# Multicluster Tools Scheduling: An Integrated Event Graph and Network Model Approach

Shengwei Ding, Jingang Yi, *Member, IEEE*, and Mike Tao Zhang, *Senior Member, IEEE*

***Abstract*—Steady-state throughput and scheduling of a multicluster tool become complex as the number of modules and clusters grows. We propose a new methodology integrating event graph and network models to study the scheduling and throughput of multicluster tools. A symbolic decision-move-done graph modeling is developed to simplify discrete-event dynamics for the multicluster tool. This event graph is further used for searching feasible action sequences of the cluster tool. By representing sequences with networks, an extended critical path method is applied to calculate the corresponding cycle time. Grouping methods that are based on network are also introduced to reduce the searching complexity. Compared with optimization-based scheduling approaches, the proposed methodology can directly capture the cyclic characteristic of cluster tool schedules and be applied to analyze the impact of process and wafer flow variations on cycle time and robot schedules. We have successfully applied this new methodoloy to dozens of cluster tools at Intel Corporation. A chemical–mechanical planarization polisher is employed as an example to illustrate and validate the proposed methodology.**

***Index Terms*—Cluster tool, event graph, network, scheduling, semiconductor manufacturing.**

## I. Introduction

MODELING and scheduling are critical to improving the production throughput and enhancing the design of cluster tools in semiconductor manufacturing. A cluster tool consists of three types of modules: process modules (PM), transfer modules (TM), and cassette modules (CM) [Fig. 1(a)]. Process modules execute the semiconductor manufacturing processes, cassette modules store wafers for load and unload, and transfer modules (such as single-blade or double-blade robots) move the wafers among process modules as well as between process and cassette modules. In general, a single-cluster tool consists of one transfer module and a few cassette and process modules [Fig. 1(a)], and a multicluster tool consists of several single-clusters that are interconnected via buffer modules.

Scheduling of a single-cluster tool depends greatly on the cluster tool configurations and wafer flows. For example, if there are two wafers to be picked and placed by a single-blade transfer robot, the moving sequence can only be sequential pick/place pairs. In contrast, a double-blade robot can use the second blade (or arm) as a buffer; therefore, various pick/place sequences exist. Consequently, the analysis of double-blade robot schedules could become complicated even for a single wafer flow [1], [2]. For multicluster tools, steady-state throughput depends on the multiple robot coordinations, and robot scheduling is much more complicated.

When cluster tools run at the steady state, robots and process modules repeat their movements periodically. One-unit cycle production is widely considered. One-unit cycle means that within one cyclic period, each robot and process action has taken place exactly once, and only one wafer has been finished processing during this time period [1]. For cluster tools, we call the minimal one-unit cyclic period a fundamental period, denoted as $\mathbf{FP}$. It is easily observed that achieving a fundamental period of a cluster tool is equivalent to maximizing the tool throughput. The robot and process schedules that achieve the fundamental period are called optimal schedules. For a cluster tool, there could exist multiple optimal schedules that lead to the same fundamental period [3].

In [4] and [5], analytical models at steady state throughput are discussed for a single-cluster tool equipped with single-blade and double-blade robots, respectively. For a single-cluster tool with a single-blade robot, an optimal schedule in [4] is a "pull" strategy. For a double-blade robot, Venkatesh *et al.* [5] propose the optimal schedule by a "swap" action. In [1] and [2], some general results of throughput and scheduling analysis for single-grip robotic cells are discussed. It has been proven that for the single-blade cluster tool the "pull" strategy is *one* optimal schedule. The results presented in [3] for double-grip robotic cells show that the "swap" schedule is *one* of the optimal strategies. Moreover, it has been shown that finding all optimal schedules for a single-cluster tool with a double-blade robot is NP-hard [3]. Perkinson *et al.* [6] present the impact of parallel (redundant) process modules and revisiting wafer flows on steady-state throughput.

To model the cluster tool process flows, Srinivasan [7] and Zuberek [8] use Petri nets to study the performance of the cluster tool processes. Rostami *et al.* [9] and Rostami and Hamidzadeh [10] have used linear programming and heuristic methods to study the optimal schedules for a single-cluster tool with residency constraints on transfer and process modules. Simulation of cluster tools also plays an important role in studying the throughput and in optimizing the process and design. LeBaron and Hendrickson [11] and LeBaron and Pool [12] discuss the use of AutoSched AP (ASAP) to simulate the cluster performance by emulating the real-time cluster tool scheduler. In [13], genetic algorithms are utilized to improve cluster tool performance based on the simulation study. Recently, event graph

Manuscript received May 17, 2005; revised April 17, 2006.

S. Ding is with the Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720 USA (e-mail: dingsw@cal.berkeley.edu).

J. Yi is with the Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: jgyi@tamu.edu).

M. T. Zhang is with AzFSM Industrial Engineering, Intel Corporation, Chandler, AZ 85248 USA (e-mail: mike.zhang@intel.com).

Digital Object Identifier 10.1109/TSM.2006.879414

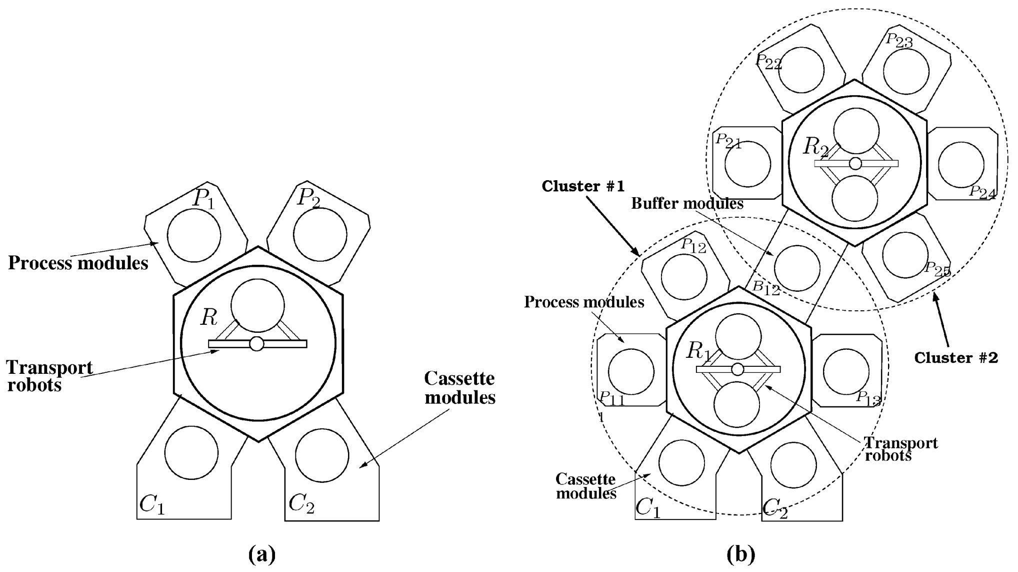

Fig. 1. Layout schematic of cluster tools: (a) a single-cluster tool and (b) an interconnected multicluster tool.

modeling of cluster tools has been studied to accurately calculate the throughput and to identify bottlenecks for various configurations [14]–[16]. Herrmann *et al.* [17] modeled some simple cluster tools into a network and applied the critical path method (CPM) to study the impact of processing time variations on steady-state throughput and robot scheduling for single-cluster tools.

Most of the previously mentioned works discuss the single-cluster tool. The geometry of the single-cluster tool is compact, and analysis and scheduling for such a tool is relatively straightforward. For a multicluster tool, it is challenging to analyze the throughput and robot schedules due to the fact that multiple robots could run independently and coordinately. Using a decomposition method approach, Yi *et al.* [18] first formally discuss the throughput analysis and robot scheduling of a general multicluster tool. In [19], a similar decomposition idea is also discussed for one cluster tool with three transfer robots and parallel process modules. Simulation study has been applied to verify that a "pull" strategy for a single-blade robot could result in the maximum tool throughput most of the time.

In this paper, we propose an integrated event graph and network model to optimize the robot scheduling of multicluster tools. Scheduling a cluster tool is proposed as a two-step process: in the first step, a set of feasible sequences of robot movements and process module actions are determined. In the second step, the relative action timings of feasible sequences found in the first step are calculated. Finally, the optimal schedule(s) can be determined by comparison among all feasible sequences that give the shortest cycle time. We utilize the event graph [20], a discrete-event modeling method, as a vehicle to facilitate the first step. In the second step, feasible action sequences are modeled as a network and the critical path method (CPM) [21] is then applied to calculate the shortest time span. The minimim cycle time and the corresponding optimal schedules can thus be found by comparing the cycle time for all networks. Compared with the methods in [17], the proposed approach considers a more complex multicluster tool and can handle more general cluster tool configurations.

The remainder of the paper is organized as follows. In Section II, the event graph modeling of cluster tools is discussed. Section III presents the sequence search using event graph modeling. Section IV discusses the construction of networks from action sequences and the CPM to calculate the fundamental period (cycle time). Sensitivity analysis using network and computational reduction schemes using the grouping method are also discussed in this section. An example of the modeling and scheduling analysis of a CMP polisher is illustrated in Section V. The concluding remarks are presented in Section VI.

## II. Multicluster Tools and Event Graph Modeling

### A. *Multicluster Tools*

A multicluster tool is defined as a combination of several single-robot clusters that are interconnected through buffer modules. In general, an interconnected $M$-cluster tool as shown in Fig. 2 is considered. The $i$th cluster is denoted as $\mathbb{C}_i$ and the buffer module between $\mathbb{C}_{i-1}$ and $\mathbb{C}_i$ is denoted as $B_i$. We assume that each cluster connects to at least one other cluster and that the interconnections among clusters do not form a loop. Moreover, we only consider the case that all wafers follow the same identical visit flow and that the wafer flow visits each process module only once, e.g., no re-entrant visit or parallel processing modules. It is also assumed that: 1) the cassette modules always have wafers/spaces for transfer modules (robots) to pick/place at any time; 2) all robots take deterministic times to pick and place wafers; and 3) all processing times are deterministic. We also assume that no failure is considered for any components in the cluster tool.

### B. *Event Graph Modeling*

Due to the characteristics of the cluster tool operations, it is natural to use a discrete-event dynamic model to study such a system. In this paper, we use event graph modeling to capture the dynamics of cluster tools. In the following, an example of a

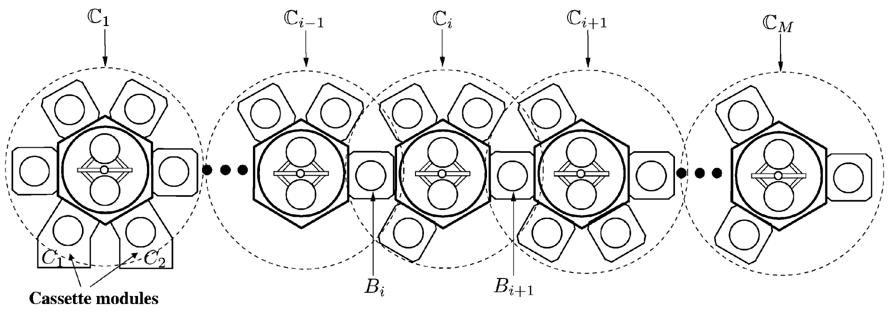

Fig. 2. Layout schematic of interconnected $M$-cluster tools.

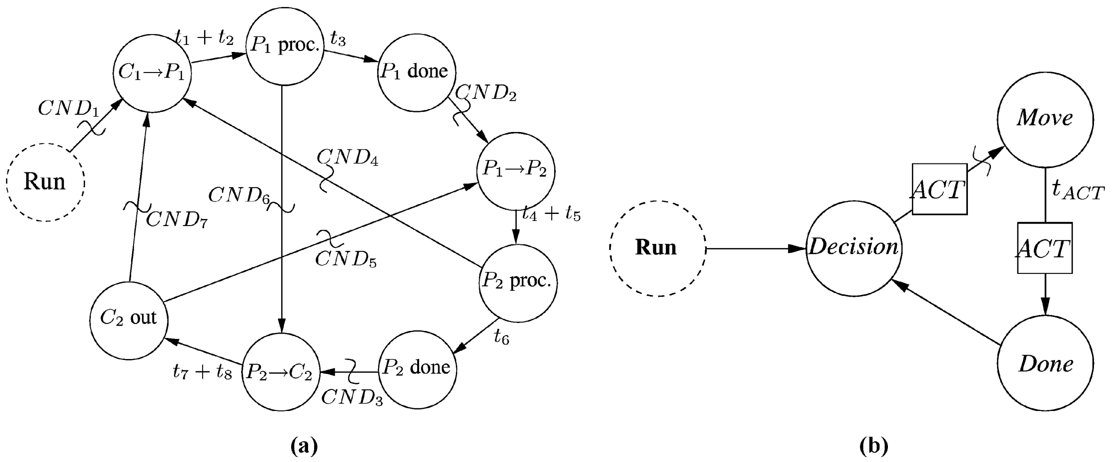

Fig. 3. Event graph-based modeling: (a) event graph for the example cluster tool shown in Fig. 1(a) and (b) decision-move-done event graph model.

single-cluster tool is used to illustrate how to construct an event graph model.

We consider the example of a single-cluster tool with two process modules, $P_1$ and $P_2$, and a single-blade transfer robot $R$, as shown in Fig. 1(a) ($C_i$, $i = 1, 2$ denote the cassette modules, and $P_i$, $i = 1, \cdots, N$ denote the $N$ process modules in a single cluster tool). Wafer flow is denoted as $C_1 \to P_1 \to P_2 \to C_2$, where the arrow stands for wafer transfer by robot $R$. Fig. 3(a) shows the event graph built for such a single-cluster tool, in which each solid circle represents one event where the state of the tool is changed. An action can be represented by two vertices, one for action start (e.g., "$P_1$ proc.") and another for action finish (e.g., "$P_1$ done"). A connecting arrow represents transition from one state to another with a specific time. For example, the connection from event "$C_1 \to P_1$" to event "$P_1$ proc." represents that the robot spends $(t_1 + t_2)$ time to take one wafer from $C_1$ and places it into $P_1$. A "$\sim$" sign on a connection arrow represents the required condition ($\text{CND}_1, \cdots, \text{CND}_7$ as listed in Table I) to proceed to the action.

As the number of modules or clusters increases, the layout of such an event graph becomes very large. Such complexity can be avoided by introducing a symbolic representation of the event graph as shown in Fig. 3(b). With this symbolic approach, all time-consuming actions are categorized into three vertices: decision, move (or action), and done. An action index, denoted as $\text{ACT}_i$, is further defined for all such actions as shown in Table II. A state variable vector $\mathbf{S}(t) = \{S_i(t)\}$ is also needed to track each transfer and process module operation status at time $t$. For

TABLE I
CAUSALITY CONDITIONS FOR THE EXAMPLE TOOL

| Label | From event | To event | Required conditions |
|---|---|---|---|
| $CND_1$ | Run | $C_1 \to P_1$ | $C_1$ has wafers, $R$ free, $P_1$ empty |
| $CND_2$ | $P_1$ done | $P_1 \to P_2$ | $P_1$ finishes, $R$ free, $P_2$ empty |
| $CND_3$ | $P_2$ done | $P_2 \to C_2$ | $P_2$ finishes, $R$ free |
| $CND_4$ | $P_2$ process | $C_1 \to P_1$ | $R$ free, $P_1$ empty |
| $CND_5$ | $C_2$ out | $P_1 \to P_2$ | $R$ free, $P_1$ done, $P_2$ empty |
| $CND_6$ | $P_1$ process | $P_2 \to C_2$ | $R$ free, $P_2$ done |
| $CND_7$ | $C_2$ out | $C_1 \to P_1$ | $R$ free, $P_1$ empty |

example, for the single-cluster tool in Fig. 1(a), $S_1(t)$ indicates the status of robot $R_1$, and $S_2(t)$ and $S_3(t)$ indicate the status of process modules $P_1$ and $P_2$ at time $t$, respectively. Tables III and IV show the assigned values of state variables to completely capture the status of $R_1$, $P_1$, and $P_2$, respectively. In the decision-move-done model, action ACT is triggered by the *decision* vertex if the corresponding condition is satisfied. The *move* vertex executes the action by changing the state variables and setting the action status. After the time specified by $t_{\text{ACT}}$, the *done* vertex marks the action as finished and updates the state variables.

With state variable $\mathbf{S}$ and action index variable ACT, the following steps can be employed to automatically simplify the event graph model in Fig. 3(a) into a decision-move-done cycle

TABLE II
ACTION LABELS FOR THE EXAMPLE TOOL

| $ACT_i$ | **Action** | Nodes in Fig. 3(a) | | **Time** $t_i$ (s) |
|---|---|---|---|---|
| | | Move | Done | |
| 1 | $C_1 \to R$ | $C_1 \to P_1$ (pick) | $C_1 \to P_1$ | 5 |
| 2 | $R \to P_1$ | $C_1 \to P_1$ (place) | $C_1 \to P_1$ | 5 |
| 3 | $P_1$ proc. | $P_1$ process | $P_1$ done | 30 |
| 4 | $P_1 \to R$ | $P_1 \to P_2$ (pick) | $P_1 \to P_2$ | 5 |
| 5 | $R \to P_2$ | $P_1 \to P_2$ (place) | $P_1 \to P_2$ | 5 |
| 6 | $P_2$ proc. | $P_2$ process | $P_2$ done | 35 |
| 7 | $P_2 \to R$ | $P_2 \to C_2$ (pick) | $P_2 \to C_2$ | 5 |
| 8 | $R \to C_2$ | $P_2 \to C_2$ (place) | $C_2$ out | 5 |

TABLE III
STATE VARIABLE $\mathbf{S}$ FOR ROBOT ACTION IN THE EXAMPLE TOOL

| **Values** | $-1$ | 0 | 1 | $\cdots$ | $i$ | $\cdots$ |
|---|---|---|---|---|---|---|
| $S_1$ | In motion | Free | Ready for $ACT_1$ | $\cdots$ | Ready for $ACT_i$ | $\cdots$ |

TABLE IV
STATE VARIABLE $\mathbf{S}$ FOR PROCESS MODULE ACTION IN THE EXAMPLE TOOL

| **Values** | $-1$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| $S_2$ or $S_3$ | In motion | Free | Ready for proc. | Proc. | Proc. done |

in Fig. 3(b). Such a construction process is described in Algorithm 1. In Algorithm 1, decision logic mapping $f$ determines the state values at which the action ACT$_i$ can execute (as shown in Table V, "$*$" stands for any arbitrary values). The decision logic can be represented by the node "*decision*" in event graph. State updating variable $\Delta\mathbf{S}(\text{ACT}_i)$ is defined as the state change due to ACT$_i$ (as shown in Table V). The mapping $g$ determines the state status change after ACT$_i$ finishes (as shown in Table V, "$I$" stands for no change in update.)

```
Algorithm 1: Generating Decision-Move-Done event graph model
  input : Cluster tool configuration and wafer flow
  output: Decision-Move-Done event graph model
  Define S and $ACT$
  for $ACT_i$, $i = 1, \cdots, m$ do
      Create two nodes: a start event "Move" and an end event "Done"
      (as shown in Table 2).
      Determine S at which $ACT_i$ is executable by mapping $f : ACT \to \mathbf{S}$
      Determine the "Move" node update: $\mathbf{S} + \Delta\mathbf{S}(ACT_i) \to \mathbf{S}$
      Determine the "Done" node update by mapping $g(\mathbf{S}, ACT) : \mathbf{S} \times ACT \to \mathbf{S}$
```

The advantages of the *decision-move-done* event graph model are twofold. First, all actions are abstracted and represented by the same motion cycle: decision, move, and done. Thus, the amount of vertices in the model is significantly reduced to three and the transition conditions for all actions can be mathematically modeled into mappings $f$, $g$, and state changes $\Delta\mathbf{S}$. This could unify and generalize such an event graph model for all actions of a cluster tool. Second, we can model the dynamical systems as a simple graph and the use of the graphic algorithms could provide advantages for searching and scheduling analysis.

## III. SEARCHING FOR FEASIBLE ACTION SEQUENCES

We solve the scheduling problem of an $M$-cluster tool in two steps. The first step is to determine the sequence of actions, and the second step is to find the relative starting times of these actions. In this section, we discuss the first step of searching feasible action sequences.

### A. Searching Feasible Action Sequences

Using the decision-move-done event graph model discussed in the previous section, we can construct a graphic representation of the dynamical systems and search algorithms can be built on the event graph models as described in Algorithm 2. Since searching for action orders is of interest, zero execution time for all actions can be assumed.[1] The conditions and restrictions to build an initial state will be discussed in Section III-B.

If all possible sequences are visited, it is an NP-hard problem with computation complexity $O(m!)$, where $m$ is the total number of actions. However, if there is an infeasible action in a sequence, the corresponding sequence will fail. Therefore, the algorithm does not step into the rest of the actions in this sequence. As a result, the total amount of searches is significantly reduced. Moreover, the nature of a multicluster system enables the actions to be divided into groups and grouping methods can be applied to reduce the searching process complexity. The grouping methods will be further discussed in Section IV-C.

```
Algorithm 2: Searching feasible action sequences
  input : State information (S, ACT, decision logics, and causality conditions)
  output: All feasible sequences Seq
  Initialization state $\mathbf{S}_0$
  $\mathbf{Seq}_0 \leftarrow \emptyset$; $\mathbf{V} \leftarrow \{\mathbf{ACTs}\}$
  $k \leftarrow 0$; $m \leftarrow \dim(\mathbf{V})$
  Search(0, $\mathbf{Seq}_0$, $\mathbf{V}$, $\mathbf{S}_0$)

  function Search($k$, Seq, V, S)
  if $k = m$ then Record Seq
  for $x \in \mathbf{V}$ do
      if $x$ is feasible according to the decision mapping then
          $\mathbf{V}' \leftarrow \mathbf{V} - \{x\}$
          $\mathbf{Seq}' \leftarrow \{\mathbf{Seq}, x\}$
          $\mathbf{S}' \leftarrow$ Update S as $x$ proceeds $Move$ and $Done$
          Search($k+1$, $\mathbf{Seq}'$, $\mathbf{V}'$, $\mathbf{S}'$)
```

### B. Initial State Value $\mathbf{S}_0$

The searching results of Algorithm 2 depend on the choices of initial state value $\mathbf{S}_0$. Different initial state values may lead to different feasible action sequences. If an optimal schedule is expected to be found, initial state value $\mathbf{S}_0$ should be *one* of the snapshots of such an optimal schedule. Therefore, we have to choose the initial state value $\mathbf{S}_0$ such that optimal schedules can be found by Algorithm 2. In this section, we discuss how to choose the initial state value $\mathbf{S}_0$ such that Algorithm 2 will always succeed in finding optimal sequences.

Since the actual processing times are not considered in the algorithm, we can reduce the variation of the initial state by

[1]The use of zero action time could result in a larger set of feasible sequences during the searching stage. However, the network analysis discussed in the next section will optimize these feasible sequence sets with action time.

TABLE V
DECISION LOGIC MAPPING $f$ ("*DECISION*"), STATE CHANGE $\Delta\mathbf{S}$ ("*MOVE*"), AND STATE UPDATE MAPPING $g$ ("*DONE*") FOR THE EXAMPLE TOOL

| $ACT_i$ | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Decision | $S_1$ | 0 | 2 | $*$ | 0 | 5 | $*$ | 0 | 8 |
| mapping | $S_2$ | $*$ | 0 | 1 | 3 | $*$ | $*$ | $*$ | $*$ |
| $f : ACT \rightarrow \mathbf{S}$ | $S_3$ | $*$ | $*$ | $*$ | 0 | 0 | 1 | 3 | $*$ |
| | $S_1$ | $-1$ | $-3$ | 0 | $-1$ | $-6$ | 0 | $-1$ | $-9$ |
| State change | $S_2$ | 0 | $-1$ | $+1$ | $-4$ | 0 | 0 | 0 | 0 |
| $\Delta\mathbf{S}$ | $S_3$ | 0 | 0 | 0 | 0 | $-1$ | $+1$ | $-3$ | 0 |
| State update | $S_1$ | 2 | 0 | $I$ | 5 | 0 | $I$ | 8 | 0 |
| mapping | $S_2$ | $I$ | 1 | 3 | 0 | $I$ | $I$ | $I$ | $I$ |
| $g : \mathbf{S} \times ACT \rightarrow \mathbf{S}$ | $S_3$ | $I$ | $I$ | $I$ | $I$ | 1 | 3 | 0 | $I$ |

assuming all actions start in the done state. The done state of an action includes: 1) the process module finishes the process and the processed wafer is ready to pick up; 2) the robot finishes placing a wafer and the blade is ready to pick other wafers; and 3) the robot finishes picking a wafer and the blade is occupied by a wafer. From such a state, the key configuration that could lead to an efficient scheduling sequence is the number of the empty wafer slots within the cluster tool. If $\mathbf{S}_0$ is chosen such that there are a few empty slots, then it could lead to deadlock. On the other hand, if $\mathbf{S}_0$ is chosen in a way that there are plenty of empty slots, then the resulting searching sequences will not be optimal since the resources are not fully utilized to process wafers.

We define "empty slot" $E_i$ for the single cluster $\mathbb{C}_i$ as the collection of all empty spaces among process modules $P_i$, robots $R_i$, and buffer modules $B_i$ that can hold wafers within $\mathbb{C}_i$

$$E_i = P_i^E \bigcup R_i^E \bigcup B_i^E, \quad i = 1, 2, \cdots, M \tag{1}$$

where $P_i^E$ is the set of empty process modules in $\mathbb{C}_i$, $R_i^E$ is the set of empty blades of robot $R_i$, and $B_i^E$ is the empty buffer space in $B_i$. The $j$th process module in $\mathbb{C}_i$ is denoted as $P_{ij}$. Note that $E_i$ does not include the wafer slots in the buffer module $B_{i+1}$ between clusters $\mathbb{C}_i$ and $\mathbb{C}_{i+1}$. Denote the wafer capacity of $E_i$ as $d(E_i)$. The selection rule of the initial state condition $\mathbf{S}_0$ can be summarized as follows.

Initial state value $\mathbf{S}_0$ selection rules

1) $d(E_i) = 2$.
2) Each process module $P_{ij}$ of cluster $\mathbb{C}_i$ should have one finished wafer, and the wafer is ready for picking up by robot $R_i$.
3) If $d(B_i) = 2$, then one wafer should allocate at one empty slot of $B_i$, and this wafer should be ready to be picked up by $R_i$. The other slot of $B_i$ should be empty. If $d(B_i) = 1$, then the buffer module $B_i$ must be kept empty.
4) If $R_i$ is double-blade, then one blade should hold one wafer, and this wafer should be ready to place into $B_i$. The other blade of $R_i$ should be empty. If $R_i$ is single-blade, then it must be allocated empty and ready to pick up a wafer.

To understand the above selection rules, we first claim the following fact.

*Proposition 1:* $d(E_i) = 2$ (if $R_i$ is single-blade) or $d(E_i) = 1$ (if $R_i$ is double-blade), $i = 1, \cdots, M$, is a minimum empty slot requirement for a deadlock-free schedule of an $M$-cluster tool at steady state.

*Proof:* See the Appendix. ■

From Proposition 1, we need to choose the initial state $\mathbf{S}_0$ such that the schedule for each single-robot cluster is always deadlock free. For cluster $\mathbb{C}_i$, if $R_i$ is single-blade, there must be at least two empty slots; if $R_i$ is double-blade, one empty slot is the minimum deadlock-free requirement. Moreover, from the proof of Proposition 1, if $R_i$ is double-blade, having two empty slots $(d(E_i) = 2)$ in cluster $\mathbb{C}_i$ yields a shorter or equal cycle time compared with one empty slot. Therefore, two empty slots (e.g., $d(E_i) = 2$) is the minimum deadlock-free and the most efficient requirement for cluster $\mathbb{C}_i$.

Proposition 1 implies that if a cluster has more than two empty slots, it must have deadlock-free schedules. We can extend the results to show that $d(E_i) = 2$ could lead to the shortest cycle time. To prove this fact, we consider whether more than two empty slots $(d(E_i) \geq 3)$ can yield a shorter cycle time. Denote the cycle time of cluster $\mathbb{C}_i$ under $d(E_i) \geq 3$ as $\mathbf{FP}_i'$ and the cycle time under $d(E_i) = 2$ as $\mathbf{FP}_i''$. We can observe that $\mathbf{FP}_i' \geq \mathbf{FP}_i''$, namely, the cycle time under $d(E_i) \geq 3$ is at most equal to or longer than that under $d(E_i) = 2$. This fact can be obtained from the same idea as in the proof of Proposition 1. In fact, more than two empty slots within a cluster will not yield the optimal schedules and the maximum throughput. For a single cluster, the optimal schedules are achieved with two empty slots [4], [5]. Having more empty slots may reduce the cluster tool productivity since it does not fully utilize the resources.

Next, considerations must be taken as to where to allocate these two slots within a cluster. Items 2–4 in the $\mathbf{S}_0$ selection rules define one choice. Selection rule 2 implies that one wafer should always be allocated to each process module. Rules 3 and 4 distribute the empty slot by the configurations of the buffer module $B_i$ and robot $R_i$. It is ensured that each cluster has one empty robot blade and one empty output connector. Thus, each robot can start at least one action immediately, which avoids wasting robot transferring time or an immediate deadlock.

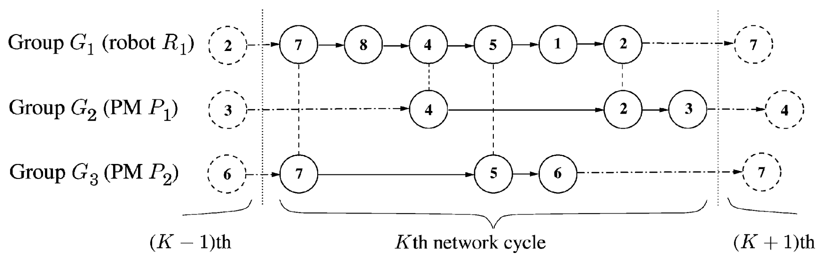

Fig. 4. Network for the example tool shown in Fig. 1(a).

## IV. Scheduling Analysis

The feasible action sequences can be modeled into an unidirected network with actions as nodes and action relationships as edges. The network can further be applied to calculate the cycle time attached to the sequence. A similar approach has been discussed in [17] for a specific simple single-cluster tool. Here, we conduct a comprehensive discussion about network modeling and sensitivity analysis to handle more general and complex multicluster tools.

### A. Network Representation of Sequence

Given a feasible sequence, we design the network diagram in a way that each horizontal line (called group) represents actions in one robot or one process module (as shown in Fig. 4). For an $M$-cluster tool with $N$ total process modules (including buffer modules), there are $N+M$ groups, $G_j, j=1,\cdots,(N+M)$, each of which is constructed by associated actions in the order as they appear in the feasible sequence. Different representations of the same action (e.g., $\mathrm{ACT}_7$, $\mathrm{ACT}_4$, $\mathrm{ACT}_5$, and $\mathrm{ACT}_2$) can appear in different groups but are restricted to be vertically aligned (as shown by dotted lines). At steady state, the network group is arranged so that each group is repeated with unchanged action orders and relative timings. For example, Fig. 4 is the network diagram (we focus on $K$th network cycle) for the example of the single-cluster tool discussed in Section II at steady state. There are three groups within such a cluster tool: robot $R$ (as $G_1$) and those ($G_2$ and $G_3$) for the action sequences of process modules $P_1$ and $P_2$, respectively. The action sequence that we use to build such a diagram is $\{\mathrm{ACT}_7, \mathrm{ACT}_8, \mathrm{ACT}_4, \mathrm{ACT}_5, \mathrm{ACT}_1, \mathrm{ACT}_6, \mathrm{ACT}_2, \mathrm{ACT}_3\}$. In a network, the actions in one group follow the same order as they show up in the feasible sequence. However, actions belonging to different groups do not necessarily have to follow the order of the feasible sequence. For example, $\mathrm{ACT}_3$ and $\mathrm{ACT}_6$ are two actions without direct relationship. If swapping them in a feasible sequence makes an other feasible sequence, the new sequence will lead to the same network. For example, action sequences

$$\{\mathrm{ACT}_7,\mathrm{ACT}_8,\mathrm{ACT}_4,\mathrm{ACT}_5,\mathrm{ACT}_1,\mathrm{ACT}_2,\mathrm{ACT}_6,\mathrm{ACT}_3\}$$
or
$$\{\mathrm{ACT}_7,\mathrm{ACT}_8,\mathrm{ACT}_4,\mathrm{ACT}_5,\mathrm{ACT}_1,\mathrm{ACT}_2,\mathrm{ACT}_3,\mathrm{ACT}_6\}$$
or
$$\{\mathrm{ACT}_7,\mathrm{ACT}_8,\mathrm{ACT}_4,\mathrm{ACT}_5,\mathrm{ACT}_6,\mathrm{ACT}_1,\mathrm{ACT}_2,\mathrm{ACT}_3\}$$

could lead to the same network. Therefore, multiple feasible sequences could result in the same network.

For each action $\mathrm{ACT}_i, i=1,\cdots,m$, we can define the direct preceding actions $\mathbf{ACT}_i^{\mathrm{pre}}$ as a set of actions that are directly connected with and precede $\mathrm{ACT}_i$ in the network, and the direct subsequent actions $\mathbf{ACT}_i^{\mathrm{sub}}$ as a set of actions that are directly connected with and follow $\mathrm{ACT}_i$ in the network. For example, for $\mathrm{ACT}_5$ in the example cluster tool, the preceding action set is $\mathbf{ACT}_5^{\mathrm{pre}} = \{\mathrm{ACT}_4,\ \mathrm{ACT}_7\}$ and the subsequent action set is $\mathbf{ACT}_5^{\mathrm{sub}} = \{\mathrm{ACT}_1,\ \mathrm{ACT}_6\}$ (Fig. 4). If an action is in the front of all groups that it belongs to within one network cycle, its preceding action set is defined as an empty set $\emptyset$, e.g., $\mathrm{ACT}_7$ in the example. Similarly, an action at the end of all groups within one network cycle will have the empty direct subsequent set, e.g., $\mathrm{ACT}_3$ and $\mathrm{ACT}_6$ in Fig. 4.

### B. Scheduling Using Extended CPM

Given a network, the critical path method (CPM) [21] can be utilized to calculate the fundamental period $\mathbf{FP}$. In the following, we first describe how the basic CPM idea can be applied to calculate the minimum cycle time of an isolated (or static) network cycle (e.g., only $K$th cycle in Fig. 4). Then, we extend the basic CPM to compute $\mathbf{FP}$ of the connected (or dynamic) networks (e.g., combined $K$th network cycle with neighboring cycles such as $(K-1)$th and $(K+1)$th cycles in Fig. 4).

In one isolated cycle period, every action runs exactly once in the order as in the network (e.g., $K$th cycle in Fig. 4). If the first action starts at time zero and the last action ends at time $\mathbf{FP}^0$, then the cycle time of the isolated network cycle is $\mathbf{FP}^0$. For $\mathrm{ACT}_i$ in the network, denote the earliest starting time as $T_i^{\mathrm{EST}}$ and the latest starting time as $T_i^{\mathrm{LST}}$.

Given that all action starts no earlier than time zero, we can first set $T_l^{\mathrm{EST}}=0$ for any action $\mathrm{ACT}_l$ such that $\mathbf{ACT}_l^{\mathrm{pre}}=\emptyset$. By iteratively going forward along the network, we can calculate $T_i^{\mathrm{EST}}$ in a forward calculation as

$$\text{Forward calculation}: \quad T_i^{\mathrm{EST}} = \max_{ACT_j \in \mathbf{ACT}_i^{\mathrm{pre}}} \{T_j^{\mathrm{EST}} + t_j\} \quad \text{for all } i=1,\cdots,m \tag{2}$$

where $t_j$ is the processing time of action $\mathrm{ACT}_j$. Once this forward calculation reaches the end of the network, we obtain $T_i^{\mathrm{EST}}$ for all $\mathrm{ACT}_i$. Then, the cycle time $\mathbf{FP}^0$ can be obtained as

$$\mathbf{FP}^0 = \max_i \{T_i^{\mathrm{EST}} + t_i\}. \tag{3}$$

Next, we can then set $T_l^{\mathrm{LST}} = \mathbf{FP}^0 - t_l$ for any action $\mathrm{ACT}_l$ such that $\mathbf{ACT}_l^{\mathrm{sub}} = \emptyset$. Then, $T_i^{\mathrm{LST}}$ for all actions $\mathrm{ACT}_i$ can be obtained using the following backward calculation:

$$\text{Backward calculation}: \quad T_i^{\mathrm{LST}} = \min_{ACT_j \in \mathbf{ACT}_i^{\mathrm{sub}}} \{T_j^{\mathrm{LST}}\} - t_i \quad \text{for all } i = 1, \cdots, m. \tag{4}$$

With $T_i^{\mathrm{EST}}$ and $T_i^{\mathrm{LST}}$, we can define the slack value $T_i^{\mathrm{SLK}}$ for action $\mathrm{ACT}_i$ as

$$T_i^{\mathrm{SLK}} = T_i^{\mathrm{LST}} - T_i^{\mathrm{EST}}, \quad i = 1, \cdots, m. \tag{5}$$

It is easy to observe that the slack value $T_i^{\mathrm{SLK}}$ implies the flexibility of action time $t_i$ of $\mathrm{ACT}_i$ within $[0, \mathbf{FP}^0]$. Namely, the processing time of $\mathrm{ACT}_i$ can be prolonged to $t_i + T_i^{\mathrm{SLK}}$ without increasing the cycle time $\mathbf{FP}^0$. If an action has a zero slack value, then a small increase in the action time may cause a comparable time increase on cycle time. By [21], there exists a connected path in the network that all actions in the path have zero slack values. Such a path is called the critical path of the network. The method to identify the critical path (e.g., forward and backward calculations) is called critical path method.

The calculation described above only applies on the isolated network cycle case. At steady state when cycles are repeated continuously, the above calculated cycle time might not be the minimal since actions from different groups and different cycles could run parallelly. For example, for the network shown in Fig. 4, action $\mathrm{ACT}_3$ in one cycle could start at the same moment when $\mathrm{ACT}_7$ of the next cycle starts. Then, the steady state $\mathbf{FP}$ might be smaller than $\mathbf{FP}^0$. To overcome such a problem caused by multigroup structures in a network, we could extend the basic CPM. For each group $G_j$, we calculate minimal cycle time $\mathbf{FP}_{G_j}$. Then, the maximum of $\mathbf{FP}_{G_j}$ should be the steady-state cycle time of the connected network.

The basic idea of the extended CPM is to find a steady-state cycle in which each group $G_j$ reaches the minimum time span $\mathbf{FP}_{G_j}$. Assume in such a cycle, actions in group $G_j$ start as early as $T_{G_j}^{\mathrm{GST}}$ and end as late as $T_{G_j}^{\mathrm{GET}}$. Then, $\mathbf{FP}_{G_j}$ is calculated as

$$\mathbf{FP}_{G_j} = T_{G_j}^{\mathrm{GET}} - T_{G_j}^{\mathrm{GET}}, \quad j = 1, \cdots, N + M. \tag{6}$$

In each group $G_j$, denote the first action and the last action as $\mathrm{ACT}_{s(G_j)}$ and $\mathrm{ACT}_{e(G_j)}$, respectively. To find $T_{G_j}^{\mathrm{GST}}$ and $T_{G_j}^{\mathrm{GET}}$, we need to calculate $T_{s(G_j)}^{\mathrm{LST}}$ and $T_{e(G_j)}^{\mathrm{EST}}$ using the CPM concept.

It is noted that if all $T_{G_j}^{\mathrm{GST}}$ are known, then a unique set of $T_i^{\mathrm{EST}}$, $i = 1, \cdots, m$, can be determined. To implement it, we can find all $G_j$ with $\mathbf{ACT}_{s(G_j)}^{\mathrm{sub}} = \emptyset$ and then set $T_{s(G_j)}^{\mathrm{EST}} = T_{G_j}^{\mathrm{GST}}$. Applying one forward calculation with an additional condition[2] that any action in $G_j$ cannot start earlier than $T_{G_j}^{\mathrm{GST}}$, we can calculate all $T_i^{\mathrm{EST}}$. Similarly, if $T_{G_j}^{\mathrm{GET}}$ are all known, we can find a unique set of $T_i^{\mathrm{LST}}$. With such observations, we can summarize the $\mathbf{FP}$ calculation as in Algorithm 3. The algorithm is also based on the results given in Proposition 2. Since the groups within a network are connected (due to the network construction), the algorithm could terminate eventually within at most $((N + M)/2)$ iterations.[3]

*Proposition 2:* The fundamental period $\mathbf{FP}$ of an $M$-cluster tool system can be calculated as

$$\mathbf{FP} = \max_{1 \le j \le (N+M)} \{\mathbf{FP}_{G_j}\}$$

and the bottlenecks of the system are groups with $\mathbf{FP}_{G_k} = \mathbf{FP}$, where $\mathbf{FP}_{G_j}$ can be calculated by (6).

*Proof:* It is first proven that $\mathbf{FP} \ge \mathbf{FP}_{G_j}$ for any group $G_j$. Then, we show that there exists a feasible schedule such that $\mathbf{FP} = \max_{1 \le j \le (N+M)} \{\mathbf{FP}_{G_j}\}$.

In the steady state, the network cycle repeats (Fig. 4). Assume that the cycles are indexed by $K$. For the $K$th network cycle, denote the $i$th action within the network as $\mathrm{ACT}_i^K$. For network cycle $K$, let action $\mathrm{ACT}_{e(G_j)}^K$ end at time $T_{G_j}^{\mathrm{GET}}$. The earliest time $\mathrm{ACT}_{s(G_j)}^{K+1}$ can start is also $T_{G_j}^{\mathrm{GET}}$. On the other hand, the latest time $\mathrm{ACT}_{s(G_j)}^K$ can start is $T_{G_j}^{\mathrm{GST}}$. Therefore, the minimum time difference between $\mathrm{ACT}_{s(G_j)}^K$ and $\mathrm{ACT}_{s(G_j)}^{K+1}$ is then $\mathbf{FP}_{G_j} = T_{G_j}^{\mathrm{GET}} - T_{G_j}^{\mathrm{GST}}$. So, $\mathbf{FP} \ge \mathbf{FP}_{G_j}$.

Given $\mathbf{FP} = \max_{1 \le j \le (N+M)} \{\mathbf{FP}_{G_j}\}$, denote the following schedule as $\pi$. Let $\mathrm{ACT}_i^K$, $i = 1, \cdots, m$, $K \in \mathbb{N}$ start at time $K \times \mathbf{FP} + T_i^{\mathrm{EST}}$. Schedule $\pi$ is feasible since no action overlap exists inside or between cycles by such a construction. The steady-state cycle time of $\pi$ is $\mathbf{FP}$ because, for any action, the difference of starting times of a same action in any two consecutive cycles is $\mathbf{FP}$. ■

**Algorithm 3**: Computing the fundamental period

```
input : Network and action processing time
output: The fundamental period FP
$\mathcal{G}_A = \{G_1, \cdots, G_{N+M}\}$; $\mathcal{G}_S = \{G_1\}$; $\mathcal{G}_E = \emptyset$; $T_{G_1}^{GST} = 0$; $T_{G_1}^{GET} = +\infty$
for $l = 2$ to $N + M$ do $T_{G_l}^{GST} = -\infty$; $T_{G_l}^{GET} = +\infty$
while $\mathcal{G}_S \neq \mathcal{G}_A$ do
    Do Forward calculation (Eq.(2))
    $\mathcal{G}_E \leftarrow \mathcal{G}_S$
    for $G_k \notin \mathcal{G}_S$ do
        if $G_k$ shares any action in $\mathcal{G}_S$ then
            $\mathcal{G}_E \leftarrow \mathcal{G}_S + \{G_k\}$
    Do Backward calculation (Eq.(4))
    $\mathcal{G}_S \leftarrow \mathcal{G}_E$
    for $G_k \notin \mathcal{G}_E$ do
        if $G_k$ shares any action in $\mathcal{G}_E$ then
            $\mathcal{G}_S \leftarrow \mathcal{G}_E + \{G_k\}$
for $l = 1$ to $N + M$ do $\mathbf{FP}_{G_l} = T_{G_l}^{GET} - T_{G_l}^{GST}$
$\mathbf{FP} = \max_{1 \le l \le (N+M)} \{\mathbf{FP}_{G_l}\}$
```

## *C. Methods to Reduce Complexity of Searching Algorithm 2*

As noted in Section IV-A, the network, instead of the sequences, determines the scheduling and fundamental period of

[2]This constraint is needed to restrict the starting time calculation for actions that belong to several different groups, such as $\mathrm{ACT}_4$ (in both $G_1$ and $G_2$) in the example shown in Fig. 4.

[3]Since each iteration runs one forward and one backward calculations, one iteration will expand two groups and the total number of iterations is only a half of the total group numbers.

the cluster tool. In Algorithm 2, once a sequence of a network is found, other sequences that lead to the same network are considered redundant from the viewpoint of network. Therefore, it is desirable to skip searching these redundant sequences. The grouping method presented in this section can be utilized to take advantage of the network features and to reduce the searching complexity.

Since the network topology is constructed by an interaction of groups, the network is determined once internal action sequences of all groups are known. We can thus reduce the complexity of the searching algorithm in such a way that it only seeks for possible action permutations that belong to the same network group and ignore the action order between different groups. For example, in the cluster example in Section IV-A, the order between $\text{ACT}_5$ and $\text{ACT}_1$ is important because it affects the robot operation. However, the order between $\text{ACT}_2$ and $\text{ACT}_6$ is not important in the search algorithm since they belong to different groups ($G_1$ and $G_3$, respectively).

Search Algorithm 2 builds a recursion tree [22]. In one iteration of the $n$th level of the recursive tree, the algorithm first seeks for all independently executable actions. Assuming the set of such actions as $\mathbf{E}$, each action in $\mathbf{E}$ corresponds to one subtree in the way that the algorithm steps into the $(n+1)$th level of search iteration immediately after executing the action. Since one search iteration only executes one action, the algorithm needs to step into the $m$th level of the recursion tree to obtain a feasible sequence. The following two grouping methods allow one subtree to include several actions such that one iteration step can simultaneously execute a few actions before stepping into the next searching iteration. Since the total amount of actions is fixed, the levels required for the recursion tree will be reduced and therefore the algorithm complexity is reduced.

The first grouping method is to continuously find several actions from one robot action group and execute these actions in one search iteration. Once the algorithm steps into the $n$th search iteration, actions in $\mathbf{E}$ can be picked one by one to start the subtree. If $\text{ACT}_i \in \mathbf{E}$ is chosen and $\text{ACT}_i \in G_j$, a robot action group, denoting $\mathbf{E}_S = \mathbf{E} \cap \overline{G}_j$, where $\overline{G}_j$ is a set of actions *not* in $G_j$, we do the following recursive steps. In those steps, all actions assigned to $\text{ACT}_{\text{ex}}$ are executed in the search iteration. For the example in Section IV-A, $\text{ACT}_7$ and $\text{ACT}_8$ are actions that can be executed consecutively.

```
Grouping method 1
for each $ACT_i \in \mathbf{E}$ at the nth level do
    $ACT_k = ACT_i$
    repeat
        $ACT_{ex} = ACT_k$
        Execute $ACT_{ex}$
    until only one action $ACT_k \in G_j$ becomes executable
          and all actions in $\mathbf{E}_S$ are still executable
    Step into $(n+1)$th level in the recursive tree
```

The second grouping method is to run several actions simultaneously from various groups in one search iteration. In one $n$th level search iteration, we first find the set of independently executable actions $\mathbf{E}$. We then find all subsets $\mathbf{E}_G \subseteq \mathbf{E}$ that meet the following conditions: 1) each action in $\mathbf{E}_G$ belongs to a unique group; 2) all actions in $\mathbf{E}_G$ can be executed at the same moment; and 3) once all actions in $\mathbf{E}_G$ are executed, no other action in $\mathbf{E}$ is still executable. We then execute all actions in each $\mathbf{E}_G$ and step into the next level search iterations. By doing so, the recursion tree has one subtree corresponding to each $\mathbf{E}_G$. As the example in Section IV-A, if $\text{ACT}_5$ is executed in the previous search iteration and only $\text{ACT}_6$, $\text{ACT}_1$, $\text{ACT}_2$, $\text{ACT}_3$ are left to be scheduled, $\mathbf{E}_G = \{\text{ACT}_6, \text{ACT}_1\}$ because conditions 1)–3) are satisfied for those two actions. Then, $\text{ACT}_6$ and $\text{ACT}_1$ will be executed together in the current search iteration by this grouping strategy.

Although based on different features of the network topology, the previous two grouping methods can be integrated in Algorithm 2 simultaneously. Using the grouping method, the amount of redundant searches can be reduced significantly. By the example in Section V, we further notice that once the integrated network grouping method is applied, the recursive search method returns the same amount of sequences as that of the amount of derived networks, which implies that the algorithm reaches 100% efficiency in eliminating redundant sequences.

### D. Sensitivity Analysis

The sensitivity analysis of cluster tools is to study the $\mathbf{FP}$ variations due to changes of processing time of an action. This is important in practice because: 1) processing time sometimes is not constant (we will show such an example in Section V) and 2) there exist random process variations which could lead to processing time changes. The sensitivity analysis could guide us to choose a robust schedule under such variations.

Recall that the slack value for $\text{ACT}_i$ is the difference between $T_i^{\text{EST}}$ and $T_i^{\text{LST}}$ [see (5)]. At steady state, the actual earliest start time or latest end time of an action is much more flexible since there is no fixed earliest start time or latest end time within one cycle. We consider the following methodology to handle such a potential issue.

Denote the feasible schedule that results in $\mathbf{FP}$ in Proposition 2 and Algorithm 3 as $\pi$. Without loss of generality, we consider network cycle $K$ of $\pi$. It can find at least one group $G_l$ that $\mathbf{ACT}_{s(G_l)}^{\text{pre}} = \emptyset$. Constrained by the preceeding network cycle $K-1$, the earliest start time for $\text{ACT}_{s(G_l)}^{K}$ is the earliest time that $\text{ACT}_{e(G_l)}^{K-1}$ is finished, which is equal to $T_{G_l}^{\text{GET}} - \mathbf{FP}$. On the other hand, by the construction of $\pi$, the latest start time for $\text{ACT}_{s(G_l)}^{K}$ is $T_{G_l}^{\text{GST}}$. Thus, we find a temporary slack value for $\text{ACT}_{s(G_l)}$ as $T_{G_l}^{\text{GST}} + \mathbf{FP} - T_{G_l}^{\text{GET}}$. We denote the slack value constrained by the preceeding network cycle as $T_i^{\text{PreSLK}}$ for action $\text{ACT}_i$. To calculate such a slack value for other actions, we let ${}^{\text{pre}}T_{G_j}^{\text{GST}} = T_{G_j}^{\text{GET}} - \mathbf{FP}$ for each group $G_j$, $j = 1, \cdots, (N+M)$. We can then do one forward calculation based on ${}^{\text{pre}}T_{G_j}^{\text{GST}}$ and find the earliest start times as ${}^{\text{pre}}T_i^{\text{EST}}$. Thus

$$T_i^{\text{PreSLK}} = T_i^{\text{LST}} - {}^{\text{pre}}T_i^{\text{EST}}, \quad i = 1, \cdots, m. \qquad (7)$$

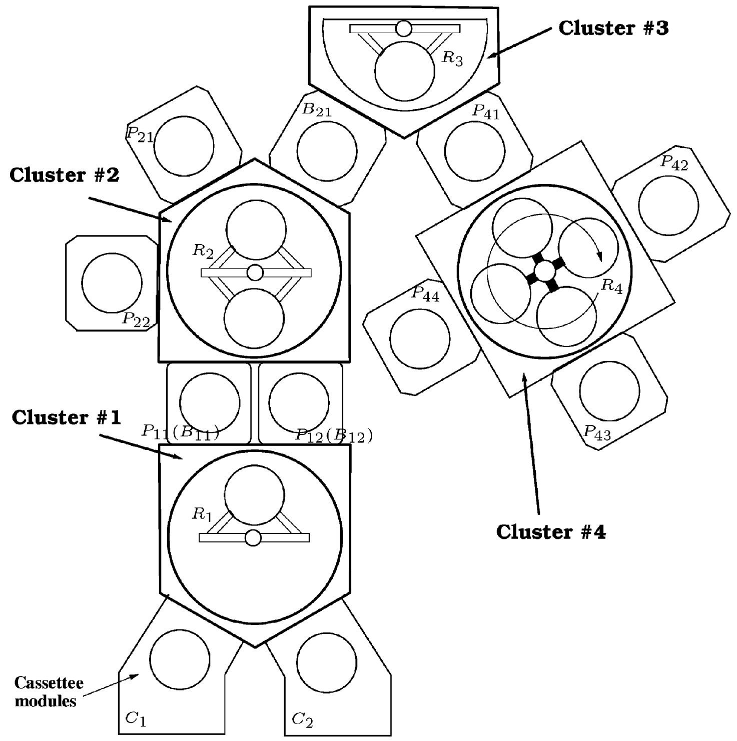

Fig. 5. Simplified layout schematic of the CMP multicluster polisher.

Similarly, the slack value could also be constrained by the succeeding cycle and we denote such a slack time as $T_i^{\text{SucSLK}}$ for action ACT$_i$. For group $G_j$, we can obtain ${}^{\text{Suc}}T_{G_j}^{\text{GET}} = T_{G_j}^{\text{GST}} + \mathbf{FP}$. One backward calculation can be carried out to find the latest start time for action ACT$_i$ as ${}^{\text{sub}}T_i^{\text{LST}}$. Then, we have

$$T_i^{\text{SucSLK}} = {}^{\text{Suc}}T_i^{\text{LST}} - T_i^{\text{EST}}, \quad i = 1, \cdots, m. \tag{8}$$

The slack values $T_i^{\text{PreSLK}}$ and $T_i^{\text{SucSLK}}$ can be viewed as indications of the geometric ACT$_i$ location within the group. If action ACT$_i \in G_j$ has $T_i^{\text{PreSLK}} < T_i^{\text{SucSLK}}$, it is most likely that ACT$_i$ is located close to the end of $G_j$. When action ACT$_i$ takes a long time to finish, there is more flexiblity in $\pi$ to move $T_{G_j}^{\text{GET}}$ later compared with moving $T_{G_j}^{\text{GST}}$ earlier without changing $\mathbf{FP}$. If ACT$_i$ is the only action with varying processing time, we can calculate its overall slack value as

$$T_i^{\text{SLK}} = \max\left\{T_i^{\text{PreSLK}}, T_i^{\text{SucSLK}}\right\}, \quad i = 1, \cdots, m. \tag{9}$$

It is common that actions in different optimal scheduling sequences may have totally different slack values. If there exists one action ACT$_i$ with random processing time $t_i$ within a cluster tool, we can find the optimal schedules using the mean value $E(t_i)$ of ACT$_i$. Then, among these optimal sequences, the sequence with the largest slack value $T_i^{\text{SLK}}$ of action ACT$_i$ could reach the best optimality because this optimal sequence could result the minimal $\mathbf{FP}$ variation. Similarly, $T_i^{\text{SLK}}$ can also be considered the maximum residency time that an action could encounter without increasing $\mathbf{FP}$. For example, if ACT$_i$ is a chamber process, then the wafer could stay in the chamber as long as $T_i^{\text{SLK}}$ after the process finishes without increasing $\mathbf{FP}$.

## V. Experiments

We have successfully applied the proposed methodology to dozens of tools at Intel Corporation. Some examples are shown in [25]. The benefits include better throughput estimation, faster what-if analysis, and optimal scheduling solutions with varying processing times and cluster tool configurations. In this section, we give a case study on how to apply the methododogy, described in this paper, to a chemical–mechanical planarization polisher.

### A. Chemical–Mechanical Planarization (CMP) Polisher

Fig. 5 shows a schematic of a CMP polisher used in semiconductor manufacturing. The chemical–mechanical planarization process is widely used to planarize the wafer surface and to enhance the photolithograph process performance. The CMP polisher can be modeled as a four-cluster tool. There are two single-blade robots $R_1$ and $R_3$, a double-blade robot $R_2$, and an indexer $R_4$. The indexer $R_4$ moves wafers simultaneously from process modules $P_{41}$ to $P_{42}$, $P_{42}$ to $P_{43}$, $P_{43}$ to $P_{44}$, and $P_{44}$ to $P_{41}$, respectively. The wafers pass through the cluster tool as in the following flow chart:

$$\begin{aligned} C_1 &\xrightarrow{R_1} P_{11}(B_{11}) \xrightarrow{R_{21}} B_{21} \xrightarrow{R_3} P_{41} \xrightarrow{R_4} P_{42} \xrightarrow{R_4} P_{43} \xrightarrow{R_4} P_{44} \\ &\xrightarrow{R_4} P_{41} \xrightarrow{R_3} B_{21} \xrightarrow{R_{22}} P_{21} \xrightarrow{R_{22}} P_{22} \xrightarrow{R_{22}} P_{12}(B_{12}) \xrightarrow{R_1} C_2. \end{aligned}$$

### B. Experiment Results

Event graph modeling of the CMP tool can be handled as follows. First, we define the state variable and action indexes. The decision-move-done event graph can be constructed for the whole cluster tool. In total, there are 15 state variables (total number of cassette, process, buffer, and transfer modules) and

TABLE VI
ACTIONS OF THE CMP POLISHER

| $ACT_i$ | **Action** | **Time** $t_i$ $(s)$ |
|---|---|---|
| 1 | $C_1 \xrightarrow{R_1} P_{11}(B_{11})$ | 15 |
| 2 | $P_{11}$ process | 10 |
| 3 | $P_{11}(B_{11}) \xrightarrow{R_{21}} B_{21}$ first half | 10 |
| 4 | $P_{11}(B_{11}) \xrightarrow{R_{21}} B_{21}$ second half | 17 |
| 5 | $B_{21} \xrightarrow{R_3} P_{41}$ | 18 |
| 6 | $P_{41} \xrightarrow{R_4} P_{42}, P_{42} \xrightarrow{R_4} P_{43}, P_{43} \xrightarrow{R_4} P_{44}, P_{44} \xrightarrow{R_4} P_{41}$ | 5 |
| 7 | $P_{42}$ processing | 60 |
| 8 | $P_{43}$ processing | 60 |
| 9 | $P_{44}$ processing | 60 |
| 10 | $P_{41} \xrightarrow{R_3} B_{21}$ | 26 |
| 11 | $B_{21} \xrightarrow{R_{21}} P_{21}$ first half | 18 |
| 12 | $B_{21} \xrightarrow{R_{21}} P_{21}$ second half | 20 |
| 13 | $P_{21}$ processing | 30 |
| 14 | $P_{21} \xrightarrow{R_{21}} P_{22}$ first half | 15 |
| 15 | $P_{21} \xrightarrow{R_{21}} P_{22}$ second half | 20 |
| 16 | $P_{22}$ processing | 30 |
| 17 | $P_{22} \xrightarrow{R_{22}} P_{12}(B_{12})$ first half | 16 |
| 18 | $P_{22} \xrightarrow{R_{22}} P_{12}(B_{12})$ second half | 20 |
| 19 | $P_{12}$ processing | 20 |
| 20 | $P_{12}(B_{12}) \xrightarrow{R_1} C_2$ | 20 |

20 actions (as shown in Table VI) for the event graph model. Indexer $R_4$ cannot hold wafers independently and thus is not considered as an independent decision state. The initial state values are chosen according to the rules discussed in Section III-B: each process module ($P_{11}$, $P_{12}$, $P_{21}$, $P_{22}$, or $P_{4i}$, $i = 1, \cdots, 4$) has one finished wafer, $R_2$ has one wafer on one of its blades, $R_1$ and $R_3$ are empty, and the buffer module $B_{21}$ is empty. We created the event graph using simulation package SIGMA [23] and the model is automatically exported into C language for searching and network analysis.

*1) Throughput Calculation:* The event graph-based modeling and algorithms in Sections III and IV are utilized and implemented in C language to find the optimal schedules. The experiments are carried out on a computer with Intel Centrino 1.3 GHz CPU and 256 M RAM. Table VII summarizes the computational results.

We first search all sequences for the given wafer flow without network analysis. Such a search algorithm takes about 219 063 s to finish. This search generates more than 1.86 billion feasible sequences (out of a total 20! $\approx 2.4 \times 10^7$ billion sequences). Among these feasible sequences, about 1.25 billion sequences give $\mathbf{FP} = 136$ s, which is the minimal cycle time. Then, we apply the network model-based grouping methods. In about 0.1 s of computing time, the network-based search algorithm generates 58 possible networks. Among them, 17 networks give the fundamental period $\mathbf{FP} = 136$ s. The resulting $\mathbf{FP} = 136$ s/wafer (namely the steady-state throughput is 26.5 wafers per hour) fits very well with the maximum throughput, 26 wafers per hour, of such a CMP polisher in an R&D fab using one of the resulting action sequences.

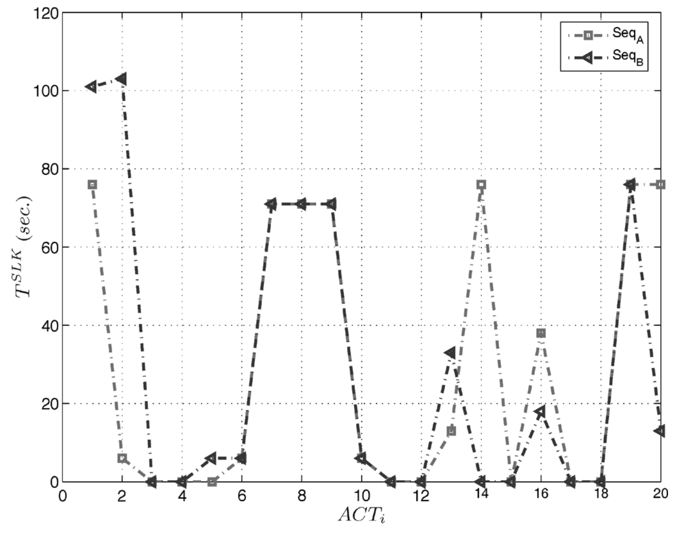

Fig. 6. Sensitivity comparison results of two optimal schedule sequences $\mathbf{Seq}_A$ and $\mathbf{Seq}_B$.

*2) Sensitivity Analysis:* To illustrate the sensitivity analysis, we choose two different sequences $\mathbf{Seq}_A$ and $\mathbf{Seq}_B$, which both produce the $\mathbf{FP} = 136$ s. Table VIII shows all ordered actions in every group of $\mathbf{Seq}_A$ and $\mathbf{Seq}_B$. Here, we use table, instead of network, as shown in Fig. 4, to simplify the network presentation. From Table VIII, we find that the two networks are different in $G_1$ (robot of Cluster 1) and $G_2$ (robot of Cluster 2), while other groups are the same. Using the slack time calculation in Section IV-D, we calculate the slack time for each action based on the nominal processing time given by Table VI. Fig. 6 shows the slack time for both sequences $\mathbf{Seq}_A$ and $\mathbf{Seq}_B$. In the following, we show two case studies of the impact of the processing time variations on the fundamental period.

Due to the incoming film thickness variations, the polishing processing time could vary to achieve a predefined target film thickness. The mechanism to automatically determine when to stop polishing is called end-point detection [24]. It is appropriate to assume that the processing time of the last polishing module $P_{44}$ ($\text{ACT}_9$) has a large variation if the end-point detection device is used. As shown in Fig. 6, the slack values for $P_{44}$ ($\text{ACT}_9$) in both action sequences are equal to 71 s. This calculation implies that we can increase the processing time of $P_{44}$ from 60 to 131 s without changing the fundamental period of the system. Since this slack value is the same for the two sequences, the processing time variation has the same effect for both $\mathbf{Seq}_A$ and $\mathbf{Seq}_B$.

However, for other processing time variations, the impact on the cluster throughput is different. For example, the slack times of $P_{21}$ ($\text{ACT}_{13}$) is 13 s in $\mathbf{Seq}_A$ and 33 s in $\mathbf{Seq}_B$. Assume that the $P_{21}$ processing time is uniformly distributed in [5, 55] s (average of 30 s) and all other actions have deterministic processing time as those in Table VI. For $\mathbf{Seq}_B$, the fundamental period can be maintained at 136 s all the time since the slack time for $P_{21}$ allows 33-s variation. For $\mathbf{Seq}_A$, the 136-s fundamental period cannot sustain when $t_{21} > 43$ s. For example, if $t_{21} = 51$ s, the

TABLE VII
TESTING RESULTS OF THE CMP POLISHER WITH REAL THROUGHPUT 26 WAFERS PER HOUR BY A PRIORITY-BASED SCHEDULER

| Searching strategies | Compt. time ($s$) | Feasible seqs./nets. | Optimal seqs./nets. | **FP** ($s$) |
|---|---|---|---|---|
| All sequences | 219063 | $1.86 \times 10^9$ | $1.25 \times 10^9$ | 136 |
| Network with acceleration | 0.1 | 58 | 17 | 136 |

TABLE VIII
TWO SEQUENCES $\mathbf{Seq}_A$ AND $\mathbf{Seq}_B$ IN NETWORK

| $G_j$ | $\mathbf{Seq}_A$ | $\mathbf{Seq}_B$ |
|---|---|---|
| 1 | $ACT_1 \to ACT_{20}$ | $ACT_{20} \to ACT_1$ |
| 2 | $ACT_3 \to ACT_{17} \to ACT_4 \to ACT_{18} \to ACT_{14} \to ACT_{15} \to ACT_{11} \to ACT_{12}$ | $ACT_3 \to ACT_{17} \to ACT_4 \to ACT_{14} \to ACT_{15} \to ACT_{18} \to ACT_{11} \to ACT_{12}$ |
| 3 | $ACT_5 \to ACT_{10}$ | $ACT_5 \to ACT_{10}$ |
| 4 | $ACT_6$ | $ACT_6$ |
| 5 | $ACT_3 \to ACT_2$ | $ACT_3 \to ACT_2$ |
| 6 | $ACT_4 \to ACT_5 \to ACT_{10} \to ACT_{11}$ | $ACT_4 \to ACT_5 \to ACT_{10} \to ACT_{11}$ |
| 7 | $ACT_5 \to ACT_6 \to ACT_{10}$ | $ACT_5 \to ACT_6 \to ACT_{10}$ |
| 8 | $ACT_6 \to ACT_7, ACT_8, ACT_9$ | $ACT_6 \to ACT_7, ACT_8, ACT_9$ |
| 9 | $ACT_{14} \to ACT_{12} \to ACT_{13}$ | $ACT_{14} \to ACT_{12} \to ACT_{13}$ |
| 10 | $ACT_{17} \to ACT_{15} \to ACT_{16}$ | $ACT_{17} \to ACT_{15} \to ACT_{16}$ |
| 11 | $ACT_{20} \to ACT_{18} \to ACT_{19}$ | $ACT_{20} \to ACT_{18} \to ACT_{19}$ |

TABLE IX
IMPACT OF PROCESSING TIME $t_{13}$ VARIATION OF $P_{21}$ (ACT$_{13}$) ON **FP** (MEAN AND STANDARD DERIVATION) FOR SCHEDULES $\mathbf{Seq}_A$ AND $\mathbf{Seq}_B$

| $P_{21}$ ($ACT_{13}$) process variation | $\mathbf{Seq}_A$ | $\mathbf{Seq}_B$ |
|---|---|---|
| $ACT_{13}$ slack value ($s$) | 13 | 33 |
| $t_{13}$: uniform dist.$\sim \mathbf{U}[0, 55]$ ($s$) | $E(\mathbf{FP}) = 136.6$, $\sigma(\mathbf{FP}) = 1.73$ | $E(\mathbf{FP}) = 136$, $\sigma(\mathbf{FP}) = 0$ |
| $t_{13}$: normal dist. $\sim \mathcal{N}(30, 10)$ ($s$) | $E(\mathbf{FP}) = 136.2$, $\sigma(\mathbf{FP}) = 1.15$ | $E(\mathbf{FP}) = 136$, $\sigma(\mathbf{FP}) = 0.08$ |

corresponding fundamental period is $\mathbf{FP} = 144$ s. If $P_{21}$ processing time instead satisfies a normal distribution with mean $\mu = 30$ s and standard deviation $\sigma = 10$ s, for $\mathbf{Seq}_A$, the $\mathbf{FP} = 136$ s can be maintained at 95.54% of the time while for $\mathbf{Seq}_B$ at 99.99% of the time. The impact under such a variation can be clearly seen from the mean and standard deviation calculations given in Table IX. Obviously, $\mathbf{Seq}_B$ is better than $\mathbf{Seq}_A$ in this case. A similar study can be found in [25].

## VI. CONCLUSION

This paper presents an integrated event graph and network approach to model and analyze the scheduling and throughput of multicluster tools for semiconductor manufacturing. First, a decision-move-done event graph is built to model the multicluster tools so that all actions can be captured as a simple graph and a set of mappings. The use of the event graph model can simplify the tree search methods to find all feasible action sequences and to facilitate the network construction. Network model-based algorithms are then proposed to calculate the minimal cycle time and scheduling of the cluster tools through computing the minimum time span of the network using the critical path method. Grouping methods are employed to reduce the complexity of the network model-based searching process. Furthermore, the slack property of actions can be used for sensitivity analysis to calculate the impact of the processing time variations on the throughput and to compare various schedules. The sensitivity analysis could be used as a guidance for a robust schedule when there exist multiple optimal action sequences that lead to the same throughput and under process variations. The proposed methodology has been successfully implemented on dozens of tools at Intel Corporation. The implementation shows that the methodology is very efficient in calculating the optimal throughputs and identifying productivity improvement opportunitites of the complicated cluster tools. Experimental results of a chemical–mechanical planarization polisher are used as an example to demonstrate and validate the proposed methodology.

## APPENDIX
## PROOF OF PROPOSITION 1

Without loss of generality, the case that all robots $R_i$ are of single blade is considered. For the double-blade robot cases, the proof is similar.

It must be proven that $d(E_i) = 2$ is the minimum empty slots for deadlock-free schedules of cluster $\mathbb{C}_i$. This claim is proven by induction for a $k$-cluster tool.

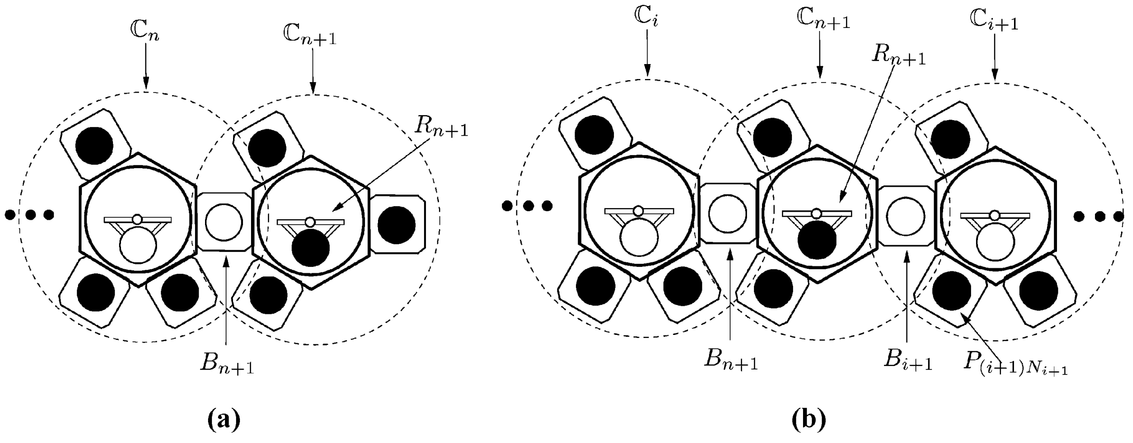

Fig. 7. Two scenarios of $\mathbb{C}_{n+1}$ locations in $(n+1)$-cluster tool: (a) $\mathbb{C}_{n+1}$ at last cluster and (b) $\mathbb{C}_{n+1}$ in middle of clusters.

1) For $k=1$, a single-cluster tool is considered. It is easy to see that if there is only one empty slot within the cluster $d(E)=1$, this slot must be the returning cassette module $C_2$, and all other modules (including robot) are occupied by wafers. Then, the single-blade robot cannot move wafers among process modules and cassette and deadlock is encountered. Therefore, $d(E) \geq 2$ is a requirement and the minimum size of $E$ must be 2.
2) Suppose the statement is true for the case $k=n$ such that $d(E_i) \geq 2$ for all $i=1,\cdots,n$. It must be proven that the inequality also holds for $k+1=n+1$. Consider where the additional $(n+1)$th cluster is located when we extend the $n$ clusters to an $(n+1)$ cluster.
   a) If the $(n+1)$th cluster is at the end of the cluster chain, as shown in Fig. 7(a), it is clear that the cluster is deadlock in steady state if $d(E_{n+1})=1$, since all wafers within $\mathbb{C}_{n+1}$ have been processed and are ready to move to next module. However, the single-blade robot $R_{n+1}$ cannot load wafers from $B_{n+1}$ after it moves out the holding wafer to cluster $\mathbb{C}_n$ through $B_{n+1}$. Note that once one wafer moves into the cluster through the buffer module, one wafer must move out during the same cycle. Therefore, $d(E_{k+1}) \geq 2$.
   b) If the $(n+1)$th cluster is in the middle of the cluster chain between clusters $\mathbb{C}_i$ and $\mathbb{C}_{i+1}$, $i=0,1,\cdots,n-1$, as shown in Fig. 7(b), one can consider the next action that robots can execute. All wafers in $\mathbb{C}_i$, $\mathbb{C}_{n+1}$, and $\mathbb{C}_{i+1}$ have been processed and are ready to move to next station. Suppose $d(E_{n+1})=1$. Notice that $d(E_{i+1})=2$ and robot $R_{i+1}$ of $\mathbb{C}_{i+1}$ will move one wafer from module $P_{(i+1)N_{i+1}}$ to $B_{i+1}$. At the same time, robot $R_{n+1}$ has to move the current-holding wafer to buffer module $B_{n+1}$. Then, the schedule of robot $R_{n+1}$ becomes deadlocked since no wafer can be moved through the cluster $\mathbb{C}_{n+1}$. Therefore, $d(E_{k+1}) \geq 2$.

Combining the previous analyses, we conclude that $d(E_i)=2$, $i=1,\cdots,M$, is the minimum empty slot size for a deadlock-free schedule for an $M$-cluster tool equipped with single-blade robots.

## Acknowledgment

The authors would like to thank the three anonymous reviewers for their constructive feedback. They also thank K. Bryant and R. Akhavan-Tabatabaei at AzFSM (Fab 12/22/32) Industrial Engineering, Intel Corporation, for help in preparing the paper.

## References

[1] Y. Crama and J. van de Klundert, "Cyclic scheduling of identical parts in a robotic cell," *Oper. Res.*, vol. 45, no. 6, pp. 952–965, 1997.
[2] M. Dawande, C. Sriskandarajah, and S. Sethi, "On throughtput maximization in constant travel-time robotic cells," *Manuf. Service Oper. Manage.*, vol. 4, no. 4, pp. 296–312, 2002.
[3] I. Drobouchevitch, S. Sethi, and C. Sriskandarajah, "Scheduling dual gripper robotic cells: One-unit cycles," *Eur. J. Oper. Res.*, vol. 171, no. 2, pp. 598–631, 2006.
[4] T. Perkinson, P. McLarty, R. Gyurcsik, and R. Cavin, "Single-wafer cluster tool performance: An analysis of throughput," *IEEE Trans. Semiconduct. Manufact.*, vol. 7, no. 3, pp. 369–373, Aug. 1994.
[5] S. Venkatesh, R. Davenport, P. Foxhoven, and J. Nulman, "A steady-state throughput analysis of cluster tools: Dual-blade versus single-blade robots," *IEEE Trans. Semiconduct. Manufact.*, vol. 10, no. 4, pp. 418–424, Nov. 1997.
[6] T. Perkinson, R. Gyurcsik, and P. McLarty, "Single-wafer cluster tool performance: An analysis of the effects of redundant chambers and revisitation sequences on throughput," *IEEE Trans. Semiconduct. Manufact.*, vol. 9, no. 3, pp. 384–400, Aug. 1996.
[7] R. Srinivasan, "Modeling and performance analysis of cluster tools using Petri nets," *IEEE Trans. Semiconduct. Manufact.*, vol. 11, no. 3, pp. 394–403, Aug. 1998.
[8] W. Zuberek, "Timed Petri nets in modeling and analysis of cluster tools," *IEEE Trans. Robot. Automat.*, vol. 17, no. 5, pp. 562–575, Oct. 2001.
[9] S. Rostami, B. Hamidzadeh, and D. Camporese, "An optimal periodic scheduler for dual-arm robots in cluster tools with residency constraints," *IEEE Trans. Robot. Automat.*, vol. 17, no. 5, pp. 609–618, Oct. 2001.
[10] S. Rostami and B. Hamidzadeh, "Optimal scheduling techniques for cluster tools with process-module and transport-module residency constraints," *IEEE Trans. Semiconduct. Manufact.*, vol. 15, no. 3, pp. 341–349, Aug. 2002.
[11] H. T. LeBaron and R. A. Hendrickson, "Using emulation to validate a cluster tool simulation model," in *Proc. 2000 Winter Simulation Conf.*, Orlando, FL, 2000, pp. 1417–1422.
[12] H. T. LeBaron and M. Pool, "The simulation of cluster tools: A new semiconductor manufacturing technology," in *Proc. 1994 Winter Simulation Conf.*, Orlando, FL, 1994, pp. 907–912.
[13] M. A. Dümmler, "Using simulation and genetic algorithms to improve cluster tool performance," in *Proc. 1999 Winter Simulation Conf.*, Phoenix, AZ, 1999, pp. 875–879.

[14] D. A. Nehme and N. G. Pierce, “Evaluation the throughput of cluster tools using event-graph simulations,” in *Proc. 1994 IEEE/SEMI Adv. Semiconduct. Manufact. Conf.*, Cambridge, MA, 1994, pp. 189–192.

[15] D. Pederson and C. Trout, “Demonstrated benefits of cluster tool simulation,” in *Proc. 2002 Int. Conf. Modeling Analysis Semiconduct. Manufact.*, Tempe, AZ, 2002, pp. 84–89.

[16] S. Ding and J. Yi, “An event graph based simulation and scheduling analysis of multicluster tools,” in *Proc. 2004 Winter Simulation Conf.*, Washington, DC, 2004, pp. 1915–1924.

[17] J. Herrmann, N. Chandrasekaran, B. Conaghan, M. Nguyen, G. Rubloff, and R. Zhi, “Evaluating the impact of process changes on cluster tool performance,” *IEEE Trans. Semiconduct. Manufact.*, vol. 13, no. 2, pp. 181–192, May 2000.

[18] J. Yi, S. Ding, and D. Song, “Steady-state throughput and scheduling analysis of multicluster tools for semiconductor manufacturing: A decomposition approach,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Barcelona, Spain, 2005, pp. 293–299.

[19] N. Geismar, C. Sriskandarajah, and N. Ramanan, “Increasing throughput for robotic cells with parallel machines and multiple robots,” *IEEE Trans. Automat. Sci. Eng.*, vol. 1, no. 1, pp. 84–89, Jan. 2004.

[20] L. Schruben, “Simulation modeling with event graphs,” *Commun. ACM*, vol. 26, no. 11, pp. 957–963, 1983.

[21] J. Kelley, Jr., “Critical-path planning and scheduling: Mathematical basis,” *Oper. Res.*, vol. 9, no. 3, pp. 296–320, 1961.

[22] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[23] L. Schruben, SIGMA User’s Guide, Dept. Industrial Eng. Oper. Res., Univ. California, Berkeley, 2000.

[24] D. Hetherington and D. Stein, “Recent advances in endpoint and in-line monitoring techniques for chemical-mechanical polishing processes,” in *Proc. CMP-MIC Conf.*, Santa Clara, CA, 2001, pp. 315–323.

[25] S. Ding, J. Yi, M. T. Zhang, and R. Akhavan-Tabatabaei, “Performance evaluation and schedule optimization of multicluster tools with stochastic process times,” in *Proc. IEEE Conf. Automat. Sci. Eng.*, Shanghai, China, 2006.

**Shengwei Ding** received the B.S. and M.S. degrees in electrical engineering from Zhejiang University, China, in 1996 and 1999, respectively. He received the Ph.D. degree in industrial engineering and operation research from the University of California, Berkeley, in 2004.

He is currently with the Department of Industrial Engineering and Operations Research, University of California, Berkeley. His research interests include simulation, scheduling, queuing models, and production management and semiconductor manufacturing.

Dr. Ding is a member of IIE and INFORMS.

**Jingang Yi** (S’99–M’02) received the B.S. degree in electrical engineering from the Zhejiang University, China, in 1993, and the M.Eng. degree in precision instruments from Tsinghua University, China, in 1996. He also received the M.A. degree in applied mathematics and Ph.D. degree in mechanical engineering from the University of California, Berkeley, in 2001 and 2002, respectively.

From May 2002 to January 2005, he was with Lam Research Corporation, Fremont, CA, as a member of technical staff. Since January 2005, he has been with the Department of Mechanical Engineering, Texas A&M University, as a Visiting Assistant Professor. His research interests include intelligent and autonomous systems, mechatronics, automation science and engineering with applications to semiconductor manufacturing and intelligent transportation systems.

Dr. Yi is a member of the American Society of Mechanical Engineering (ASME). He was the recipient of the Kayamori Best Paper Award of the 2005 IEEE International Conference on Robotics and Automation (ICRA).

**Mike Tao Zhang** (S’98–M’01–SM’05) received the M.S. and Ph.D. degrees, from the Department of Industrial Engineering and Operations Research, in 2000 and 2001, respectively, as well as the Management of Technology certificate, in 2000, from the Haas School of Business, all from the University of California, Berkeley.

He is currently a Staff Engineer at Intel Corporation, Chandler, AZ. He has been a Senior Engineer, a Group Leader, and a Department Manager at various Intel sites. He was awarded two patents and has published about 50 papers and three books/book chapters. His research interests include industrial automation, manufacturing systems, operations research/management, and supply chain management.

Dr. Zhang is a Member of the Honor Society of Phi Kappa Phi and also a Senior Member of the Institute of Industrial Engineers (IIE). He is Co-Chair of the IEEE Robotics and Automation Society Technical Committee on Semiconductor Factory Automation. He is an Associate Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING and a Guest Editor of *Assembly Automation*, the IEEE Robotics and Automation Magazine, and the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING. He is the recipient of the Intel ATM Achievement Award and the IIE Outstanding Young Industrial Engineer Award.