IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 5, 2021, accepted March 18, 2021, date of publication April 15, 2021, date of current version May 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3073618

Hierarchical Aggregation/Disaggregation for Adaptive Abstraction-Level Conversion in Digital Twin-Based Smart Semiconductor Manufacturing

MOON GI SEOK[1], WENTONG CAI[1], (Member, IEEE), AND DAEJIN PARK[2]
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798
[2]School of Electronics Engineering, Kyungpook National University, Daegu 41566, Republic of Korea

Corresponding author: Daejin Park (boltanut@knu.ac.kr)

This work was supported in part by the A*STAR Cyber-Physical Production System (CPPS)—Towards Contextual and Intelligent Response Research Program through the RIE2020 IAF-PP under Grant A19C1a0018, in part by the Model Factory@SIMTech, and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) by the Ministry of Science and ICT under Grant NRF2019R1A2C2005099 and Grant NRF2018R1A6A1A03025109.

ABSTRACT In smart manufacturing, engineers typically analyze unexpected real-time problems using digitally cloned discrete-event (DE) models for wafer fabrication. To achieve a faster response to problems, it is essential to increase the speed of DE simulations because making optimal decisions for addressing the issues requires repeated simulations. This paper presents a hierarchical aggregation/disaggregation (A/D) method that substitutes complex event-driven operations with two-layered abstracted models—single-group mean-delay models (SMDMs) and multi-group MDMs (MMDMs)—to gain simulation speedup. The SMDM dynamically abstracts a DE machine group's behaviors into observed mean-delay constants when the group converges into a steady state. The MMDM fast-forwards the input lots by bypassing the chained processing steps in multiple steady-state groups until it schedules the lots for delivery to subsequent unsteady groups after corresponding multi-step mean delays. The key component, the abstraction-level converter (ALC), has the roles of MMDM allocation, deallocation, extension, splitting, and controls the flow of each group's input lot by deciding the destination DE model, SMDM, and MMDMs. To maximize the reuse of previously computed multi-step delays for the dynamically changing MMDMs, we propose an efficient method to manage the delays using two-level caches. Each steady-state group's ALC performs statistical testing to detect the lot-arrival change to reactivate the DE model. However, fast-forwarding (FF) results in incorrect test results of the bypassed group's ALCs due to the missed observations of the bypassed lots. Thus, we propose a method for test-sample reinitialization that considers the bypassing. Moreover, since a bypassed group's unexpected divergence can change the multi-step delays of previously scheduled events, a method for examination of FF history is designed to trace the highly influenced events. This proposed method has been applied in various case studies, and it has achieved speedups of up to about 5.9 times, with 2.5 to 8.3% degradation in accuracy.

INDEX TERMS Abstraction-level conversion, aggregation/disaggregation, wafer fabrication, discrete-event modeling, smart manufacturing.

I. INTRODUCTION

Semiconductor manufacturers have applied industry 4.0-based smart manufacturing concepts to their wafer-fabrication plants (commonly called fabs) for efficient manufacturing management [1]–[3]. The smart fab concept is an approach to integrate manufacturing execution systems (MESs) with monitoring and controlling facilities (MCFs). They support automated data collection and instantaneous response to the MESs in real time based on anaysis of the collected data.

The MCFs maintain their fab models with high accuracy by calibrating parameters based on the collected data. The online parameter calibration results in reliable performance expectations in terms of throughput, cycle times, etc. When an unexpected situation occurs (i.e., machine breakdowns and any

The associate editor coordinating the review of this manuscript and approving it for publication was Sunith Bandaru.

production requirement change caused by a new demand), the MCFs attempt to find an optimal solution derived through simulation-based optimization (SBO) considering various decision candidates dispatching rules, lot-release rates, and other operating parameters [4]–[8]. As a result, the MCFs are effectively able to control the MESs based on the optimal solution, though it requires a large simulation time cost. To reduce the response time of the MCFs' feedback controls, accelerated simulation of fab models under multiple design candidates is essential.

Discrete-event models (DEMs) and their simulation methodologies have been widely used to describe large-scale fab systems' complex functions [13]–[16]. The DEM of a fab is designed to reflect the fab's specific details to reach the desired accuracy and examine the decision parameters. The details include complicated manufacturing processing steps, detailed routings, dispatching rules, tool specifications, etc. A DEM consists of composable subcomponents, which operate in an event-driven manner and exchange events to influence others. The composability leads to efficient model maintenance by revising only relevant subcomponents according to the changes in the production scenario or MES. However, the DEM simulation requires a large amount of execution time due to iterative event processing. The shortcoming hinders enough traverse of decision candidates and results in a slow control response to MES. Thus, the speedup of the DEM simulation of a target fab is mandatory to consider a sufficiently large number of design candidates

Several techniques to speed up the DEM simulation have been proposed. The methods are classified into parallel DEM simulation, queueing modeling, and surrogate modeling. The parallel simulation using multiple cores can be a speedup solution for a single simulation instance [17], [18]. However, when traversing the decision candidate space in SBO or replicating multiple simulation instances to meet statistical confidence, multiple simulation instances can run independently. When executing the independent instances in parallel, the parallel simulation approach can be inefficient due to the synchronization overhead in terms of simulation clock and data among models running on different CPUs [19], [20].

The existing acceleration approaches, such as queuing and surrogate modelings, are known as alternative solutions that abstract the production dynamics of target fab systems into analytic equilibrium equations to achieve reasonable accuracy. The queueing modeling constructs a queueing network, consisting of queuing nodes of each machine group (also called workstation); the machine group is the set of machines performing the same or similar operations and sharing a queue [9]–[11]. Each queueing node is modeled using derived arrival and service processes to predict the average waiting time in steady states. However, as J. George discussed in [12], a particular assumption of queuing theory, which is a sequence of inter-arrival times and services times are independent and identically distributed (i.i.d), are not acceptable in a common scenario. The scenario includes (1) dynamic changes of wafer-lot release or dispatching schedules (that can unexpectedly cause an arrival-pattern variation of high-mix lots) and (2) machine groups with batching and cascading machines. Moreover, most existing models are restricted to the FIFO dispatching rule.

Surrogate modeling is an approach describing the analytic closed-form solutions for each machine group with empirical approximations [21]–[24]. Surrogate model evolve in various forms: a constant delay, probability distributions, or exponential/quantile functions. Surrogate models' parameter values are trained on DEM simulation results of multiple scenarios for a specific design of experiments (DOE). Similar to queueing models, surrogate models are limited to certain production scenarios with consistent product-mix, process flow, and volume due to the static and analytic representation. In the smart fab case, if new operational parameter values vary outside the bound of the DoE coverages, caused by a production demand for new products, a deadline/priority change, or unexpected machine downs, surrogate models cannot guarantee satisfactory results.

To resolve these static abstraction problems for various complex scenarios, we previously presented the conducted studies to adjust machine groups' abstraction level in runtime adaptively [25], [26]. The key components introduced in the studies, abstraction-level converters (ALCs), dynamically choose their group's active models from between DEMs and mean-delay models (MDMs), depending on whether their groups run in a steady or transient state. An MDM is an aggregated model of corresponding DEMs formed by abstracting the DEM's subcomponents' complex behaviors into observed mean delays. The adaptive abstraction conversion was applied and evaluated in various scenarios with dynamically changing lot release and multiple dispatching rules and showed meaningful speedups and accuracy. The dynamic abstraction reduces MCFs' engineering costs for preparing the intermediate queueing and surrogate models in advance. Moreover, it is possible to analyze various real-time circumstances by adjusting related DEM parameters for the MES controls, not limited to a few intermediate models' parameters and their DoE bounds.

The previously proposed method abstracts each steady-state discrete-event (DE) group model into a single-group mean-delay model (SMDM), which abstracts the group's complex DE operation into a mean-delay constant. This paper proposes a new abstraction layer for better speedup. The abstraction layer consists of multiple multi-group MDM (MMDM) that manage expected multi-step delays of input lots in consecutive steady-state groups, as shown in Fig. 1. Based on the multi-step delays, the MMDM fast-forwards input lots through bypassing intermediate steady-state machine groups until the subsequent machine group is in an unsteady state. After fast-forwarding (FF), the lots are scheduled to wait in an event list of the simulation engine before being delivered after the multi-step delay. The simulation engine processes stored events in the event list one by one in chronological order, which guarantees the causal relationships of the DEM's multiple input events

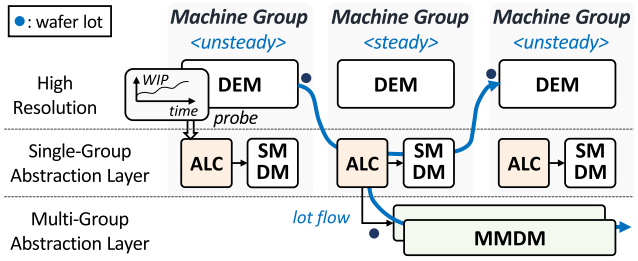


FIGURE 1. Conceptual framework of the proposed hierarchical A/D approach.

from different sources. However, the event-scheduling is the main overhead in the DEM simulation. The FF technique reduces the number of event scheduling.

In previous studies, ALCs' primary roles were switching active models of machine groups according to the group's steady-state convergence (or divergence). For the FF, we extend the previous ALC to support the A/D processes by communicating with neighboring ALCs (whose groups handle previous and subsequent processing steps) or with MMDMs to allocate/deallocate and extend/split the MMDMs. The MMDM manages the lot's multi-step mean delays by a defined lot type. Considering the MMDM's dynamic aggregation through absorbing other MMDMs, we proposed a delay-management method using two-level cache tables.

However, FF can prevent the ALC's proper behavior of the lot-arrival-change detection, which is one of the conditions to reactive the group's active model from SMDM to DEM. For the ALC's correct detection, we proposed a method to reinitialize the arrival-change-testing samples according to the bypassing capability. Moreover, the FF assumes that a forwarded lot's bypassed groups remain in steady states before being delivered to the next unsteady group. Still, a steady-state group can diverge at any point in runtime. To handle the problem, we proposed an event-rescheduling method to cancel the previously forwarded events and make new forwarded events based on the new steady-state situations, using FF history tables (managed by the SMDM and MMDM).

Overall, the contributions of our proposed method can be summarized as follows.

- We propose a new abstraction layer using MMDMs for the FF of lots across successive steady-state groups. The new layer supporting the group bypassing reduces the number of events scheduled at the wafer-lot exchanges between steady-state groups, which results in more speed than the single-group A/D approach (that employs only SMDMs) does.
- To reduce the redundant calculation of multi-step delays under the dynamic MMDM extension, we proposed an efficient delay-management method using two-level caches.
- We resolved the problems caused by bypassing some intermediate groups. The problems were (1) the lot-arrival changes when detecting a divergence condition and (2) the influence of bypassed groups' divergences on previously forwarded lots. To address these problems, we proposed a sample reinitialization technique and FF-history management methods.

The rest of the paper is organized as follows. Section II describes the background of previous single-group abstraction and the proposed hierarchical A/D framework. Section III details the multi-step delay management method for the FF using two-level caches. Section IV shows the detailed procedures to resolved the problems caused by the FF. Section V presents the experimental results of the proposed approach, and Section VI concludes the paper.

II. HIERARCHICAL AGGREGATION AND DISAGGREGATION APPROACH

In this section, we introduce an overview of the previously proposed A/D method for each individual machine group. Then, we propose an overall framework and the operational mechanism of hierarchical A/D for the FF.

A. BACKGROUND OF THE ADAPTIVE A/D APPROACH FOR INDIVIDUAL MACHINE GROUPS

In the original A/D method, the ALC dynamically switches each individual machine group's active model between the DEM and its corresponding SMDM. The SMDM is an aggregated model of its DEM, which abstracts the complicated event-driven operations and event exchanges of a steady-state DEM's subcomponents into observed mean delays.

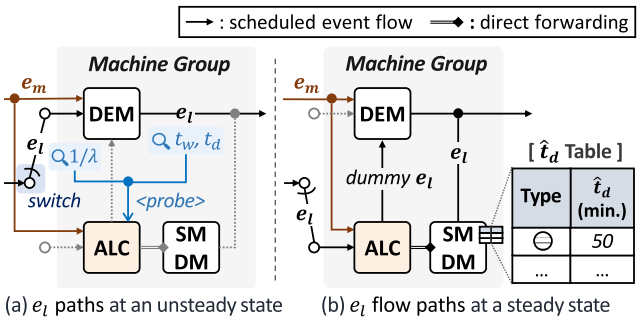


FIGURE 2. A machine group's event flow-path change depending on a steady state.

A machine group's active model switches between the DEM and SMDM when the group's ALC detects a convergence into or divergence from the steady state. To detect a steady-state convergence, the ALC monitors the queuing parameters of processed **wafer-lot events** (e_ℓ), leaving the ALC's machine group (see Fig. 2(a)). The observed parameters are the **inter-arrival time** ($1/\lambda$), **waiting time in a queue** (t_w), and **time spent before reaching the next machine group** (t_d). The $1/\lambda$ is the arrival-time distance between **lots of the same type**.

After constructing two consecutive samples of $1/\lambda$ and t_w, if both two samples of $1/\lambda$ and t_w are statistically indistinguishable, then the ALC confirms that the machine group

converges into a steady state in terms of work-in-progress (WIP) level, according to Little's law. The statistical indistinguishability is determined by measuring the difference between the two samples' cumulative distribution functions based on the Kolmogorov–Smirnov test [30], [31].

Depending on each lot's product, due date, or processing recipe, t_w or t_d can vary considerably. We defined the parameter composition that affects the priority decision among waiting lots and processing time by lot type. The $1/\lambda$, t_w, and t_d samples consist of multiple types of lots' $1/\lambda$, t_w, and t_d observations. In a steady-state machine group, the average number of WIP is $\sum_l \hat{\lambda}^l \cdot \hat{t}_d^l$, where l is a **lot type**, $\hat{\lambda}^l$ is the mean $1/\lambda$ between the l-type lots, and $\hat{t}_d^l$ is the mean delay of the l-type lot.

When a group consists of batch machines or employs a lot of non-batch equipment, a sharp increase of observations occurs within a short-time range. In this case, the building of a sample by observation number can lead to a hasty decision, leading in turn to many imprudent DEM-SMDM switches. For a reliable decision, the ALC manages multiple local $1/\lambda$ groups after partitioning individual $1/\lambda$ values based on a defined time window. For example, if the window is 30 minutes and an e_ℓ's $1/\lambda$ triggers a new local-group generation at a simulation time of 1 : 00 PM, the following $1/\lambda$ observations occurring by 1 : 30 PM belong to the same local $1/\lambda$ group. Each $1/\lambda$, t_w, and t_w sample consists of the mean $1/\lambda$, t_w, and t_w of a defined number of local parameter groups.

At the convergence detection, the ALC derives a delay table that contains the lot types and their corresponding mean delays (see Fig. 2(b)), and then passes the table over to the SMDM. The ALC initializes the $1/\lambda$ sample or other data to detect an input-arrival change or to generate dummy lots (for a workload consistency between the DEM and SMDM) at the DEM reactivation. Lastly, as shown in Fig. 2(b), the ALC requests the flow-path changes of lot events (e_ℓ) to the simulation engine.

In a steady state, the ALC monitors any divergence condition and forwards input lot events to its SMDM. The monitoring divergence conditions are classified as follows.

1) Arrival of an unobserved lot type (based on the steady-state $1/\lambda$ sample),
2) A statistical $1/\lambda$ deviation of input lots from the steady-state $1/\lambda$ sample, and
3) Any event arrival that influences processing time (e.g., machine breakdown, preventive maintenance, etc.).

During the simulation, the event exchanges between model components are performed in two different ways: scheduling-based delivery and direct forwarding.

The scheduling-based event delivery enables the receiver to collect input events in the order of **the event time** (t_e), which allows the receiver to preserve a causal relationship between multiple types of events from numerous sources. Before the delivery, events are stored in the simulation engine's event list chronologically. Direct forwarding means an e_ℓ source directly sends events to their destination model, without the engine's intervention; thus, the receiver instantly handles the input events that arrive in the committing order. An example of scheduling-based delivery is e_ℓ delivery between machine groups, and an example of direct forwarding is e_ℓ delivery from the ALC to the SMDM.

B. OVERVIEW OF PROPOSED HIERARCHICAL A/D APPROACH

We extend a previous method by introducing a new abstraction layer to support the lot's FF across consecutive steady-state groups based on the accumulated mean delays. FF can reduce the number of lots visiting steady-state groups, which alleviates the e_ℓ scheduling overhead of the simulation engine. The overall framework of the proposed approach is illustrated in Fig. 3.

When detecting a steady-state convergence, the ALC performs the primary operations to reactivate the SMDM and to prepare for the further $1/\lambda$ divergence test, as described in Sec. II-A. Then, the ALC attempts to allocate new or extend existing MMDMs based on neighboring groups' steadiness; the neighboring groups consist of previous groups that sent lot events to the current group and the next groups that will handle the subsequent processing steps of lot events. The lot's production route (***r***) is a sequence of machine groups for a specific chain of processing steps. In this paper, we simply represent each processing step as an integer according to the execution order.

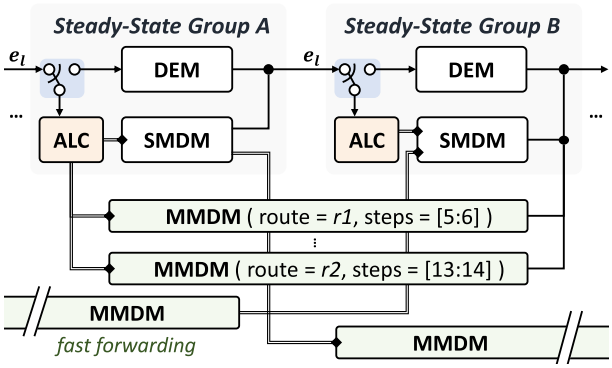
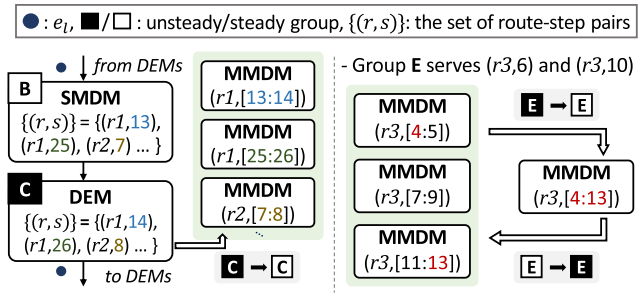


FIGURE 3. The overall framework for hierarchical A/D.

The MMDM manages accumulated mean delays for specific processing steps of a particular production route as an aggregated SMDM. The SMDM enables aggregation of different types of machine groups based on the homogeneity and merging capability in terms of the constant delay accumulation. The MMDM asks the corresponding SMDM for the $\hat{t}_d$ of a specific processing step.

If a machine group serves lots whose routes are different, or if it has any reentrant lot flows, then the group's steady-state convergence leads the multiple MMDM generations by the group's ALC, as shown in Fig. 4(a). In the example, groups *B* and *C* are reentrant groups and serve lots whose routing paths are *r1*, *r2*, and so on. A reentrant group's convergence can aggregate multiple MMDMs into a single



(a) A case of multi MMDM generation (b) A case of multiple MMDM A/D

FIGURE 4. Examples of the multiple MMDM generation, aggregation, and splitting.

MMDM, as shown in Fig. 4(b). Similarly, the group's divergence can split an existing MMDM into multiple MMDMs.

Depending on the production scenario, some routes can have a step whose next step is decided stochastically. Sometimes, a lot's next step for a wafer inspection can be skipped (i.e., $Pr(s_{n+1} = j+1 | s_n = j) < 1$, where n is the step order and j is a step among a specific route's successive steps). Alternatively, a lot can return to a previous step due to an incorrect overlay or defect control (i.e., $Pr(s_{n+1} = k | s_n = j) > 0$, where a step k is less than j).

To manage static multi-step delays for input e_ℓs without concerning the stochastic step-to-step transition, we defined a step aggregation rule for a new MMDM allocation below.

Definition 1: If two groups that serve a route's successive steps of j and $k(= j+1)$ converge into steady states, then an MMDM for the two steps cannot be allocated when $Pr(s_{n+1} = k \mid s_n = i) < j$.

When a departing e_ℓ's next step varies stochastically and a subsequent group for a selected next step is in a steady state, the e_ℓ from an SMDM/MMDM is directly forwarded to the subsequent MDM, one of either SMDM or MMDM. When we denote the FF function as δ_f and an MDM's FF as $\delta_f(e_\ell)$, we can describe the mechanism of the FF across multiple MDMs below.

- $\delta_f \circ \delta_f \circ \delta_f \cdots : e_\ell \rightarrow e_\ell$

Until an output e_ℓ of $\delta_f(e_\ell)$ meets a scheduling condition, the e_ℓ traverse across MDMs without being sent to the simulation engine through the δ_f composition. The scheduling condition is defined below.

Definition 2: The e_ℓ is scheduled to be delivered to the receiver after a multi-step mean delay without a further FF under a situations, where

- the subsequent group runs in a steady state,
- the last step is the final step of the e_ℓ's route, and
- the subsequent group's SMDM does not include the e_ℓ's delay information.

During the multiple executions of δ_f, multi-step delays are iteratively accumulated. We denote the **previously accumulated delay up to the current MDM** as $\hat{t}_{pd}$ and the **accumulation of multiple mean delay in the current MMDM** as $\hat{t}_{md}(= \sum \hat{t}_d)$. We assume that the e_ℓ contains $\hat{t}_{pd}$ internally. When a source MDM forwards an e_ℓ to another MDM through the receiver's δ_f, the source should revise the e_ℓ's $\hat{t}_{pd}$ by considering the newly derived $\hat{t}_{md}$. Otherwise, $\hat{t}_{pd}$ is always 0.

When we denote a lot's intermediate processing step as i, departing e_ℓ's $\hat{t}_{pd}$ ($\hat{t}^+_{pd}$, for the further FF) is derived as follows.

$$\hat{t}^+_{pd} = \begin{cases} \hat{t}^-_{pd} + \hat{t}_d, & \text{if source is SMDM or} \\ \hat{t}^-_{pd} + \hat{t}_{md}(= \sum \hat{t}_d) & \text{otherwise.} \end{cases} \quad (1)$$

The $\hat{t}^-_{pd}$ is the input e_ℓ's $\hat{t}_{pd}$. If an input e_ℓ's next group is unsteady, a source MDM schedules the e_ℓ using the derived event time t_e; the t_e is the sum of the **current simulation time** t_c, $\hat{t}^-_{pd}$, and the expected delay in the MDM (that is one of $\hat{t}_d$ (for SMDM) or $\hat{t}_{md}$ (for MMDM)) (i.e., $t_e = t_c + \hat{t}^+_{pd}$). Before scheduling, the MDM should manipulate the output e_ℓ based on the last step so that the destination model can receive the same event regardless of the source types (i.e., DEM, SMDM, and MMDM). The following section deals with the $\hat{t}_{md}$ management to reduce the overhead of the $\hat{t}_d$ accumulation considering the dynamic MMDM extension.

III. MULTI-STEP MEAN-DELAY MANAGEMENT

We previously defined the lot type as a combination of group-specific parameters that consist of a processing recipe (that affects the processing time) and other parameters related to the dispatching rule (that influences the waiting time), as described in Sec. II-A). However, if the target fab changes the dispatching rule and employs multiple dispatching rules for different groups, then the sub-parameters of lot type can vary depending on the time and group's dispatching policy. In the proposed work, we aim at managing the $\hat{t}_{md}$ by type for the MMDM. For that, a comprehensive lot type should be redefined regardless of the specific time or individual group, so we redefine a new lot type as follows.

*Definition 3: The lot type is the tuple of a **lot header** (h) and required **processing step** (s_a) at the **current-group arrival** with the following assumption: If wafer lots' headers are the same, then they share the same production parameter values in each group.*

We denote the **lot type** as (h, s_a). Examples of production parameters are the production route, the chain of processing steps, each step's required processing recipe, user-predefined priority, due date, etc. The header symbolizes the comprehensive static values or dynamic value changes of parameters involved in the dispatching and processing. The type's s_a is utilized to choose the step-dependent parameters, such as recipe. Moreover, the s_a can be used for the priority decision at particular dispatching rules, such as the shortest remaining process time and the least slack time. The lot's dispatching priority may vary according to the change of dispatching rule, but the new type guarantees that the lots whose type is the same have the same priority during the dispatching in each group.

All components in the proposed framework operate based on the newly defined type, even for the ALC's primary

operations: observing the queuing parameters, testing the steady-state convergence (or divergence), etc.

A. THE DELAY MANAGEMENT OF MMDM USING TWO CACHE TABLES

In the proposed approach, the SMDM manages $\hat{t}_d$ for a single step of all group-related routes, and the MMDM supervises $\hat{t}_{md}$ for multiple successive steps of a specific route. We denote the MMDM's **minimum** and **maximum** of the **successive steps** as $\boldsymbol{s_{\min}}$ and $\boldsymbol{s_{\max}}$, respectively. The lots having different headers but following the same route can share the same MMDM for FF.

The MMDM receives an e_ℓ whose s_a is larger than $s_{\min}$ (i.e. $s_{\min} < s_a \leq s_{\max}$) under two conditions: (1) the e_ℓ was previously scheduled before the s_a-handling group converged into a steady state or (2) the s_a has multiple precedent steps because of the route's stochastic step transition. Regarding the second condition, for example, if a step j's subsequent step is one of $j+1$ and $j+2$, and $j+1$'s subsequent step is always $j+2$, then an MMDM can be generated for the merged steps of $j+1$ and $j+2$ (i.e., $[s_{\min} : s_{\max}] = [j+1 : j+2]$) when their groups are in steady states. Then, the MMDM can receive an e_ℓ whose last step was j and s_a is $j+2$, which is larger than $s_{\min}$.

An input e_ℓ's last step in an MMDM can also be less than the MMDM's $s_{\max}$ if an SMDM for an intermediate step, i ($i < s_{\max}$), did not observe the delay for the type of (h, i) during the t_d-sample constitution (utilized for the steady-state convergence detection). In this case, the e_ℓ is scheduled to be delivered to the SMDM's group after an accumulated delay up to the $i-1$. The e_ℓ arrival reactivates the group's DEM because the unobserved lot-type arrival is one of the previously defined DEM activation conditions (see Sec. II-A). Likewise, the relationship between an e_ℓ's required step at the arrival and **last step** $\boldsymbol{s_l}$ and the e_ℓ-handling MMDM can be represented as $\boldsymbol{s_{\min} \leq s_a \leq s_l \leq s_{\max}}$.

The SMDM manages $\hat{t}_d$ by the lot type, so the lots having the same header shares the same $\hat{t}_d$ for a specific processing step in a steady-state group. In a MMDM, the $\hat{t}_d$ accumulation for an input e_ℓ's $\hat{t}_{md}$ proceeds until an SMDM does not have the delay for the type of (h, i). Thus, the lots having the same type have the same s_l and $\hat{t}_{md} (= \sum_{i=s_a}^{s_l} \hat{t}_d(h, i)$, where $\hat{t}_d(h, i)$ is the $\hat{t}_d$ for the i-step processing of the h-header lots).

Depending on the target fab, the number of processing steps in a route can be more than 600. Using the deterministic FF in the MMDM, we propose a delay management method using two-level cache tables defined below to remove the iterative delay computations by reusing the previous calculation results considering adaptive MMDM revision.

Definition 4: The cache tables consist of $cache_1$ and $cache_2$, where

- $cache_1$ is a hash table that maps the lot type h (as a key) to $(s_l, \hat{t}_{md})$; and
- $cache_2$ is the secondary cache that maps h to the list of the record $(s_a, s_l, \hat{t}_{md})$, sorted in the increasing order of s_a.

The lot type can be converted to a unique integer considering the maximum values of h and steps of routes; therefore, $cache_1$ can be built as a hash map that utilizes integer keys to search the $\hat{t}_{md}$. If an e_ℓ's $(s_l, \hat{t}_{md})$ is not found in $cache_1$, then the proposed scheme attempts to derive the $(s_l, \hat{t}_{md})$ using a $(s_a, s_l, \hat{t}_{md})$ record in $cache_2$, whose s_a is close to the e_ℓ's s_a. The closeness is determined by whether s_a and s_a have the same step class id (cid). The step cid is simply defined below.

Definition 5: A step (i)'s cid is $\lfloor (i - s_{\min})/d \rfloor$, where d is a user-defined range of each class.

In the $cache_2$, each h's **record list** $\boldsymbol{L_2}$ has the previously calculated $\hat{t}_{md}$ values for each step class. When we denote a record $(s_a, s_l, \hat{t}_{md})$ as $(s_a^\circ, s_l^\circ, \hat{t}_{md}^\circ)$, depending on the s_a of an input e_ℓ's type, the following relationship can be derived.

Lemma 3.1: Let $s_a^\circ < s_a$ and $s_a \leq s_l^\circ$. Then, e_ℓ's s_l is the same as s_l°.

Proof: The record guarantees that all $\hat{t}(h, i)$, where $s_a^\circ \leq i \leq s_l^\circ$, are observed. The s_l° always satisfies one of two properties: (1) $s_l^\circ = s_{\max}$ or (2) $\hat{t}_d(h, s_l^\circ + 1)$ was not observed, so the e_ℓ's $\hat{t}_{md}$ accumulation cannot proceed over s_l°. Since each $\hat{t}_d(h, j)$, where $s_a^\circ < s_a \leq j \leq s_l^\circ$, is prepared, the accumulation is safely possible up to s_l°. □

From a similar point of view, the following two properties are easily derived.

Property 1: If $s_a^\circ > s_a$, then the record is always helpful because it reduces the inspection range for the accumulation from $[s_a : s_{\max}]$ to $[s_a : s_a^\circ - 1]$. If each $\hat{t}(h, i)$, where $s_a \leq i \leq s_a^\circ - 1$, is observed, then the e_ℓ's s_l and s_l° are the same.

Property 2: If $s_l^\circ < s_a$, then the record is not relevant because e_ℓ's accumulation range starting from s_a does not overlap with the record's ($[s_a^\circ : s_l^\circ]$).

Using the two cache tables, the overall procedures of the MMDM to derive an input e_ℓ's $(s_l, \hat{t}_{md})$ are described in Alg. 1. If $cache_1$ does not include the $(s_l, \hat{t}_{md})$ value for an input e_ℓ, the method starts the node traverse of the h's L_2. If the e_ℓ's s_a and the node's s_a have the same cid value, then the final $(s_l, \hat{t}_{md})$ is derived as in Lines 23-33. If no data matches the e_ℓ's cid, then the proposed scheme attempts to find a record whose s_a° is higher than the e_ℓ's s_a because of the record's usefulness (based on Prop. 1). After checking the record's range intersection (based on Prop. 2) and comparing the inspection distance between the previous and next record, the final record is chosen, as in Lines 11-16. Alternatively, a remaining record whose cid is less than the e_ℓ's cid is chosen if the range is overlapped and the inspection range decreases. If not, the record value is revised as Line 19 for the further accumulation in Lines 32-33. Then, the result is saved to $cache_1$ and $cache_2$ as in Lines 34-35. If a relevant $(s_l, \hat{t}_{md})$ cannot be found in either cache, then the regular accumulation is performed and the results are saved to the two caches.

B. CACHE UPDATE FOR EXTENDED/SPLIT MMDMs

The steady-state convergence of a machine group can lead to two or more MMDMs' aggregation into a single MMDM. For the new extended MMDM, the converging group's ALC attempts to construct the new extended MMDM's $cache_2$

Algorithm 1: The MMDM's $(s_l, \hat{t}_{md})$ Derivation for an Input e_ℓ

```
1  (h, s_a): e_ℓ's type
2  cid, cid⁻, node, s_a⁻, s_l⁻, t̂_d⁻, s_a°, s_l°, t̂_d°: temporary variables
3  if cache_1 has (s_l, t̂_md) of e_ℓ's type then // check cache_1
4      update (s_l, t̂_md) using the row
5  else if cache_2 has L_2 of h then// check cache_2
6      cid ← ⌊(s_a − s_min)/d⌋, node ← first(L_2)
7      while true do
8          (s_a°, s_l°, t̂_md°) ← node, cid° ← ⌊(s_a° − s_min)/d⌋
9          if cid = cid° then
10             break
11         else if cid < cid° then
12             if prev(node) is valid then
13                 (s_a⁻, s_l⁻, t̂_md⁻) ← prev(node)
14                 if s_l⁻ ≥ s_a and (s_a − s_a⁻) < (s_a° − s_a) then
15                     (s_a°, s_l°, t̂_md°) ← (s_a⁻, s_l⁻, t̂_md⁻)
16             break
17         else if node = tail(L_2) then
18             if s_l° < s_a or (s_a − s_a°) > (s_l° − s_a) then
                   // for inspection from s_a to s_max
19                 (s_a°, t̂_md°) ← (s_max° + 1, 0)
20             break
21         else
22             node ← next(node)
23     s_l ← s_l°, t̂_md ← t̂_md° // initialize var.
24     if s_a° > s_a then
25         for i = s_a; i < s_a°; i = i + 1 do
26             if t̂_d(h, i) is unobserved then
27                 s_l ← i's prev. step, t̂_md ← t̂_md − t̂_md°
28                 break
29             else
30                 t̂_md ← t̂_md + t̂_d(h, i)
31     else
32         for i = s_a°; i < s_a; i = i + 1 do
33             t̂_md ← t̂_md − t̂_d(h, i)
34     if cid ≠ cid° then insert (s_a, s_l, t̂_md) to cache_2
35     insert (s_l, t̂_md) to cache_1
36 else // regular accumulation
37     s_l ← s^max, t̂_md ← 0
38     for i ← s_a; i < s_max; i = i + 1 do
39         repeat the procedures in Lines 26-30
40     insert (s_l, t̂_md) to cache_1
41     if s_l ≥ s_a then insert (s_a, s_l, t̂_md) to cache_2
42
43 return (s_l, t̂_md)
```

based on the previous nodes' $cache_1$, considering the new steps of $[s_{\min} : s_{\max}]$. After identifying all headers from the multiple $cache_1$s in the previous MMDMs, the ALC asks each previous MMDM for each h-related representative data of $(s_a^*, s_l^*, \hat{t}_{md}^*, s_{\max}^*)$, where s_a^* is the minimum among the saved s_a values in the MMDM's $cache_1$ for the header h and other sub-parameters also come from the MMDM's $cache_1$ and $s_{\max}$, as shown in Fig. 5(a). Then, a temporary list L^* is

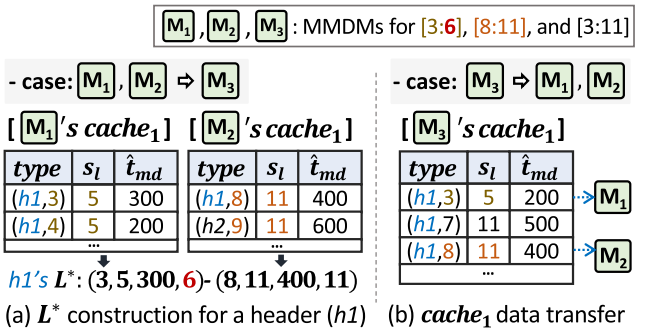


FIGURE 5. Example of the L^* initialization and $cache_1$ data distribution at the MMDM A/D.

constructed using the $(s_a^*, s_l^*, \hat{t}_{md}^*, s_{\max}^*)$ nodes in the order of s_a^*. The new MMDM's $cache_2$ is calculated through the L^* revision.

For the revision, we define a cumulative relationship between the L^*'s nodes below.

Definition 6: For the successive two nodes in the L^*, if the s_l^* and $s_{\max}^*$ of the current node are the same, and all $\hat{t}_d$ of the intermediate steps between the current node's s_l^* and the next node's s_a^* are observed, then two nodes are in the cumulative relationship.

Based on the defined cumulative relationship, we can derive the following property.

Property 3: After the next node in a h-related L^* is updated for the new MMDM's $s_{\max}$ and the current and next nodes are in the cumulative relationship, then the current node's s_l^* is set to the next node's s_l^*. The current node's $\hat{t}_{md}^*$ is updated as the sum of $\hat{t}_{md}^\circ$, $\hat{t}_{md}^+$, and $\sum_{i=s_l^\circ+1}^{s_a^+-1} \hat{t}(h,i)$, where $(s_l^\circ, \hat{t}_{md}^\circ)$ is the current node's $(s_l^*, \hat{t}_{md}^*)$ and $(s_a^+, \hat{t}_{md}^+)$ is the next node's $(s_a^*, \hat{t}_{md}^*)$.

Based on the property, the overall procedures to derive the L_2 for the new MMDM's $cache_2$ is described in Alg. 2. At first, the last node of L^* (for an h) is updated based on a new $s_{\max}$ if the $s_{\max}$ is extended and $\hat{t}_d(h, s_{\max})$ is observed, as in Lines 15-10. The converging group's ALC attempts to check the cumulative relationship of each node (in the L^*) with its previous node in reverse order from the last. If two consecutive nodes are in the cumulative relationship, the previous node's $(s_l^*, \hat{t}_{md}^*)$ is updated according to Prop. 3, as in Lines 15-22. The ALC creates a new L_2 using the nonempty nodes of L^* after checking the cid duplication, then passes the L_2 to the new MMDM.

When a machine group diverges, the group's ALC can deallocate an existing MMDM or reduce the length of an MMDM's merged steps. Sometimes, the ALC can split an MMDM into multiple MMDMs when the diverging group handles numerous steps of the existing MMDM. Before processing the MMDMs, the ALC derives the set of new merged-step ranges ($\{[s_{\min} : s_{\max}]\}$) for the segmented MMDMs. Each merged step range should meet the property of $s_{\max} > s_{\min}$ because the MMDM's minimum step size is 2.

For each record in the existing MMDM's $cache_1$, the ALC transforms the record into $(s_a, s_l, h, \hat{t}_{md})$. Based on the

Algorithm 2: The Derivation Method for a New L_2 of a Specific h

```
s_max: the extended MMDM's s_max
cid, cid*, node, s*_a, s*_l, t̂*_md, s*_max, s+_a, s+_l, t̂+_md: tmp. vars
// revise tail node in L_p
node ← tail(L*), (s*_a, s*_l, t̂*_md, s*_max) ← node
if s*_l = s*_max and s*_max < s_max then
    for i = s*_max + 1; i ≤ s_max; i = i + 1 do
        if t̂(h, i) is unobserved then
            i ← i − 1 break
        else
            t̂*_md ← t̂*_md + t̂(h, i)
    node ← (s*_a, i, t̂*_md, s_max)
while prev(node) is valid do
    (s+_a, s+_l, t̂+_md, s+_max) ← node
    node ← prev(node), (s*_a, s*_l, t̂*_md, s*_max) ← node
    if s*_l = s*_max then
        for i = s*_l + 1; i < s+_a; i = i + 1 do
            if t̂(h, i) is unobserved then
                i ← i − 1 break
            else
                t̂*_md ← t̂*_md + t̂(h, i)
        if i = s+_a then
            i ← s+_l, t̂*_md ← t̂*_md + t̂+_md // Prop. 3
        s*_l ← i
    node ← (s*_a, s*_l, t̂*^{,s*_max}_md)
L_2 ← {}, cid ← −1
foreach (s*_a, s*_l, s*_max, t̂*_md) in L* do
    cid* ← ⌊(s*_a − s_min)/d⌋
    if cid* > cid then
        append (s*_a, s*_l, t̂*_md) to L_2
Save L_2 in new MMDM's cache_2 using h
```

record's $[s_a : s_l]$, if the record and a segmented MMDM has the relationship of $[s^*_a : s^*_l] \subset [s_{\min} : s_{\max}]$, then the record is move to the MMDM's $cache_2$ after checking the cid duplication, as shown in Fig. 5(b).

IV. 1/λ-CHANGE DETECTION AND FF-HISTORY MANAGEMENT

FF through the new abstraction layer reduces e_ℓ scheduling when delivering events to the destination groups. However, FF leads to problems in detecting the change in e_ℓ-arrival due to the bypassed lots. Moreover, the divergence of a group can influence a previously fast-forwarded e_ℓ because the $\hat{t}_{md}$ derivation expects that all groups involved with the bypassed steps remain in steady states before the scheduled e_ℓ is delivered to its destination. The following section describes the methods to resolve these two problems.

A. REFRESHMENT OF 1/λ-TESTING SAMPLE

The ALC of a steady-state group routes an input e_ℓ to a destination MDM (one of the SMDM or MMDMs). Before routing, the ALC monitors the input e_ℓ's $1/\lambda$ to update a recent time-varying $1/\lambda$ sample, which is utilized to detect an e_ℓ-arrival change through statistical testing with a counterpart sample, a steady-state $1/\lambda$ sample. The steady-state sample is static and derived at the detection of a steady-state convergence. If two samples are statistically distinguishable, the group's DEM is reactivated instead of the SMDM. We denote the **steady-state** and **recent 1/λ samples** and $\boldsymbol{S_s}$ and $\boldsymbol{S_r}$, respectively. Both samples consist of the mean $1/\lambda$ observations of each local $1/\lambda$ group (described in Sec. II-A).

Due to the missed $1/\lambda$ observation of the bypassing e_ℓs, the S_r is highly likely to deviate from the S_s. Considering the lot bypassing, we propose an S_s refreshment method following this perspective: the ALC does not check the arrivals of the e_ℓ outputs from a steady-state group because the steady-state system's outputs are already stable. Thus, if a previous group runs in a steady state, the S_s is reinitialized after filtering the $1/\lambda$ observations from the group-departing e_ℓs. Conversely, if a group diverges, then the previous filtering is reverted.

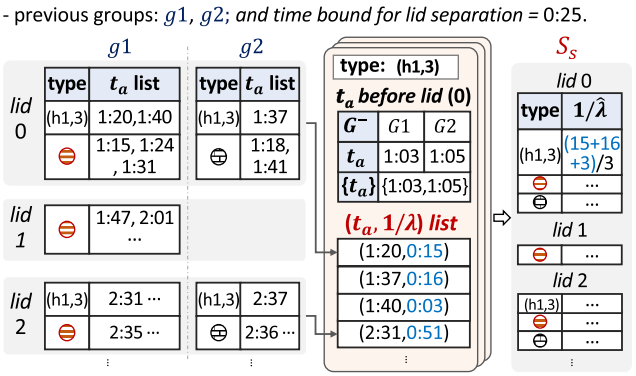


FIGURE 6. The sample data management and 1/λ sample derivation.

For the convenient refreshment of S_s, the ALC manages the set of t_a lists by each previous machine groups and by local t_a groups, as shown in Fig. 6. We denote each local t_a group's id as lid. For the construction of the S_s, the ALC maintains the set of intermediate lists that consist of the $(t_a, 1/\lambda)$ nodes by the lot type. Each node's $1/\lambda$ in the intermediate lists is calculated using the previous node's t_a. To derive the first node's $1/\lambda$, the ALC manages the last t_a values by each previous machine group before the first lid observation. Each lid's $1/\hat{\lambda}^l$ of the S_s is derived from the $(t_a, 1/\lambda)$ nodes of the l-type intermediate list.

If a previous group converges or diverges, the ALC traverses the $(t_a, 1/\lambda)$ lists to delete or add the corresponding nodes that are collected from the previous group's e_ℓ outputs. For example, in Fig. 6, if the group $g1$ converges, the specific nodes of the $(h1, 3)$-type intermediate list, which are $(1 : 20, 0 : 15)$, $(1 : 40, 0 : 03)$, and $(2 : 31, 0 : 51)$, are removed based on the $g1$'s t_a lists in each lid. Since the insertion or removal of a $(t_a, 1/\lambda)$ node influences the next node's $1/\lambda$, the next node should be revised based on the changed previous node's t_a. If a previous group's convergence (or divergence) changes the last t_a of a type before the

first *lid* collection, the first node's $1/\lambda$ of the intermediate list should be revised accordingly. If any $(t_a, 1/\lambda)$ node of an *lid* is inserted, deleted, or revised, the *lid*'s $1/\hat{\lambda}$ is recalculated using the new nodes' $1/\lambda$ values.

Whenever the S_s is refreshed, the S_r is also initialized to S_s, which makes the sample distance between S_s and S_r zero by ignoring the previous sample distance based on the optimistic expectation: the group operates in a steady state without starting a divergence. After the refreshment of S_s, if the number of the remaining *lid*s is less than a defined minimum bound for the KS test, the ALC collects the t_a values of new *lid*s using incoming e_ℓs from previous unsteady groups. The collection proceeds until the number of *lid* is over the minimum bound. The frequent S_r initialization and additional t_a collection for a small-size sample can lead to the late detection of the e_ℓ-arrival change at the group's divergence, which decreases the accuracy.

The ALC sometimes can receive a scheduled e_ℓ whose last step undergoes a steady-state group if the receiver group turns into a steady state after the departure of the event. Then, the ALC directly forwards the e_ℓ to a destination MDM without monitoring the $1/\lambda$ for the recent $1/\lambda$ sample.

B. FF-HISTORY MANAGEMENT

When a fast-forwarded e_ℓ is scheduled using a multi-step mean delay, FF assumes that all involved machine groups remain in steady states before the e_ℓ arrives at its destination on time. However, a group's unexpected divergence can violate the assumption. Even in an SMDM without FF, a current-time divergence can influence a previously scheduled e_ℓ using a single-step mean-delay ($\hat{t}_d$) if the event's t_e is later than t_c. We embrace this violation within an SMDM as a marginal error based on the assumption that a divergence can affect some of the overall WIP lots (whose number is $\sum_l \hat{\lambda}^l \cdot \hat{t}_d^l$, where l is the lot type). Moreover, the influence is not exceptionally significant because the WIP lots are in an early stage before the t_d changes drastically.

Unlike the violation within the SMDM, FF across multiple groups enables distant future events (scheduled using the accumulated mean delays for numerous processing steps) to bypass many intermediate groups. The FF of remote forthcoming lots can significantly reduce the simulation accuracy because a bypassed group's current-time divergence can change the overall production dynamics significantly by affecting other machine groups. The gap in multi-step delay between the accurate DE simulation result and MMDM-based estimation can vary seriously as the number of bypasses increases. Therefore, we define an e_ℓ-rescheduling condition to be actively resolved, below.

Definition 7: If a fast-forwarded e_ℓ's t_a for an i-th step at a diverging group (that is previously bypassed by the lot) is to the future of t_c (i.e., $t_a[i] > t_c$), then the event satisfies the rescheduling condition at that step.

If a previously scheduled e_ℓ meets the rescheduling condition, then the e_ℓ is canceled to be discarded after being cloned. The cloned event is scheduled to be delivered to the diverging group for the i-th step at the future time of $t_a[i]$. The cancelled e_ℓ in the simulation engine's event list is abandoned when the e_ℓ is processed as a head event.

Some machine groups can handle multiple processing steps for $\{i\}$ for a specific route. If a forwarded e_ℓ, which is previously scheduled, had bypassed one or more steps among the $\{i\}$, then the rescheduling condition should be checked at each bypassed step. For the efficient examination of the rescheduling condition, ALCs manage global FF history tables to save the FF histories by routes, as defined below.

Definition 8: Each FF history table (HT) for a route is designed to map each fast-forwarded e_ℓ to the pair of $flag_a$ and LL_b, where

- $flag_a$ is the integer that contains all of the bypassed steps at the bit level; and
- LL_b is the list of the $(flag_p, L_p)$ nodes, $flag_p$ is the integer that contains the successive and increasing-ordered bypassed steps at the bit level, L_p is the list that contains the partial bypassed records of $(b, t_a[b])$, b is a bypassed step; and $t_a[b]$ is the t_a at the b-handling group for the b-th step.

When a group that handles several processing steps of a route diverges, the $flag_a$ helps to check whether a fast-forwarded e_ℓ following the route had previously bypassed the diverging group. For example, if a lot had bypassed 2 and 3 steps of a route, then the $flag_a$ is set to 1100_2. If the diverging group's processing steps for a route are 3, 7, and 10, then the static group flag ($flag_g$) is 10010001000_2. Through the bit-wise AND operation between the $flag_a$ and $flag_g$ (i.e., $flag_a \wedge flag_g$), the bypassed group can be confirmed if the operation result is not 0.

Whenever a stochastic transition happens during FF, a fast-forwarded e_ℓ's LL_b in the route's HT inserts a new node of $(flag_p, L_p)$. For example, if the sequence of the bypassed steps of a lot is 2, 3, 4, 6, 7, and 8, then the $flag_p$s of the lot's LL_b are 11100_2 and 111000000_2. The L_p contains the records of the minimum and maximum bypassed steps ($b^\circ_{\min}$ and $b^\circ_{\max}(= s_l)$) in an MDM. In the previous example, if a lot bypassed steps 2, 3, and 4 in an MMDM, then the $(b, t_a[b])$ nodes for steps 2 and 4 are recorded, which are utilized to find the t_a at the diverging group if $flag_a \wedge flag_g$ is not zero.

If an MDM receives an e_ℓ from an ALC, then the $b^\circ_{\min}$ at the MDM is (s_a+1) because all lot events routed by ALCs are scheduled events that arrive on time (i.e., $t_e = t_c = t_a[s_a]$). Based on this property, the $b^\circ_{\min}$ record for an MDM can be summarized as follows:

$$(b^\circ_{\min}, t_a[b^\circ_{\min}]) = \begin{cases} (s_a, t_c+\hat{t}_{pd}), & \text{if src. is MDM or} \\ (s_a+1, t_c+\hat{t}_d(h, s_a)) & \text{otherwise.} \end{cases} \tag{2}$$

When an MDM attempts to save an e_ℓ's $b_{\min}$ record, if the LL_b for the event does not exist, then the MDM initializes the LL_b whose first node is the pair of 0 and empty L_p and adds the record to the empty L_p. When the FF occurs over the successive steps across multiple MDMs, each MDM inserts the $b_{\min}$ record into the current L_p—that is, $\mathsf{tail}(LL_b)$'s L_p.

When an MDM schedules a forwarded e_ℓ or delivers the e_ℓ to another MDM, the MDM attempts to insert the $b^{\circ}_{\max}$ record. The record's $t_a[b^{\circ}_{\max}]$ is derived as $t_c + \hat{t}_{pd} + \hat{t}_{md} - \hat{t}_d(h, s_l)$. During the attempt, the MDM should inspect the s_l to see whether the s_l is a bypassed step and s_l is larger than the last node's b in the current L_p. Because all bypassed steps of e_ℓ at the SMDM have the same s_a and s_l, the SMDM should not insert the record for the s_l being bypassed. After inserting the last record of an e_ℓ in the current L_p without any further insertion due to the e_ℓ scheduling or a stochastic step transition, the $flag_p$ should be updated as $(2^{b_{\max}-b_{\min}+1}-1) \ll b_{\min}$, where $b_{\min}$ and $b_{\max}$ are the first and last nodes' b elements, respectively. For example, if an L_p's $b_{\min}$ and $b_{\max}$ are 2 and 4, then the $flag_p$ is $00111_2 \ll 2$, which is 11100_2. If the e_ℓ's FF is over, then the e_ℓ-scheduling MDM updates the $flag_a$ through the OR operation of the LL_b's $flag_p$ values. If the scheduled e_ℓ is delivered at $t_e(= t_c)$, then the e_ℓ's history is removed from the HT.

When a group involved with specific routes diverges, all FF histories of e_ℓs in the routes' HTs are examined. The history examination of an e_ℓ is skipped if the AND operation result between the e_ℓ's $flag_a$ and the group's $flag_g$ becomes 0. If the result is not 0, the diverging group's ALC starts to traverse the nodes in the e_ℓ's LL_b in order to check the e_ℓ's rescheduling condition after identifying the t_a(s) at the group.

While traversing each L_p in the e_ℓ's LL_b, the ALC removes the L_p if the $t_a[b_{\max}]$ is less than t_c, based on the following two lemmas, which alleviates the future traverse overhead in another bypassed group's forthcoming divergence.

Lemma 4.1: Let the set of the group-involved steps be $\{i\}$ *and* $i^* \in \{i\}$. *Suppose that an* L_p*'s* $flag_p$ *and* $t_a[b_{\max}]$ *meet the properties of* $((1 \ll i^*) \wedge flag_p) > 0$ *and* $t_a[b_{\max}] \le t_c$. *Then,* $t_a[i^*]$ *cannot meet the rescheduling condition (that is,* $t_a[i^*] > t_c$*).*

Proof: Let $(b_1, t_a[b_1])$ and $(b_2, t_a[b_2])$ be two successive nodes in L_p. The records have relationships such that $b_1 < b_2$, $t_a[b_1] < t_a[b_2]$ because each record was stored sequentially during FF over the successive steps. Thus, we may notice that $i^* \le b_{\max}$, $t_a[i^*] \le t_a[b_{\max}] \le t_c$. Therefore, $t_a[i^*] \le t_c$; hence, $t_a[i^*]$ cannot be larger than t_c. □

Lemma 4.2: If $t_a[b_{\max}] \le t_c$, *then no bypassed steps in the* L_p *can meet the rescheduling condition in the future.*

Proof: Let $b \in [b_{\min} : b_{\max}]$ and t_c^+ be a future time for a group divergence that affects b. For all b, we can notice that $t_a[b] \le t_a[b_{\max}] \le t_c \le t_c^+$. Therefore, no $t_a[b]$ can be larger than any future time. □

If $flag_a \wedge flag_g$ is not 0, then the overall procedures for an e_ℓ's FF history examination using the LL_b are described in Alg. 3. After removing the inspection-free L_ps based on Lemma 4.1 and 4.2 (as Line 4), the diverging group's ALC checks whether any step in the L_p is involved with the diverging group using $flag_p \wedge flag_g$. If the L_p is involved, then the ALC removes the inspection-free nodes and checks the L_p's emptiness (as in Lines 7-10). If the L_p is not empty, then the ALC attempts to find specific steps involved with the current L_p ($\{i^*\}$), as in Lines 11-15. If $\{i^*\}$ is not empty, then the ALC attempts to derive each $t_a[i^*]$ to determine whether the e_ℓ meets the rescheduling condition at the i^*.

For each $t_a[i^*]$ examination, the ALC traverses the L_p's node until it finds a node whose b is larger than or equal to i^*. After deriving the $\hat{t}_{md}$ for the multiple step of $[i^* : b-1]$,

Algorithm 3: An e_ℓ's History Examination Using the LL_b, If $(flag_a \wedge flag_g) > 0$

```
1  cache: a hash table that maps (h, b⁻, b⁺) to Σ_{j=b⁻}^{b⁺} t̂_d(h, j)
2  foreach (flag_p, L_p) in LL_b do
3      (b_min, t_a[b_min])←head(L_p), (b_max, t_a[b_max])←tail(L_p)
4      if t_a[b_max] ≤ t_c then delete (flag_p, L_p) and continue
5      flag ← flag_p ∧ flag_g
6      if flag ≠ 0 then
           // examine L_p's nodes
7          foreach node in L_p do
8              if node's t_a[b] ≤ t_c then delete node
9              else break
10         if L_p is empty then delete (flag_p, L_p) and conti.
           // derive the group-involved steps ({i*})
11         flag ← flag ≫ b_min, {i*} ← ∅
12         for i = 0; i ≤ b_max − b_min; i = i + 1 do
13             if ((1 ≪ i) ∧ flag) > 0 then
14                 add (i + b_min) to {i*}, flag←flag∧(1 ≪ i)
15                 if flag = 0 then break
           // examine t_a at each i*
16         node←head(L_p)
17         {i*} ← {i*} − {j | j ∈ {i*} ∧ j < node's b}
18         foreach i* in {i*} do
19             while (node's b) < i* do node←next(node)
               // derive t_a[i*] using node
20             (b, t_a[b]) ← node
21             if cache has key of (h, i*, b − 1) then
22                 t̂_md ← (h, i*, b − 1)'s value
23             else
24                 t̂_md ← Σ_{j=i*}^{b−1} t̂_d(h, j), save t̂_md to cache
25             t_a[i*] ← t_a[b] − t̂_md
26             if t_a[i*] > t_c then // resch. needs?
27                 e'_ℓ ← e_ℓ.clone(), cancel e_ℓ, and resch. e'_ℓ
28                 replace the HT's key as e'_ℓ
29                 remove the next records in the L_p
30                 remove the subsequent L_p(s) in the LL_b
                   // update new last bypassing step
31                 b' ← i* − 1, t_a[b'] ← t_a[i*] − t̂_d(h, l)
32                 if prev(node) ≠ null then
33                     b⁻ ← prev(node)'s b
34                 else
35                     b⁻ ← b_min
36                 if t_a[b'] > t_c and b' > b⁻ then
37                     change node to (b', t_a[b'])
38                 else delete node
39                 break
40         if L_p is empty then delete (flag_p, L_p)
41         else update flag_p using L_p
42 update flag_a using the set of flag_p
```

the ALC calculates the $t_a[i^*]$, as in Lines 20-25. Because the lots following the same route are forwarded through the same path, there is a high possibility that the $\hat{t}_{md}$ computation will be duplicated. Thus, the ALC saves the $\hat{t}_{md}$ in cache to reuse for the other forwarded lots during the current HT examination. If $t_a[i^*]$ is larger than t_c, then the ALC performs the e_l-rescheduling processes for the new cloned e_ℓ (e'_ℓ) to arrive at the diverging group at the time of $t_a[i^*]$. The ALC replaces the lot's key in the HT with the e'_ℓ. Then, the ALC removes all of the subsequent records in the current L_p as well as the following L_ps due to the changed final step of i^*.

After checking the bypassing of the new last step $b'(= i^* - 1)$ and its recording condition, the ALC updates the current node to $((b', t_a[b']))$ or removes the node, as in Lines 31-39. After the examination, if the last L_p is empty, then the ALC removes the list. Otherwise, the ALC updates the $flag_p$ using the revised L_p. After the history examination, the ALC updates the $flag_a$ using each $flag_p$ in the LL_m.

V. EXPERIMENTATION

We first designed a simple test model (denoted as the case-1 model) to manifest the proposed FF mechanism. The structure of the fab model is illustrated in Fig. 7. There are five machine groups, each having a wafer-lot queue, a machine, and an operator. The inventory model supplies wafer lots to the first machine group, EG_A, for the production of two different products. For these two products, wafer lots have two types of headers: ls_1 and ls_2. They require different routing paths: the order of EG_A, EG_B, EG_C, and EG_D (for the ls_1 header), or the sequence of EG_A, EG_B, EG_C, and EG_E (for the ls_2 header). Each group has a single machine. The machine for EG_C is a batch-processing unit representing a furnace and has a relatively long processing time.

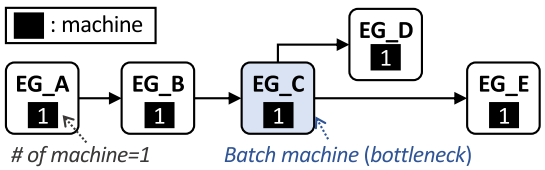


FIGURE 7. Structure of the case-1 simple factory.

Under certain uniform release rates of these two types of lots, all machine groups operate in steady states in which their queue lengths diverge within some boundaries (See Fig. 8(a)). Whenever group EG_B receive an e_m for the periodic inspection, the group's DEM becomes reactivated. Before receiving the first e_m, when all groups are running in steady states, two MMDMs for the two different routing paths fast forward the incoming lots, as shown in Fig. 8(b) and (c). FF bypasses the four steps in the serial machine groups in each path. The maintenance of EG_B influences the input arrivals of the following groups so that the following groups become unsteady in a cascading fashion. When any MMDM is deallocated due to the absence of sequential steady-state groups, some wafer lots stored in the history tables can be rescheduled if the lots meet the cancellation condition (see Def 7). The rescheduling is performed after nullification of the previously scheduled e_ℓ events; the numbers of the canceled lots are traced, as shown in Fig. 8(c).

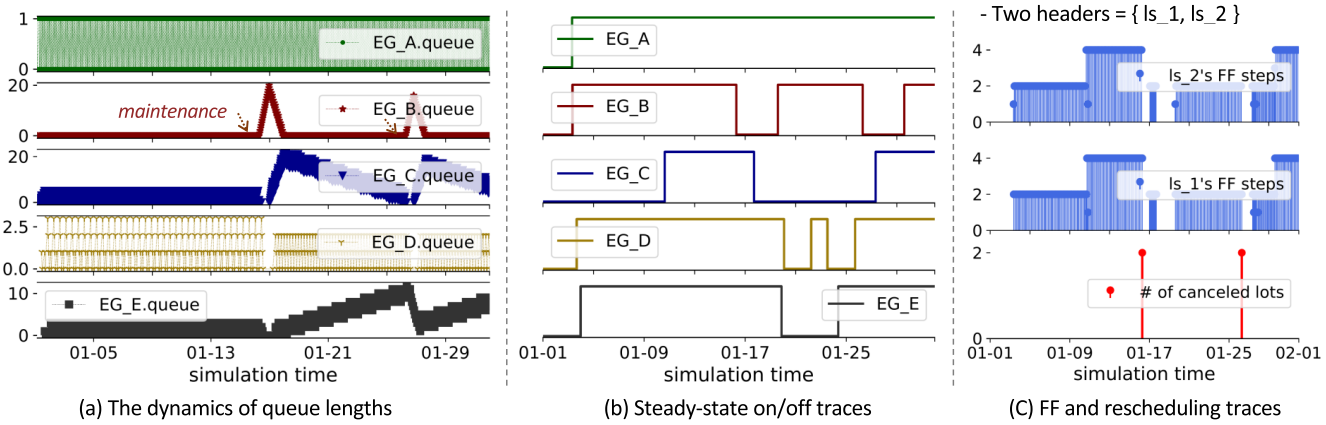
The 40-day simulation results of case 1 are summarized in Tbl 1. As the aggregation level increases, both the number of events processed by the simulation engine and the accuracy of the simulation decrease. The reduced number of events guarantees lower simulation-execution times. We measured execution times using an experimental machine with an Intel® Xeon® E5-1650 3.6GHz CPU and 32 GB of memory. We determined the testing parameters for the statistical convergence/divergence detection; among these parameters, the number of *lid* (for each test sample) was set to 20 and the p-value for the KS-based convergence test was set 0.7. We defined the simulation accuracy as the mean difference in the cycle-time distributions of the wafer lots, which results from the baseline DE-only simulation and the counterpart A/D simulation. This definition for the accuracy measurement is designed to consider the stochastic step changes of the lots.

TABLE 1. Summary of case-1 simulation results.

Aggregation Type	MMDM+SMDM	SMDM	No Aggr.
# of events	20788	26233	45088
accuracy (%)	91	93	100
MDM-activity ratio	0.62	0.53	N.A.
# of abl. change	37	47	N.A.
execution time (ms)	58	71	103

Compared to the single-group A/D (that utilizes only SMDMs for the abstraction), the proposed FF through the multi-group A/D (that employs both SMDMs and MMDMs) does not test the arrivals of the lots from steady-state groups. In the previous single-group A/D approach, the changes in the e_ℓ outputs of a group caused by the model switching between the SMDM and DEM can lead to a divergence of the next steady-state group. The proposed sample reinitialization in the multi-group A/D prevents the extreme reaction to a previous group's output change caused by the SMDM activation by excluding the observation of the e_ℓ arrival when the group is in a steady state. Thus, the reduced number of divergence detections increases the MDM activity ratio (MAR). However, the reinitialization of the testing samples can delay the response to the input-arrival change, resulting in a degradation in accuracy.

We designed another case (denoted as the case-2 model) employing a majority of machines to evaluate the speedup of simulations and the change in accuracy for the proposed approach. As shown in Fig. 9, the model has four types of machine groups for general process steps: deposition, patterning, etching, and chemical-mechanical polishing. The detailed subprocesses in each machine can differ depending on a lot's product recipe. The machines have different processing times according to the recipe and their stochastic properties; we referenced the timing parameters described in



(a) The dynamics of queue lengths (b) Steady-state on/off traces (C) FF and rescheduling traces

FIGURE 8. Overview of case-1 simulation scenario and runtime traces.

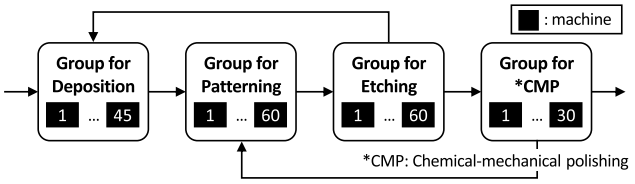
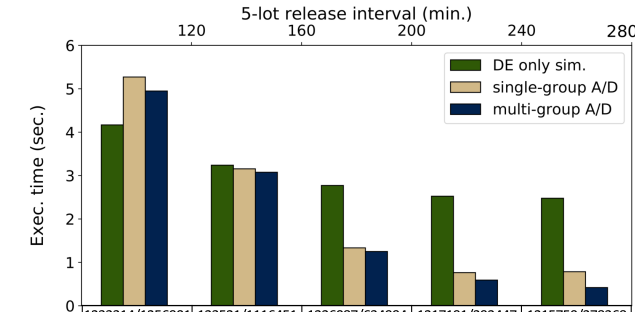
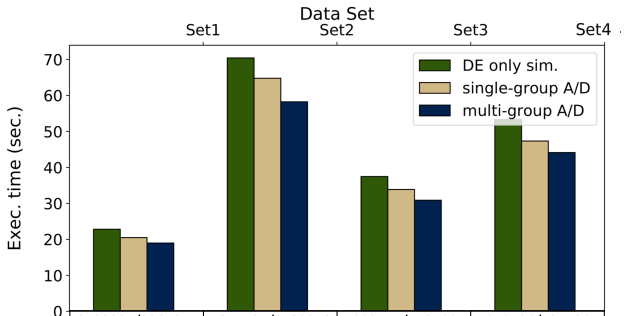


FIGURE 9. Structure of the case-2 wafer-fab model.



# of events	1233314/1256901 /1222267	123521/1116451 /1027858	1226087/634994 /4986000	1217191/392447 /235867	1215750/378368 /160449
MAR	**-/0.11/0.11**	**-/0.19/0.25**	**-/0.61/0.64**	**-/0.79/0.81**	**-/0.78/0.86**
abl. change	-/150/139	-/169/162	-/160/156	-/118/102	-/122/83
Acc. (%)	**100/95.1/93.2**	**100/92.3/91.7**	**100/97.8/96.7**	**100/98.6/97.1**	**100/98.8/97.5**
Speedup	**1/0.8/0.84**	**1/1.02/1.05**	**1/2.07/2.21**	**1/3.3/4.26**	**1/3.1/5.9**
FF steps	8796	67813	194203	250345	255135
# of cancel.	1180	24353	74550	90024	82168

FIGURE 10. The simulation result of case 2.



	Set1	Set2	Set3	Set4
# of events	3722901/3261241 /3011389	7874121/6482218 /6959834	5009574/4401665 /4154143	5953736/5275365 /5119032
MAR	**-/0.136/0.238**	**-/0.148/0.241**	**-/0.125/0.227**	**-/0.129/0.235**
abl. change	-/2166/1654	-/5169/4109	-/2882/2223	-/3307/2276
Acc. (%)	**100/96.4/91.7**	**100/95.0/0.93**	**100/96.5/94.6**	**100/96.2/95.6**
Speedup	**1/1.11/1.2**	**1/1.08/1.21**	**1/1.11/1.21**	**1/1.12/1.20**
FF steps	41280	48285	33541	12121
# of cancel.	4861	5510	3596	1222

FIGURE 11. The simulation result of SMT2020.

James's book [32], [33]. There are five types of products, each requiring different mask layers (such as 4, 8, 12, 16, or 20). As the required number of layers increases, the number of reentrances increases (e.g., deposition → patterning → etching → deposition · · ·). We measured the performance of the simulation under the proposed FF approach at various injection rates, and we display the results in Fig. 10. The total number of injected wafer lots is 5000.

Higher rates for lot release cause the monotonic increments of the machine groups' queues; thus, machine groups are more likely to be in unsteady states. The mean MAR will decrease accordingly. As the MAR decreases, the groups' DEMs mainly operate, which increases accuracy and decreases speedup. The divergence test, excluding the output of steady-state groups in the proposed FF approach, leads to a higher MAR than the single-group A/D. The reinitialization of the divergence-test sample for the FF results in lower accuracy than the single-group A/D, as discussed in the explanation for case 1. As the input rates become decrease, the MDMs are mainly active, and we achieved a speedup in the FF approach by up to 5.9 times.

We also applied the FF approach to existing test scenarios that contained high-mix production fab models [34]. The testbed called the SMT2020 dataset consists of four different types of production scenarios, whose route lengths are up to 632 operations for 44 mask layers. The total number of machines in the production datasets reaches 1054, with 107 machine groups. The overall simulation results for the two-month production with the existing WIP wafer lots are displayed in Fig. 11.

Using the proposed multi-group A/D approach, we obtained a speedup of approximately 20% for the four production scenarios compared to the DE-only simulation; the speedup for the single-group A/D was about 10%. This was a relatively low speedup compared to the case-2 results

due to frequent interruptions that prevented the MDM preservations, resulting in a low mean MAR of about 0.23. A frequent disruption is induced by various types of machine breakdowns and maintenances or by the different release rates of different types of lots with varying headers.

Except for the detection of a statistical change in lot arrival, a steady-state group can reactivate the DEM when the group's ALC receives the e_m or any new type of lot that has not been observed in the detection of steady-state convergence. The SMT2020 dataset scenarios consider various types of planned machine maintenance or unexpected breakdown that produce frequent e_m events. In the production scenarios, lot-release rates among different lot headers are significantly different. Some high-priority lots have relatively long release rates and made interruptions to steady-state group models as new types of lots because the lots are sometimes not observed during steady-state detection. In the next study, we will discuss how to compensate for the SMDM's steady-state operations under the interruptions caused by the considerably different release rates in the high-mix production and frequent arrivals of e_m, without reactivating the DEM.

VI. CONCLUSION

To speed up the DE simulation of target fab models, we proposed a hierarchical method for A/D simulation using two types of MDMs: SMDM and MMDM. The SMDM abstracts the complex DE operations of individual machine groups into mean delays. The MMDM fast-forwards input lots by bypassing intermediate steady-state groups. If the lot's subsequent group is in an unsteady state, then the MMDM schedules the lot to be stored in the event-list before being delivered to the destination group after a multi-step delay. FF aims at alleviating the event-processing overhead of the simulation engine by reducing the number of events scheduled. The critical component, ALC, switches its machine group's active model between the DEM and SMDM when the machine group converges into or diverges from a steady state. The ALC also allocates, deallocates, extends, or splits the MMDMs according to the changes in neighboring groups' steadiness. To consider the dynamic update of the MMDM and maximize the reuse of previous accumulation results, we proposed multi-step delay management using the two-level cache tables. The bypassed machine group causes a problem in detecting input-arrival change; to address this problem, we developed a method for reinitialization of the $1/\lambda$ sample according to changes in the previous groups' steadiness. Since a bypassed group's divergence can shift the previously derived multi-step delays of scheduled events, we designed an examination method using the FF history tables to reschedule the highly affected events. Compared to the accurate DE simulation, the proposed multi-group A/D shows a higher speedup than the single-group A/D approach and with less degradation in accuracy. Speedups vary by up to 5.9 times under various scenarios, with 2.5 to 8.3% reductions in accuracy, and the speedup enhancement is highly affected by the overall steadiness of the machine groups.

REFERENCES

[1] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018.

[2] H. Tang, D. Li, S. Wang, and Z. Dong, "CASOA: An architecture for agent-based manufacturing system in the context of industry 4.0," *IEEE Access*, vol. 6, pp. 12746–12754, 2018.

[3] M. Khakifirooz, M. Fathi, and K. Wu, "Development of smart semiconductor manufacturing: Operations research and data science perspectives," *IEEE Access*, vol. 7, pp. 108419–108430, 2019.

[4] D. Kopp, L. Mönch, D. Pabst, and M. Stehli, "Qualification management in wafer fabs: Optimization approach and simulation-based performance assessment," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 1, pp. 475–489, Jan. 2020.

[5] B. W. Hsieh, C. H. Chen, and S. C. Chang, "Efficient simulation-based composition of scheduling policies by integrating ordinal optimization with design of experiment," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 4, pp. 533–568, Oct. 2007.

[6] L. Reinhardt and L. Monch, "Simulation-based optimiziation to design equipment health-aware dispatching rules," in *Proc. Winter Simulation Conf. (WSC)*, Carson City, NV, USA, Dec. 2017, pp. 3565–3575.

[7] J. Liu, C. Li, F. Yang, H. Wan, and R. Uzsoy, "Production planning for semiconductor manufacturing via simulation optimization," in *Proc. Winter Simulation Conf. (WSC)*, Phoenix, AZ, USA, Dec. 2011.

[8] T. S. Ng and J. Fowler, "Semiconductor production planning using robust optimization," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage.*, Dec. 2007.

[9] D. P. Connors, G. E. Feigin, and D. D. Yao, "A queueing network model for semiconductor manufacturing," *IEEE Trans. Semicond. Manuf.*, vol. 9, no. 3, pp. 412–427, Aug. 1996.

[10] D. Grosbard, A. Kalir, I. Tirkel, and G. Rabinowitz, "A queuing network model for wafer fabrication using decomposition without aggregation," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Madison, WI, USA, Aug. 2013.

[11] D. Nazzal and L. F. McGinnis, "Queuing models of vehicle-based automated material handling systems in semiconductor fabs," in *Proc. Winter Simulation Conf.*, Orlando, FL, USA, 2005.

[12] J. G. Shanthikumar, S. Ding, and M. T. Zhang, "Queueing theory for semiconductor manufacturing systems: A survey and open problems," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 4, pp. 513–522, Oct. 2007.

[13] J.-Y. Bang and Y.-D. Kim, "Hierarchical production planning for semiconductor wafer fabrication based on linear programming and discrete-event simulation," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 326–336, Apr. 2010.

[14] W. Scholl and J. Domaschke, "Implementation of modeling and simulation in semiconductor wafer fabrication with time constraints between wet etch and furnace operations," *IEEE Trans. Semicond. Manuf.*, vol. 13, no. 3, pp. 273–277, Aug. 2000.

[15] D. Fronckowiak, A. Peikert, and K. Nishinohara, "Using discrete event simulation to analyze the impact of job priorities on cycle time in semiconductor manufacturing," in *Proc. ASMC*, Boston, MA, USA, 1996.

[16] M. A. Chik, A. B. Rahim, A. Z. M. Rejab, K. Ibrahim, and U. Hashim, "Discrete event simulation modeling for semiconductor fabrication operation," in *Proc. IEEE Int. Conf. Semiconductor Electron. (ICSE)*, Kuala Lumpur, Malaysia, Aug. 2014.

[17] Y.-H. Low, B.-P. Gan, S. Jain, W. Cai, W.-J. Hsu, S.-Y. Huang, and S. J. Turner, "Parallel discrete-event simulation of a supply-chain in semiconductor industry," in *Proc. 4th Int. Conf./Exhib. High Perform. Comput. Asia–Pacific Region*, Beijing, China, 2000.

[18] M. Seok and T. G. Kim, "Parallel discrete event simulation for DEVS cellular models using a GPU," in *Proc. SpringSim*, Orlando, FL, USA, 2012.

[19] T. Hildebrandt, D. Goswami, and M. Freitag, "Large-scale simulation-based optimization of semiconductor dispatching rules," in *Proc. WSC*, Savanah, GA, USA, 2014.

[20] Y. Ma, F. Qiao, W. Yu, and J. Lu, "Parallel simulation-based optimization on scheduling of a semiconductor manufacturing system," in *Proc. WSC*, Savanah, GA, USA, 2014.

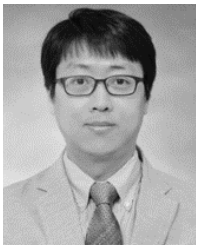
[21] R. T. Johnson, J. W. Fowler, and G. T. Mackulak, "A discrete event simulation model simplification technique," in *Proc. Winter Simulation Conf.*, Orlando, FL, USA, 2005.

[22] O. Rose, "Improved simple simulation models for semiconductor wafer factories," in *Proc. Winter Simulation Conf.*, Washington, DC, USA, Dec. 2007.

[23] C. P. L. Veeger, L. F. P. Etman, E. Lefeber, I. J. B. F. Adan, J. van Herk, and J. E. Rooda, "Predicting cycle time distributions for integrated processing workstations: An aggregate modeling approach," *IEEE Trans. Semicond. Manuf.*, vol. 24, no. 2, pp. 223–236, May 2011.
[24] J. M. Bekki, J. W. Fowler, G. T. Mackulak, and B. L. Nelson, "Indirect cycle time quantile estimation using the Cornish–Fisher expansion," *IIE Trans.*, vol. 42, no. 1, pp. 31–44, Oct. 2009.
[25] M. G. Seok, C. W. Chan, W. Cai, H. S. Sarjoughian, and D. Park, "Runtime abstraction-level conversion of discrete-event wafer-fabrication models for simulation acceleration," in *Proc. ACM SIGSIM Conf. Princ. Adv. Discrete Simulation*, Miami, FL, USA, Jun. 2020.
[26] M. G. Seok, W. Cai, H. S. Sarjoughian, and D. Park, "Adaptive abstraction-level conversion framework for accelerated discrete-event simulation in smart semiconductor manufacturing," *IEEE Access*, vol. 8, pp. 165247–165262, 2020.
[27] A. S. Shirazi, T. Davison, S. V. Mammen, J. Denzinger, and C. Jacob, "Adaptive agent abstractions to speed up spatial agent-based simulations," *Simul. Model. Pract. Theory*, vol. 40, pp. 144–160, Jan. 2014.
[28] R. Franceschini, M. Challenger, A. Cicchetti, J. Denil, and H. Vangheluwe, "Challenges for automation in adaptive abstraction," in *Proc. ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Munich, Germany, Sep. 2019.
[29] R. Franceschini, S. V. Mierlo, and H. Vangheluwe, "Towards adaptive abstraction in agent based simulation," in *Proc. Winter Simulation Conf. (WSC)*, National Harbor, MD, USA, Dec. 2019.
[30] G. Marsaglia, W. W. Tsang, and J. Wang, "Evaluating Kolmogorov's distribution," *J. Stat. Softw.*, vol. 8, pp. 1–8, Nov. 2015.
[31] L. Carvalho, "An improved evaluation of Kolmogorov's distribution," *J. Stat. Softw.*, vol. 65, no. 3, pp. 1–4, 2015.
[32] S. P. Ignizio, *Optimizing Factory Performance: Cost-Effective Ways to Achieve Significant and Sustainable Improvement*. New York, NY, USA: McGraw-Hill, 2009.
[33] L. Mönch, J. W. Fowler, and S. J. Mason, *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*. New York, NY, USA: Springer, 2012.
[34] M. Hassoun, D. Kopp, L. Mönch, and A. Kalir, "A new high-volume/low-mix simulation testbed for semiconductor manufacturing," in *Proc. Winter Simulation Conf. (WSC)*, National Harbor, MD, USA, Dec. 2019, pp. 2419–2428.

MOON GI SEOK received the B.S. degree in electronics engineering from Korea University, Seoul, South Korea, in 2009, and the M.S. degree and Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2011 and 2017, respectively. He was a Postdoctoral Researcher with KAIST and Arizona State University (ASU), Tempe, AZ, USA, in 2018 and 2019. He is currently a Research Fellow with Nanyang Technological University, Singapore. His research interests include multi-level modeling, simulation, and verification of system-on-chip (SoC) designs, and high-performance simulation methodology in various domains.

WENTONG CAI (Member, IEEE) is currently a Professor with the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU), Singapore. His research interests include modeling and simulation, and parallel and distributed computing. He is a member of the ACM. He also is an Associate Editor of the *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, an Editor of the *Future Generation Computer Systems (FGCS)*, and an Editorial Board Member of the *Journal of Simulation (JOS)*.



DAEJIN PARK received the B.S. degree in electronics engineering from Kyungpook National University, Daegu, South Korea, in 2001, the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2003 and 2014, respectively. From 2003 to 2014, he was a Research Engineer with SK Hynix Semiconductor and Samsung Electronics, where he has worked on designing low-power embedded processors architecture and implementing fully AI-integrated system-on-chip with intelligent embedded software on the custom-designed hardware accelerator, especially for hardware/software tightly coupled applications, such as smart mobile devices and industrial electronics. Since 2014, he has been a full-time Assistant Processor with the School of Electronics and Electrical Engineering and the School of Electronics Engineering, Kyungpook National University. He has published over 180 technical articles and 40 patents. In 2014, he was nominated as one of the presidential research fellows 21, Republic of Korea.