
Performance improvement	0.0868	0.1610	0.1769	0.1861	0.1918	0.1605

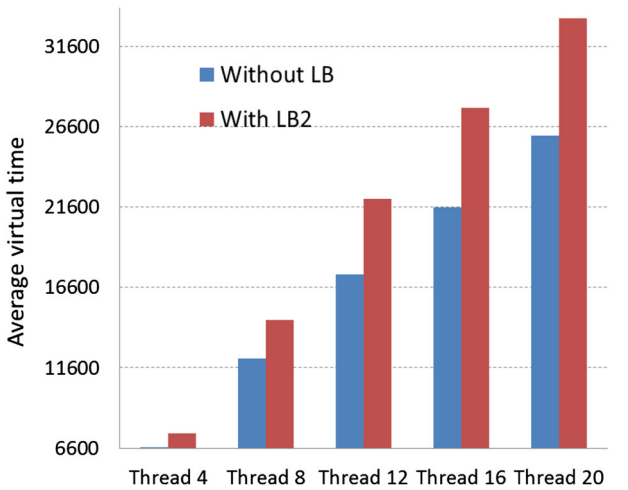

Fig. 9 Experiment results of variable structure simulation

6.2.2 Load balancing of the dynamic-structure simulation system

To enable variable structure, the radius of the controlled airspace was set as a normal value-80km. The airplanes were created to join the simulation and deleted from the simulation dynamically. The airplane would fly in the controlled space and stay in the airport after landing. The comparing experiment results are shown in Fig. 9.

From Table 3, we can infer that our method works well as the number of cores increases. Comparing with the results

Table 3 The performance improvement

System configuration	Thd 4	Thd 8	Thd 12	Thd 16	Thd 20	Aver
Performance improvement	0.1352	0.1974	0.2695	0.2848	0.2795	0.2333

of the experiments without load balancing, the average performance improvement through adopting our load balancing method can be 23.33 %, which is better than the results for the normal simulation system. With the increase of the thread number, our algorithm can achieve better performance. Thus we conclude that adopting our balancing method can upgrade the performance of the variable-structure simulation system.

6.3 Comparison with other implementations

We further compare our simulator with ROSS-MT [16], NTW-MT [17], Adevs [11] based on dynDEVS and DEVSJAVA3.0 based on [7] in terms of runtime model structure, scheduling of models, variable-structure capability, and load balancing in Table 4. As for the runtime model structure, only Adevs [11] and Ivy adopt the flattened one, which can improve efficiency by (1) eliminating unnecessary simulator and coordinator objects, and unnecessary internal synchronization messages, and (2) avoiding unnecessary event routing messages. From the perspective of time management algorithm, ROSS-MT and NTW-MT use optimistic and global induction based time warp mechanism, which does not consider the model structure. Adevs and DEVSJAVA3.0 are conservative and DEVS based, in which the model structure is only used to compute the global LBTS. In constrast, Ivy which is conservative, uses the model structure to extract the inherent parallelism by considering the related "sender" models only and naturally capture the dynamic parallelism caused by the change of interaction relations, to facilitate the parallelization of model executions (see Sects. 4.2 and 4.3). Concerning the scheduling of models, Ivy, ROSS-MT, and NTW-MT adopting the multi-thread paradigm can fully exploit multi-core machines by fast communication (e.g., using shared parameters and events, or pointers between threads). Ivy can further use the fine-grained (higher) parallelism to increase the performance. Moreover, Ivy's time management algorithm is implemented in a fully decentralized way, as SEIs will schedule each CMI to compute its LBTS when the CMI does not have safe events to process. In the case of variable-structure capability, only DEVSJAVA3.0 and Ivy can comprehensively support the change of constituent models, model ports and connections. As for the load balancing ability, ROSS-MT, NTW-MT and Adevs do not support load balancing (at least, we have not

Table 4 Comparison between simulators that support variable structure

Item	ROSS-MT	NTW-MT	Adevs	DEVSJAVA3.0	Ivy
Runtime model structure	Flattened	Flattened	Flattened	Hierarchical	Flattened
Time management algorithm	Optimistic, global induction based, ignoring interaction structure	Optimistic, global induction based, ignoring interaction structure	Conservative, using interaction structure to do global induction	Conservative, using hierarchical structure to do global induction	Conservative, fully distributed, exploiting model structure
Scheduling of models	Parallel by using threaded processes	Parallel by using threaded processes	Parallel by using threaded processes	Parallel by using threaded processes	Highly parallel by using threads and extracted parallelism
Variable-structure ability	No support	No support	Variable models (static ports), connections	Variable models (variable ports), connections	Variable models (variable ports), connections
Load balancing	No support	No support	No support	Part support	Full support

seen the related literature). DEVSJAVA3.0 supports model-continuity for automatic migration to distributed execution, but the method of migrating models is not fully optimized to make full use of multi-core architecture. Also it is not able to evaluate the load of new models and neglect the load of removed models, after the structure change. Comparatively, Ivy can fulfill such gap. Overall, Ivy has advantages on several aspects to perform the simulation of large-scale CVSSs efficiently. More detailed comparisons of experimental results between these simulators are leaved as our next step work.

7 Conclusions and remarks

In this paper, an advanced parallel simulator with load balancing strategies is proposed to support large-scale variable structure simulation. To substantially improve the capacity of simulating large-scale CVSSs, four aspects of contributions are made:

- *Support flexible structure changes of the simulation system* A natural and effective method for building modular models of the CVSSs and a corresponding lock-based concurrent execution approach are proposed, so that safe, flexible and dynamic structure changes of the coupled model, with little intervention to the simulation execution, are achieved. The changes of model ports, connections and composition are comprehensively supported. Readers-writer locks are employed to guarantee the safe concurrency among all operations related to distributed structure changes and trajectory simulations.
- *Exploit the parallelism to a large extent between simulation models* We propose a connection-based time-management algorithm that can extract the inherent parallelism by considering the related "sender" models only and naturally capture the dynamic parallelism caused by the interaction structure change, to facilitate the parallel execution of models. Then the algorithm's ability to correctly synchronize all models is proofed. Using the strong induction, we further proof that the algorithm enables deadlock-free scheduling of models.
- *Take full advantage of multi-core machines* The multi-threaded paradigm is adopted to substantially utilize abundant computing cores and low communication latency among cores, in order to efficiently schedule simulation models in fine-grained parallel.
- *Guarantee the load balance among tens or hundreds of cores* We propose an efficient dynamic load balancing method, which can migrate models among cores with very low cost (only migrating model references) and change the set of cores utilized by the simulation dynamically on demand, to address the load imbalance problems of variable-structure simulation. Using the proposed method, the unrelated SEIs are not interrupted to schedule their models. Structure changes are considered in the load balancing metrics, e.g. evaluating loads of new models, to get the exact the computation/communication load of variable structure simulation.

Based on our developed simulator Ivy and the load balancing strategies, an application example is given. The simulation results show that our methods (flexible structure-change mechanism, dynamic-parallelism extraction, fine-grained parallelization on multi-cores and efficient load balancing strategies) can greatly improve the performance. More application examples need to be implemented to help identify and eliminate the bottlenecks, in order to gain higher performance.

VSP can also bring another important capability - openness for users to adjust scenarios during simulation executions. Traditionally, scenarios predetermine the structure of simulation systems. Some skilled practitioners code conditional structure changes into models, but this makes for a less elegant and coherent model design, and makes the run-time human-machine interaction to change model structure difficult to realize. To change the model structure, according to intermediate results, is critical for simulations of CVSSs. Our work can make structure changes convenient and efficient to realize. We can imagine the situation that users change the parameters and structure of the simulation system when necessary, to make the simulation more powerful.

Prospective applications can be the design simulation of variable structure computers [47,48], which utilize the same hardware in a variety of special purpose structures to achieve performance and economic gains, and internetware [49], which is constructed by a set of autonomic software entities distributed over the Internet and a set of connectors enabling the collaboration among these entities. Our method can also be extended to support multi-resolution simulation and online simulation, such as symbiotic simulation [50], dynamic data-driven simulation [51], cyber-physical simulation [52], to change the structure of simulation models dynamically. For example, equipped with proper algorithms steering adaptive resolution and consistency maintenance, our work can be adapted to support dynamic switching among 3D models with different levels of details according to metrics such as object importance and viewpoint, in the area of dynamic data driven animation [53].

Acknowledgments This work is financially supported by National Key Lab in Intelligent Manufacturing System Technology of Complex Product and the National 863 Plan (2015AA042101), China.

References

1. Holland, J.H.: Studying complex adaptive systems. J. Syst. Sci. Complex. **19**(1), 1–8 (2006)
2. Di Marzo Serugendo, G., Gleizes, M.P., Karageorgos, A.: Self-organization in multi-agent systems. Knowl. Eng. Rev. **20**(02), 165–189 (2005)
3. Anderson, P.: Perspective: complexity theory and organization science. Organ. Sci. **10**(3), 216–232 (1999)
4. Uhrmacher, A.M.: Dynamic structures in modeling and simulation: a reflective approach. ACM Trans. Model. Comput. Simul. (TOMACS) **11**(2), 206–232 (2001)
5. Zeigler, B.P., Praehofer, H.: Systems theory challenges in the simulation of variable structure and intelligent systems. In: Computer Aided Systems Theory EUROCAST'89, pp. 41–51 (1990)
6. Barros, F.J.: Modeling formalisms for dynamic structure systems. ACM Trans. Model. Comput. Simul. (TOMACS) **7**(4), 501–515 (1997)
7. Hu, X., Zeigler, B.P., Mittal, S.: Variable structure in DEVS component-based modeling and simulation. Simulation. **81**(2), 91–102 (2005)
8. Uhrmacher, A.M., Himmelspach, J., Rohl, M., et al.: Introducing variable ports and multi-couplings for cell biological modeling in DEVS. In: Proceedings of the Winter Simulation Conference, pp. 832–840 (2006)
9. Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press, New York (2000)
10. Zacharewicz, G., Hamri, M.E.A., Frydman, C., et al.: A generalized discrete event system (g-DEVS) flattened simulation structure: application to high-level architecture (HLA) compliant simulation of workflow. Simulation **86**(3), 181–197 (2010)
11. Muzy, A., Nutaro, J.J.: Algorithms for efficient implementations of the DEVS and DSDEVS abstract simulators. In: 1st Open International Conference on Modeling and Simulation (OICMS), pp. 273–279 (2005)
12. Robson, E., Boukerche, A.: Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems. J. Parallel Distrib. Comput. **71**(1), 40–52 (2011)
13. Gan, B.P., Low, Y.H., Jain, S., et al.: Load balancing for conservative simulation on shared memory multiprocessor systems. In: Proceedings of Fourteenth Workshop on Parallel and Distributed Simulation, pp. 139–146 (2000)
14. Biswas, R., Aftosmis, M.J., Kiris, C., et al.: Petascale computing: impact on future NASA missions. In: Bader, D. (ed.) Petascale Computing: Architectures and Algorithms, pp. 29–46. CRC Press, Boca Raton (2007)
15. Habata, S., Yokokawa, M., Kitawaki, S.: The earth simulator system. NEC Res. Dev. **44**(1), 21–26 (2003)
16. Wang, J., Jagtap, D., Abu-Ghazaleh, N., et al.: Parallel discrete event simulation for multi-core systems: analysis and optimization. IEEE Trans. Parallel Distrib. Syst. **25**(6), 1574–1584 (2014)
17. Lin, Z., Tropper, C., Ishlam Patoary, M.N., et al.: NTW-MT: a multi-threaded simulator for reaction diffusion simulations in NEURON. In: Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation, pp. 157–167 (2015)
18. Bauer, P., Lindn, J., Engblom, S., et al.: Efficient inter-process synchronization for parallel discrete event simulation on multicores. In: Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation, pp. 183–194 (2015)
19. Tang, W., Yao, Y., Zhu, F.: A hierarchical parallel discrete event simulation kernel for multicore platform. Clust. Comput. **16**(3), 379–387 (2013)
20. Bergero, F., Kofman, E., Cellier, F.: A novel parallelization technique for DEVS simulation of continuous and hybrid systems. Simulation **89**(6), 663–683 (2012)
21. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. Wiley, New York (2000)
22. Himmelspach, J., Uhrmacher, A.M.: Processing dynamic PDEVS models. In: The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pp. 329–336 (2004)
23. Muzy, A., Zeigler, B.P.: Specification of dynamic structure discrete event systems using single point encapsulated control functions. Int. J. Model. Simul. Sci. Comput. **5**(03), 1450012 (2014)
24. Barros, F.J.: On the representation of dynamic topologies: the case for centralized and modular approaches. In: Proceedings of the Symposium on Theory of Modeling and Simulation-DEVS Integrative, p. 40 (2014)
25. Yang, C., Li, B.H., Chai, X., et al.: Ivy: a parallel simulator for variable structure systems under multi-core environments. Int. J. Serv. Comput. Oriented Manuf. **1**(2), 103–123 (2013)
26. Miller, R.J.: Optimistic Parallel Discrete Event Simulation on a Beowulf Cluster of Multi-core Machines. University of Cincinnati, Cincinnati (2010)
27. Vitali, R., Pellegrini, A., Quaglia, F.: Load sharing for optimistic parallel simulations on multi core machines. ACM SIGMETRICS Perform. Eval. Rev. **40**(3), 2–11 (2012)
28. Chen, L., Lu, Y., Yao, Y., et al.: A well-balanced time warp system on multi-core environments. In: Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation, pp. 1–9 (2011)
29. Jafer, S., Liu, Q., Wainer, G.: Synchronization methods in parallel and distributed discrete-event simulation. Simul. Model. Pract. Theory **30**, 54–73 (2013)
30. Peng, Y., Cai, Y., Zhong, R.H., et al.: Parallel framework for HLA federate oriented to simulation component on multicore platform. Ruanjian Xuebao/J. Softw. **23**(8), 2188–2206 (2012)
31. Peschlow, P., Honecker, T., Martini, P.: A flexible dynamic partitioning algorithm for optimistic distributed simulation. In: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, pp. 219–228 (2007)
32. Glazer, D.W., Tropper, C.: On process migration and load balancing in time warp. IEEE Trans. Parallel Distrib. Syst. **4**(3), 318–327 (1993)
33. Jiang, M.R., Shieh, S.P., Liu, C.L.: Dynamic load balancing in parallel simulation using time warp mechanism. In: International Conference on Parallel and Distributed Systems, pp. 222–227 (1994)
34. Coffman, E.G., Elphick, M., Shoshani, A.: System deadlocks. ACM Comput. Surv. (CSUR) **3**(2), 67–78 (1971)
35. Lewis, R.: A general-purpose hill-climbing method for order independent minimum grouping problems: a case study in graph colouring and bin packing. Comput. Oper. Res. **36**(7), 2295–2310 (2009)
36. Som, T.K., Sargent, R.G.: Model structure and load balancing in optimistic parallel discrete event simulation. In: Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation, pp. 147–154 (2000)
37. D'Angelo, G., Bracuto, M.: Distributed simulation of large-scale and detailed models. Int. J. Simul. Process Model. **5**(2), 120–131 (2009)
38. Carothers, C.D., Fujimoto, R.M.: Efficient execution of time warp programs on heterogeneous, NOW platforms. IEEE Trans. Parallel Distrib. Syst. **11**(3), 299–317 (2000)
39. Yang, C., Li, B.H., Chai, X., et al.: An efficient dynamic load balancing method for simulation of variable structure systems.

In: 2013 8th EUROSIM Congress on Modelling and Simulation (EUROSIM), pp. 525–531 (2013)
40. Sun, Y., Hu, X.: Performance measurement of dynamic structure DEVS for large-scale cellular space models. Simulation **85**(5), 335–351 (2009)
41. Uhrmacher, A.M.: Variable structure models: autonomy and control answers from two different modeling approaches. In: Proceedings of AI, Simulation, and Planning in High Autonomy Systems, pp. 133–139 (1993)
42. Uhrmacher, A.M., Ewald, R., John, M., et al.: Combining micro and macro-modeling in devs for computational biology. In: Proceedings of the 39th Conference on Winter Simulation: 40 years! The Best is Yet to Come, pp. 871–880 (2007)
43. Mittal, S.: Emergence in stigmergic and complex adaptive systems: a formal discrete event systems perspective. Cogn. Syst. Res. **21**, 22–39 (2013)
44. Rajaei, H., Ayani, R., Thorelli, L.E.: The local time warp approach to parallel simulation. ACM SIGSIM Simul. Dig. **23**(1), 119–126 (1993)
45. Rao, D.M., Thondugulam, N.V., Radhakrishnan, R., et al.: Unsynchronized parallel discrete event simulation. In: Proceedings of the 30th Conference on Winter Simulation, pp. 1563–1570 (1998)
46. Steinman, J., Parks, J.: A proposed open system architecture for modeling and simulation (OSAMS). In: SISO Simulation Interoperability Workshop. Orlando, FL (2007)
47. Estrin, G.: Organization of computer systems: the fixed plus variable structure computer. In: Western Joint IRE-AIEE-ACM Computer Conference, pp. 33–40 (1960)
48. Bobda, C.: Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications. Springer, Dordrecht (2007)
49. Yang, F., Lü, J., Mei, H.: Technical framework for internetware: an architecture centric approach. Sci. China Ser. F **51**(6), 610–622 (2008)
50. Aydt, H., Turner, S.J., Cai, T.W., et al.: Symbiotic simulation systems: an extended definition motivated by symbiosis in biology. In: IEEE 22nd Workshop on Principles of Advanced and Distributed Simulation, pp. 109–116 (2008)
51. Darema, F.: Dynamic data driven applications systems: a new paradigm for application simulations and measurements. Comput. Sci. ICCS **2004**, 662–669 (2004)
52. Kim, J.E., Mosse, D.: Generic framework for design, modeling and simulation of cyber physical systems. ACM SIGBED Rev. **5**(1), 1 (2008)
53. Liu, H., He, F., Cai, X., et al.: Performance-based control interfaces using mixture of factor analyzers. Vis. Comput. **27**(6–8), 595–603 (2011)

Chen Yang received his B.E. Degree and Ph.D. degree at School of Automatic Science and Electrical Engineering & School of Advance Engineering, Beihang University (BUAA). His doctoral research focused on cloud based high performance simulation and parallel discrete event simulation of complex variable-structure systems on multi-core machines. His current research interests include advanced distributed simulation, cloud computing/simulation/manufacturing, Internet of Things, Big Data, complex system modeling, etc.

Peng Chi received his doctor degree in Condensed Matter Physics from Nankai University, Tianjin, China in 2012. His Doctor's thesis is a simulation study of polymers which aims to study the physical properties of block copolymers and polyelectrolytes, by high performance computing technologies and Monte Carlo method. He is currently an engineer at Beijing Simulation Center. His current interests focus on complex system modeling language and advanced parallel simulation.

Xiao Song is an Associate Professor of Automation School, Beihang University (BUAA), Beijing, China. He was a visiting researcher at the School of Computing, National University of Singapore between 2013 and 2014. He has been a member of ASIASIM (the federation of Asia simulation societies) council meeting since 2012 and performs the role of editor-in-chief of ASIASIM newsletter since 2015. His research interests include system modeling and simulation, big data and cloud computing.

Ting Yu Lin was born in 1984. He received his B.S. degree and Ph.D. in the School of Automatic Science and Electrical Engineering, Beihang University, Beijing, China. He is currently an engineer in Beijing Simulation Center. His research interests include multi-disciplinary virtual prototype, cloud simulation and cloud manufacturing.

Bo Hu Li received his B.E. Degree in computer science and technology from Tsinghua University, China, in 1961. He was a visiting scholar majored in digital simulation in University of Michigan and University of California, Los Angeles, from 1980 to 1982. He is a Professor at School of Automatic Science and Electrical Engineering, Beihang University, and Chinese Academy of Engineering, and the chief editor of "Int. J. Modeling, Simulation, and Scientific Computing". His research interests include multi-disciplinary virtual prototype of complex products, intelligent distributed simulation and cloud manufacturing.

Xudong Chai received his B.E. Degree from Nanjing University of Aeronautics & Astronautics, and his M.E. and Ph.D. degrees in Beihang University. He was a postdoctoral scholar majored in virtual prototyping in National CIMS Center of Tsinghua University from 1999 to 2001. Currently, he is the vice director of Beijing Simulation Center, as well as member of the Council of Chinese System Simulation Association. His research interests include virtual prototyping of complex products, cloud simulation/manufacturing, high-performance simulation and integrated platform.