Optimal Scheduling of Multicluster Tools With Constant Robot Moving Times, Part I: Two-Cluster Analysis

Wai Kin Victor Chan, *Member, IEEE*, Jingang Yi, *Senior Member, IEEE*, and Shengwei Ding

***Abstract*—In semiconductor manufacturing, finding an efficient way for scheduling a multicluster tool is critical for productivity improvement and cost reduction. This two-part paper analyzes optimal scheduling of multicluster tools equipped with single-blade robots and constant robot moving times. In this first part of the paper, a resource-based method is proposed to analytically derive closed-form expressions for the minimal cycle time of two-cluster tools. We prove that the optimal robot scheduling of two-cluster tools can be solved in polynomial time. We also provide an algorithm to find the optimal schedule. Examples are presented to illustrate the proposed approaches and formulations.**

***Note to Practitioners*—The main challenge in scheduling multicluster tools in semiconductor manufacturing is to handle the cluster interactions. These interactions create phenomena that do not exist in single-cluster tools. This paper presents an analytical and algorithmic solution for scheduling two-cluster tools with constant robot moving times. The analysis and results presented in this paper provide foundations to solve optimal scheduling of multicluster tools that will be discussed in the companion paper. We propose a resource-based method to analyze the cycle time and robot schedule of multicluster tools. The advantage of such an approach is its effective and efficient treatment to capture the complex schedules among various modules and wafers. The interaction dependencies among two clusters are revealed by the resource-based analysis and scheduling algorithms are also presented. We illustrate the proposed analysis and algorithms through several examples.**

***Index Terms*—Cluster tools, multiple machines, performance/productivity, scheduling, semiconductor manufacturing.**

I. Introduction

CLUSTER tools are widely used manufacturing equipment in semiconductor industry. A typical cluster tool consists of process modules (PMs), transport modules (robots), and load-locks (cassette modules). Each wafer is transported from the load-locks to PMs according to predefined routing sequences (recipes). The robot in a cluster tool can be single-blade or double-blade. A single-blade robot usually can hold only one wafer at a time. A double-blade robot has two independent arms and therefore can hold two wafers at the same time with one on each arm. A multicluster tool is composed of multiple single cluster tools in a noncyclic fashion. Fig. 1 shows the schematic view of a two-cluster tool. In this example, there are two robots that transfer wafers among the PMs and between the two single clusters. Between the two adjacent single clusters, there is a buffer with finite wafer capacity for holding incoming and outgoing wafers.

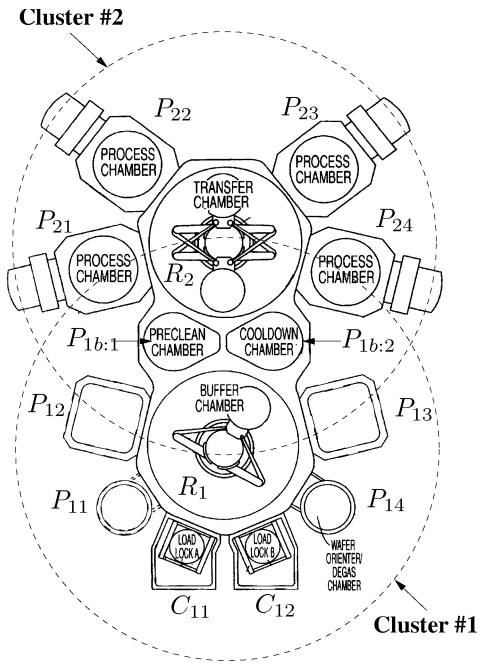

Fig. 1. A schematic of a two-cluster tool [1].

The robot scheduling and throughput analysis of a multicluster tool are more complicated than that of a single cluster tool for several reasons. First, the interconnected clusters create coupling and dependence effects that complicate the scheduling analysis. Second, congestion could propagate from cluster to cluster, making it difficult to compute the overall cycle time. Third, the number of wafers allowed in each single cluster can vary. Allowing too many wafers in the tool could create deadlock, while having too few wafers will probably starve some PMs, thus reducing the throughput.

Manuscript received June 24, 2009; revised December 10, 2009; accepted February 09, 2010. Date of publication April 15, 2010; date of current version January 07, 2011. This paper was recommended for publication by Associate Editor N. Wu and Editor M. Zhou upon evaluation of the reviewers' comments. This paper was presented in part at the 2007 IEEE International Conference on Automation Science and Engineering, Scottsdale, AZ, USA, September 22–25, 2007, and the 2008 IEEE International Conference on Automation Science and Engineering, Washington DC, USA, August 24–26, 2008.

W.-K. V. Chan is with the Department of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: chanw@rpi.edu).

J. Yi is with the Department of Mechanical and Aerospace Engineering, Rutgers University, Piscataway, NJ 08854 USA (e-mail: jgyi@rutgers.edu).

S. Ding is with the Direct Store Delivery (DSD) Division of Nestle USA, Oakland, CA 94618 USA (e-mail: dingsw@cal.berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2010.2046891

Perkinson *et al.* [2] and Venkatesh *et al.* [3] discuss analytical models for calculating the steady-state throughput of single-cluster tools equipped with single-blade and double-blade robots under the assumption of zero robot moving time, respectively. Only the "pull" (reverse) schedule (for single-blade robots) and the "push" (swap) schedule (for double-blade robots) are discussed. By the pull schedule, the single-blade robot is sequentially moving wafers from one PM to the next PM in the wafer flow, while by a swap schedule, the double-blade robot swaps wafers by using its two blades at a PM. Cluster tools are a special type of robotic cells. The optimal scheduling problem of robotic cells has been studied extensively in the past decade, see [4]–[6] and the references therein. Dawande *et al.* [7] consider the optimal scheduling of single robotic cells under constant robot moving time and provide a polynomial time algorithm for finding the optimal one-unit cycle schedule. Drobouchevitch *et al.* [8] study double-grip robotic cells and prove the optimality of the swap schedule. The work in [5], [9]–[11] analyze single robotic flow shop with single- and double-gripper robots without buffers between stations. Recently, Dawande *et al.* [12] summarize the work on sequencing and scheduling of single robotic cells. Heuristics and approximation algorithms dealing with k-unit cycles in robotic cells are also studied in [13] and [14].

For multicluster tools or multirobot cells, only few results have been reported. The work in [1], [15] discuss several rule- or priority-based heuristic methods for scheduling multicluster tools. Geismar *et al.* [16] discuss a robotic cell with three single-gripper robots for semiconductor manufacturing with simulation study. Ding *et al.* [17] present an integrated event graph and network model to find all optimal schedules of a multicluster tool. The approaches in [17] are based on simulation models rather than analytical examination. Recently, Yi *et al.* [18] discuss an analytical scheduling scheme for multicluster tools using a decomposition approach. Zero robot moving time is assumed in the analysis and only pull and push schedules are considered. Chan *et al.* [19] extend the results in Yi *et al.* [18] with a consideration of constant robot moving time. However, neither Chan *et al.* [19] nor Yi *et al.* [18] considers the issue of multiple-wafer cycles that exists in multicluster tools.

The main objective of this two-part paper is to introduce a methodology to analyze the cycle time and optimal robot-movement sequences in a multicluster tool with a tree-like topology. We extend the results in [19], [20] to analyze multicluster tools with single-blade robots and constant robot moving times. The first part of the paper focuses on two-cluster tools. The results developed in this paper will be extended in the companion paper [21] to tackle the scheduling of multicluster tools. The contributions of this first-part paper are three-folds. First, due to constant robot moving time among PMs, the results discussed in [18] are no longer valid. We propose a concept of resource cycles that allows us to analytically quantify the dependence among connected clusters. Instead of computing the robot waiting times at the modules as in [7] and [18], the resource-based analysis examines the cycle times of PMs, transport robots, and buffer-process modules. Such a unified treatment of all resources simplifies the analysis of the interactions among clusters. Second, we present an analytical solution to capture multiple-wafer cycle scheduling, which can dominate the cycle time of the cluster tools and is unique for multicluster tools. Finally, we propose an optimal schedule for two-cluster tools and prove that finding the optimal schedule of a two-cluster tool is polynomial.

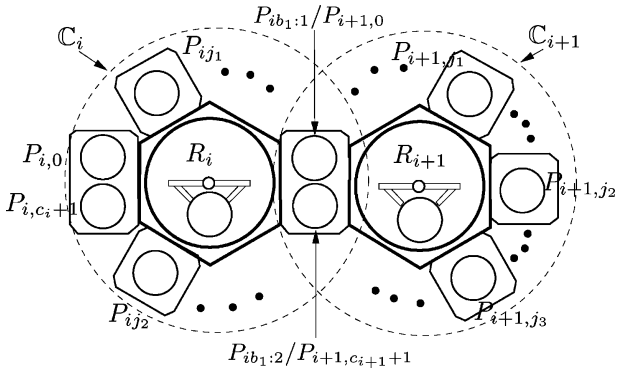

Fig. 2. A schematic of a two-cluster tool with buffer modules.

The proposed resource-based approach complements other existing modeling methods for cluster tools, such as graph-based Petri net methods [22]–[24], mathematical programs [25], [26], or simulation [17], in several aspects. First, we obtain the closed-form cycle-time formulations that uncover the structural properties of the cycle time, thereby allowing us to evaluate the optimal scheduling problem and to develop efficient algorithms to find the optimal schedule. Second, the resource-based method can be easily generalized and directly applied to a class of multicluster tools with various configurations. In the companion paper [21], we will show one such extension to tree-like cluster topology tools. This property is attractive to enable the automation of scheduling process of multicluster tools and therefore improves the scheduling efficacy and effectiveness for a variety of cluster tools.

The remainder of this paper is organized as follows. We present preliminary concepts and results in Section II. In Section III, we introduce the resource-based approach. We illustrate the cycle-time calculation and scheduling of two-cluster tools in Section IV before concluding this paper in Section V.

II. Preliminaries

A two-cluster tool is shown in Fig. 2. Each single-cluster $\mathbb{C}_i$, $i = 1, 2$, consists of c_i number of PMs, denoted by P_{ij}, $j = 1, \ldots, c_i$, and a single-blade robot, denoted by R_i. We assume that buffer modules have either one- or two-wafer capacity. A single-space buffer is denoted by P_{ib_i} and both inlet and outlet wafers share the same space. For a double-space buffer module, let $P_{ib_i:1}$ and $P_{ib_i:2}$ denote, respectively, the outlet buffer (transferring wafers from $\mathbb{C}_i$ to $\mathbb{C}_{i+1}$) and the inlet buffer (transferring wafers from $\mathbb{C}_{i+1}$ back to $\mathbb{C}_i$).

For $\mathbb{C}_i$, the buffer module P_{ib_i} is considered as a virtual process module, while for $\mathbb{C}_{i+1}$ it is viewed as a virtual load-locks module. Let t_{ij} and t_{v_i} denote the processing time for P_{ij} and P_{ib_i}, respectively. We assume no real wafer processing at all buffer modules. Associated with $\mathbb{C}_i$, there is a vector of deterministic activity times, $\mathbf{t}_i = (t_{i0}, t_{i1}, \ldots, t_{ic_i}; \epsilon_i, \delta_i)$, where ϵ_i is the time to load/unload a wafer to/from a PM,

and δ_i is the robot moving time between any pair of modules/buffers/load-locks within $\mathbb{C}_i$. To facilitate presentation, we define

$$\beta_i = 2(\epsilon_i + \delta_i), \quad \alpha_{ij} = 2\epsilon_i + \delta_i + t_{ij}, \quad \beta_i^{\delta_i} = \beta_i - \delta_i. \tag{1}$$

Note that β_i is the time needed for R_i transferring and loading/unloading a wafer from one PM to another PM, while α_{ij} is the time needed for R_i transferring to P_{ij}, waiting for the process at P_{ij} to finish, and unloading the processed wafer from P_{ij}. Such observations will be used later for cycle time calculation.

The inlet and outlet spaces of a load-locks module are two distinct objects and a robot needs δ_i to travel between the two spaces (e.g., [7]). Since we consider buffer module P_{ib_i} as a virtual load-locks module $P_{i+1,0}$ of $\mathbb{C}_{i+1}$, it is natural to assume a robot moving time δ_i from the outlet space to the inlet space of $P_{i+1,0}$. In the case of a single-space buffer, this delay time can be viewed as the robot alignment time before picking up a new wafer after unloading one to the load-locks module.

We define *basic activity* A_{ij} associated with P_{ij} in $\mathbb{C}_i$ as four consecutive movements by robot R_i: 1) moving to P_{ij}; 2) unloading a wafer from P_{ij}; 3) moving from P_{ij} to $P_{i,j+1}$; and 4) loading the wafer into $P_{i,j+1}$. Similarly, activities A_{i0} and A_{ib_i} deal with buffers P_{i0} and P_{ib_i}. A robot schedule can be decomposed into a combination of several such basic activities. The time duration to perform A_{ij} depends on its previous action. If the sequence is $A_{ik}A_{ij}$ $(k \neq j-1)$, it takes β_i to perform A_{ij}, including: 1) move to P_{ij} (δ_i); 2) unload a wafer from P_{ij} (ϵ_i); 3) move the wafer to $P_{i,j+1}$ (δ_i); and 4) load the wafer into $P_{i,j+1}$. Similarly, if the sequence is $A_{i,j-1}A_{ij}$, then at the end of activity $A_{i,j-1}$, R_i is already at P_{ij} and must wait for P_{ij}'s completion. Therefore, it takes α_{ij} to perform A_{ij} in this case. For the case $A_{ic_i}A_{i0}$, at the end of A_{ic_i}, although R_i is already at P_{i0}, we assume there is a robot moving time from the outlet space to the inlet space of load-locks (or alignment time in the single-space case). To simplify notation, we assume that this time delay is δ_i. Therefore, the time to complete A_{i0} in this case is also δ_i.

Definition 1 ([7]): A *one-unit cycle* is a sequence of feasible activities consisting of every activity exactly once and during which, exactly one wafer is imported from and exported to the outside of the cluster tool. The *one-unit cycle time* (or simply cycle time) is the minimum time required to complete the one-unit cycle.

In a single-cluster tool, a one-unit cycle can return the tool to the same state as the beginning of the cycle. However, this is not always true for multicluster tools. We will show later (in Section IV-B) that there may exist the cases of repeating the same one-unit schedule for n times in order to return the whole multicluster tool to the beginning state, where n depends on the number of wafers inside the tool. The time to execute each of these one-unit schedule could also be different. We borrow the concept of n-concatenated robot-move-sequences (n-CRM) in [11] to denote such multicycle schedules. The existence of n-CRM is unique in multicluster tools and further complicates scheduling of multicluster tools.

Remark 1: As we will show later in Section IV-B, we cannot handle these n-CRM cycles directly by using results developed for single-cluster tools [13]. In this paper, we do not try to minimize the n-CRM cycles and instead, we focus on understanding these repeating n-CRM cycles by calculating and comparing the (average) one-unit cycle time of multicluster tools. Also, the extension of k-unit cycle schedules of cluster tools [13] to multicluster tools is out of scope of this paper and we leave this subject as future work.

The throughput of a single-cluster tool is the number of wafers processed per unit time, which is equal to the reciprocal of the cycle time. For a tighter schedule in steady state, it is appropriate to assume that: 1) the processing of wafers at each PM starts right after the robot places an unprocessed wafer inside that PM and 2) the robots take actions as early as possible for a given moving sequence. Under these assumptions, we consider only the sequencing of robot movement activities and therefore the cluster tool's schedule and cycle time are determined by the robot moving sequence and the wafer distribution among cluster tools (we will show this in Section IV-A). For cluster $\mathbb{C}_i$, we denote one-unit cycle schedule as π_i that is captured by a series of activities, $\pi_i = (A_{ij_0}A_{ij_1}\cdots A_{ij_{c_i}})$, where the indexing set $j_0j_1\cdots j_{c_i}$ is a permutation of $\{0,1,2,\ldots,c_i\}$. For a two-cluster tool $\mathbb{C}_1$ and $\mathbb{C}_2$, an optimal one-unit schedule is the one-unit cycle schedule $\boldsymbol{\pi}^* = (\pi_1^*, \pi_2^*)$ and the wafer distribution $\boldsymbol{n}^* = (n_1^*, n_2^*)$ that minimize the cycle time of the two-cluster tool. After we identify the best wafer distribution $\boldsymbol{n}^*$, we shall always assume this distribution and drop its dependency from the cycle time expression. Therefore, we simply use π_i to represent $\mathbb{C}_i$'s schedule.

Similar to [18], we define the *decoupled single-cluster* $\mathbb{C}_i$ as the ith single cluster detached from a multicluster tool by replacing the buffer modules with virtual PMs and infinite number of wafers and spaces at the load-locks. When without confusion, we will use $\mathbb{C}_i$ to indicate both the ith cluster of a multicluster tool and the decoupled ith cluster.

III. Resource-Based Cycle Time Analysis

A. Resources and Activity Chains

We consider a PM or a robot as a "resource" because they are physical entities to provide services to wafers, that is, processing and transporting wafers. During a one-unit cycle at steady state, every resource will go through a cycle. For instance, the cycle of a PM can be counted from starting processing of a wafer until the time of starting processing of the next wafer. Therefore, we name such a PM cycle or robot cycle as a "resource cycle" and its duration as the "resource cycle time." Since there are c_i PMs and a robot R_i in $\mathbb{C}_i$, there are exactly c_i+1 resource cycle times. Before presenting the calculation procedure of resource cycle times, we formally introduce push and pull robot schedules.

Definition 2: A *push schedule* of $\mathbb{C}_i$ with c_i PMs is a sequence of activities with indexes in an increasing order, namely, $A_{i0}A_{i1}\cdots A_{ic_i}$, and a *pull schedule* is a sequence of activities with indexes in a decreasing order, namely, $A_{i0}A_{ic_i}\cdots A_{i1}$.

We denote the time for a wafer to go through all processes in $\mathbb{C}_i$ as T_i^F. It is straightforward to obtain

$$T_i^F = \beta_i^{\delta_i} + \sum_{l=1}^{c_i} \alpha_{il}. \tag{2}$$

Note that the cycle time of a pull schedule is equal to the above time plus the movement time from the outlet space to the inlet space of the buffer module, that is $T_i^F + \delta_i = \beta_i + \sum_{l=1}^{c_i} \alpha_{il}$.

To determine the number of wafers allowed in $\mathbb{C}_i$ under a given π_i, we note that A_{i0} is the only activity that brings new wafers into the tool, while A_{ic_i} is the only activity that removes wafers from the tool. Therefore, the number of wafers will decrease by one after A_{ic_i} and increase by one when A_{i0} begins. Other than the time when A_{i0} and A_{ic_i} occur, the number of wafers inside the tool remains constant, and is determined by the schedule π_i. Let n_i denote the number of wafers that appear inside $\mathbb{C}_i$ excluding the duration between A_{i0} and A_{ic_i}. The following concept is useful in determining n_i.

Definition 3: An *activity chain* in a single cluster is a sequence of activities with consecutively increasing order indexes appeared, not necessary adjacently, in a schedule.

As an example, consider a single-cluster tool with three PMs under schedule $\pi = (A_{i0}A_{i1}A_{i3}A_{i2})$. Activities $A_{i0}A_{i1}$ and A_{i2} have their indexes in a consecutively increasing order although they do not completely adjacent to each other (i.e., separated by A_{i3}). Therefore, these two activities constitute an activity chain. Activity A_{i3} is also an activity chain. Therefore, for π_i, there are two activity chains. In each activity chain, only one wafer is allowed during the whole cycle. Therefore, the number of wafers that could appear in the tool during a cycle is equal to the number of activity chains.

Lemma 1: Given a schedule π_i for $\mathbb{C}_i$, the number of activity chains is equal to the number of wafers n_i in $\mathbb{C}_i$.

B. Resource Cycle Time

The resource cycle for P_{ij} is considered as the sequence of activities during which P_{ij} goes through a cycle, that is, starting from the processing of a wafer at P_{ij} (i.e., activity A_{ij}) until the next wafer enters P_{ij} (i.e., activity $A_{i,j-1}$). Therefore, given a schedule $\pi_i = (A_{ij_0}A_{ij_1}\cdots A_{ij_{c_i}})$, the resource cycle for P_{ij} is the sequence of activities from A_{ij} to $A_{i,j-1}$ inclusively, i.e., $(A_{ij}A_{ij_1}\cdots A_{ij_{k(\pi_i)}}A_{i,j-1})$, where $\{j_1,\ldots,j_{k(\pi_i)}\}$ are the indexes of activities between j and $j-1$ under schedule π_i. The resource cycle for R_i is also defined as the sequence of activities during which R_i goes through a cycle. Because R_i must go through all PMs, its cycle starts from loading a wafer from the cassette module (i.e., activity A_{i0}) and ends at the finish of unloading a wafer to the cassette module (i.e., activity A_{ic_i}). Therefore, the resource cycle of R_i include all activities in π_i. Let $\mathrm{IA}(P_{ij})$ and $\mathrm{IA}(R_i)$ denote respectively the index sets of activities in the resource cycles of P_{ij} and R_i, namely, $\mathrm{IA}(P_{ij}) = \{j, j_1,\ldots,j_{k(\pi_i)}, j-1\}$ and $\mathrm{IA}(R_i) = \{0, j_1,\ldots,j_{c_i}\}$.

Definition 4: The *one-unit resource cycle time* (or simply resource cycle time) $\mathrm{RCT}(Q)$ of resource Q is the minimum cycle time duration for Q to perform all the activities in its own resource cycle without extra waiting for other resource cycles.

Note that the recourse cycle time is calculated *assuming no extra waiting time* for other resource cycles. For example, for P_{ij}, $\mathrm{RCT}(P_{ij})$ is the time to perform activities $(A_{ij}A_{ij_1}\cdots A_{ij_{k(\pi_i)}}A_{i,j-1})$ *assuming* all wafers ready for pickup when the robot arrives except those activities with indexes in consecutive order, such as $A_{i,j-1}A_{ij}$. For buffer module P_{ib_i}, let $\mathrm{RCT}(P_{ib_i})$ and $\mathrm{RCT}^0(P_{ib_i})$ denote resource cycle time of P_{ib_i} assuming its processing time as t_{v_i} and zero, respectively. Similarly, $\mathrm{RCT}(R_i)$ is the time to perform all activities in π_i *assuming* all wafers ready for pickup when the robot arrives except those activities with indexes in consecutive order. Here, consecutive order indexes mean that when the robot finishes moving a wafer from $P_{i,j-1}$ to P_{ij}, it has to wait for the entire processing at P_{ij}. This type of waiting time is not extra, but required according to the schedule. In this case, the resource cycle for P_{ij} coincides with that of the robot, namely, $\mathrm{IA}(P_{ij}) = \mathrm{IA}(R_i)$. In general, for all $j \in \{1,\ldots,c_i\}$ such that $(A_{i,j-1}A_{ij}) \subseteq \pi_i$, $\mathrm{IA}(P_{ij})$ coincides with $\mathrm{IA}(R_i)$. Therefore, all PMs and the robot can be categorized into two types of resources: $\Lambda_i^P = \{j \,|\, (A_{i,j-1}A_{ij}) \not\subseteq \pi_i\}$ and $\Lambda_i^R = \{j \,|\, (A_{i,j-1}A_{ij}) \subseteq \pi_i\}$. Thus, Λ_i^P is the index set of resources whose resource cycle is different from the robot cycle, while Λ_i^R is the index set of resources whose resource cycle times coincide with that of R_i. Note that in [7], Λ_i^P is defined as the index set of machines with partial waiting and Λ_i^R the index set of machines with full waiting.

For each $j \in \Lambda_i^P$, we define the *unavoidable activity* set of P_{ij}, denoted as $\mathrm{UA}(P_{ij})$, as the index set containing $\{j\}$ and the activities contained in both $\mathrm{IA}(P_{ij})$ and Λ_i^R, namely, $\mathrm{UA}(P_{ij}) = \{\mathrm{IA}(P_{ij}) \cap \Lambda_i^R\} \cup \{j\}$. We can compute $\mathrm{RCT}(P_{ij})$ by setting the processing times of all activities in $\{\mathrm{IA}(P_{ij}) \setminus \mathrm{UA}(P_{ij})\}$ equal to β_i (due to no extra waiting in these activities) and adding the activity times α_{ij} for all unavoidable activities, namely,

$$\mathrm{RCT}(P_{ij}) = (|\mathrm{IA}(P_{ij})| - |\mathrm{UA}(P_{ij})|)\beta_i + \sum_{k \in \mathrm{UA}(P_{ij})} \alpha_{ik}, \tag{3}$$

where $|\cdot|$ represents the cardinality of a set. The unavoidable activity set of R_i is Λ_i^R because the robot has to wait for all the PMs in Λ_i^R to finish. Since each activity other than those in Λ_i^R can be accomplished in β_i, the robot resource cycle time is thus computed as

$$\mathrm{RCT}(R_i) = \left(c_i + 1 - \left|\Lambda_i^R\right|\right)\beta_i + \sum_{j \in \Lambda_i^R} \alpha_{ij}. \tag{4}$$

Given a schedule π_i, it is straightforward to determine the resource cycle times of all process modules and the robot. First, Λ_i^R and Λ_i^P are determined by scanning each activity in π_i: if $(A_{i,j-1}A_{ij}) \subseteq \pi_i$, then $j \in \Lambda_i^R$; otherwise, $j \in \Lambda_i^P$. The computational complexity is $O(c_i)$. The determination of $\mathrm{IA}(P_{ij})$ is to include all activity indexes between j and $j-1$ in π_i. This is only required for those js in Λ_i^P. The two extreme schedules are the pull and push schedules, and both have computational complexity $O(c_i)$. The calculation of $\mathrm{UA}(P_{ij})$ is then to take the union of js in Λ_i^P and the intersection of $\mathrm{IA}(P_{ij})$ and Λ_i^R. The computational complexity is $O(c_i)$. Having these sets are determined, the calculation of the resource cycle times can be carried out easily by using (3) and (4).

Example 1: In a single-cluster tool $\mathbb{C}_i$ shown in Fig. 3 with $c_i = 3$ and schedule $\pi_i = (A_{i0}A_{i3}A_{i1}A_{i2})$, the four resources are P_{i1}, P_{i2}, P_{i3}, and R_i. For π_i, $\Lambda^P = \{3,1\}$ and $\Lambda_i^R = \{2\}$. The resource cycle of P_{i3} includes activities $(A_{i3}A_{i1}A_{i2})$. Therefore, $\mathrm{IA}(P_{i3}) = \{3,1,2\}$ and $\mathrm{UA}(P_{i3}) = \{2,3\}$. Consider the beginning state of P_{i3} as the time epoch at which a wafer just starts being processed at P_{i3}. The resource cycle for

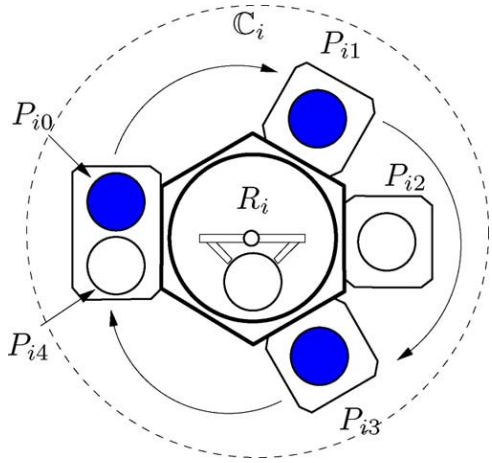

Fig. 3. A single-cluster tool $\mathbb{C}_i$ under schedule $\pi_i = (A_{i0}A_{i3}A_{i1}A_{i2})$.

P_{i3} is the minimum time for P_{i3} to return to this state by assuming no extra waiting in the cycle. Using (3), $\mathrm{RCT}(P_{i3}) = \beta_i + \alpha_{i2} + \alpha_{i3}$ since it includes the following action times: 1) process the job at P_{i3} (t_{i3}); 2) unload it from P_{i3} and move to P_{i4} ($\epsilon_i + \delta_i$); 3) load into P_{i4} and move to P_{i1} ($\epsilon_i + \delta_i$); 4) unload a wafer from P_{i1} (assuming wafer ready because t_{i1} is not in P_{i3}'s resource cycle); 5) move it to P_{i2} and load it into P_{i2} ($\delta_i + \epsilon_i$); 6) wait for the process at P_{i2} (non-extra waiting because t_{i2} is in P_{i3}'s resource cycle) and unload from P_{i2} ($t_{i2} + \epsilon_i$); and 7) move to P_{i3} and load into P_{i3} ($\delta_i + \epsilon_i$), yielding a total time of $t_{i3}+\epsilon_i+3(\delta_i+\epsilon_i)+t_{i2}+\epsilon_i+\delta_i+\epsilon_i = \beta_i + \alpha_{i2} + \alpha_{i3}$.

Similarly, the resource cycle for P_{i1} includes activities $(A_{i1}A_{i2}A_{i0})$, thus, $\mathrm{IA}(P_{i1}) = \{1,2,0\}$, $\mathrm{UA}(P_{i1}) = \{1,2\}$, and by (3), $\mathrm{RCT}(P_{i1}) = \beta_i + \alpha_{i1} + \alpha_{i2}$. Using (4), $\mathrm{RCT}(R_i) = 3\beta_i + \alpha_{i2}$. Note that since $(A_{i1}A_{i2})$ appears in π_i, the resource cycle for P_{i2} coincides with $\mathrm{RCT}(R_i)$.

C. Single-Cluster Tool Cycle Time

Because the tool cycle time cannot be smaller than any one of the resource cycle times, we have the following lemma.

Lemma 2: For a single-cluster tool, the one-unit cycle time $T_i(\pi_i)$ under schedule π_i satisfies

$$T_i(\pi_i) \geq \max\left\{\max_{j\in\Lambda_i^P}\{\mathrm{RCT}(P_{ij})\}, \mathrm{RCT}(R_i)\}\right\}.$$

Dawande *et al.* [7] discuss a concept of *basic cycles* that dominate all other schedules in terms of having smaller cycle times. A basic cycle is a schedule with activity indexes in a decreasing order except those in Λ_i^R. The next result has been shown in [7] and we here provide an alternative proof to illustrate the use of the resource-based method.

Lemma 3: For a single-cluster tool, robot schedules with activity indexes in a decreasing order except those in Λ_i^R dominate all other schedules, that is, having smaller cycle times.

Proof: Suppose $\pi_i = (A_{i0}\cdots A_{ij_l}\cdots A_{ij_{l-1}}\cdots A_{ij_{c_i}})$ includes activities not in a decreasing order, that is, it contains some pair of activities A_{ij} and $A_{i,j-1}$ and a subsequence of activities of length q : $A_{ij}A_{ij+1}\cdots A_{ij+q}$ such that $(A_{i,j-1}\cdots A_{ij}A_{i,j+1}\cdots A_{i,j+q}A_{i,l}) \subseteq \pi_i$, where $j \notin \Lambda_i^R$, $l \neq j+q+1$, and $\mathrm{IA}(P_{ij}) = \{j, j+1, \ldots, j+q, l, \ldots, j_{c_i}, 0, j_1, \ldots, j-1\}$. Then, we can move the activity sequence $A_{ij}A_{i,j+1}\cdots A_{i,j+q}$ to the immediate left of $A_{i,j-1}$, resulting in a new shorter resource cycle time for P_{ij} with an activity index set $\{j, j+1, \ldots, j+q, j-1\}$. Repeating this procedure for all $j \notin \Lambda_i^R$ will give a schedule with all activities in Λ^P in a decreasing order. This procedure does not change Λ^P and Λ^R but it does shorten all resource cycle times. This completes the proof. ∎

Because of the smaller cycle time of the basic cycles, we shall focus on the basic cycles, that is, only basic cycles are considered in scheduling all individual single clusters of a multicluster tool. When computing the resource cycle time, we assume no extra waiting. Therefore, if we ensure that there is at least one resource that can obtain its resource cycle time (without extra waiting), then the resource cycle time of this resource will determine the cycle time of the cluster tool. This observation is stated in the following theorem.

Theorem 1: For a single-cluster tool under a basic cycle π_i:

a) $|\mathrm{IA}(P_{ij})| = |\mathrm{UA}(P_{ij})| + 1$ for all $j \in \Lambda_i^P$;

b) the one-unit cycle time is

$$T_i(\pi_i) = \max\left\{\max_{j\in\Lambda_i^P}\{\mathrm{RCT}(P_{ij})\}, \mathrm{RCT}(R_i)\}\right\}. \quad (5)$$

Proof: Part (a): Suppose the basic cycle schedule has the following form: $\pi_i = (A_{i0}, A_{ij_1}, A_{ik_1}, A_{i,k_1+1}, \ldots, A_{i,k_1+l_{j_1}}A_{i,j_1-1}, A_{ij_2}, A_{ik_2}, \quad A_{i,k_2+1}, \ldots, A_{i,k_2+l_{j_2}}A_{i,j_2-1}, \ldots)$ with $j_1 - 1 > j_2, j_2 - 1 > j_3, \ldots$; otherwise it is not a basic cycle. This schedule has $\Lambda_i^R = \{k_1, k_1+1, \ldots, k_1+l_{j_1}, k_2, \ldots\}$ and $\Lambda_i^P = \{j_1, j_1-1, \ldots\}$. We focus on the resource P_{ij_1}. The proofs for other resources in Λ_i^P are similar. The resource cycle of includes activities $(A_{ij_1}, A_{ik_1}, A_{i,k_1+1}, \ldots, A_{i,k_1+l_{j_1}}A_{i,j_1-1})$.

First, we need to consider the case where there is no activity between A_{ij_1} and A_{ij_1-1}. In this case, both j_1 and $j_1 - 1$ do not belong to Λ_i^R. We have $\mathrm{IA}(P_{ij_1}) = \{j_1, j_1-1\}$ and $\mathrm{UA}\{P_{ij_1}\} = \{j_1, j_1-1\} \cap \Lambda_i^R \cup \{j_1\} = \{j_1\}$, thus $|\mathrm{IA}(P_{ij_1})| = |\mathrm{UA}(P_{ij_1})| + 1$. Suppose now that there are some activities between A_{ij_1} and A_{ij_1-1} in π_i. We have $k_1 + l_{j_1} < j_1 - 1$; otherwise, it is not a basic cycle. Moreover, it can be assumed that the index set $\{k_1, k_1+1, \ldots, k_1+l_{j_1}, j_1-1\}$ includes consecutive integers in an increasing order; otherwise, this set includes at least two index sets and we can move the set with the largest index next to j_1 to obtain a basic cycle. In this case, $\mathrm{IA}(P_{ij_1}) = \{j_1, k_1, k_1+1, \ldots, k_1+l_{j_1}, j_1-1\}$, $\mathrm{UA}\{P_{ij_1}\} = \mathrm{IA}\{P_{ij_1}\} \cap \Lambda_i^R \cup \{j_1\} = \{j_1, k_1+1, \ldots, k_1+l_{j_1}, j_1-1\}$. Therefore, $|\mathrm{IA}(P_{ij_1})| = |\mathrm{UA}(P_{ij_1})| + 1$.

To prove Part (b), from above proof, we know for some $j \in \Lambda^P$, $\mathrm{IA}(P_{ij}) = \{j, j+1, \ldots, j+l_j, j-1\}$. $\mathrm{RCT}(P_{ij})$ includes all processing times of P_{ik}s, $k \in \{j, j+1, \ldots, j+l_j\}$. If there is no waiting when the robot arrives at $P_{i,j-1}$, then $\mathrm{RCT}(P_{ij})$ is equal to the time duration for P_{ij} to return its original state in real operation. However, if waiting is needed when the robot arrives at $P_{i,j-1}$, then the time duration for P_{ij} to return its original state in real operation is determined by $\mathrm{RCT}(P_{i,j-1})$, which can in turn be determined by $\mathrm{RCT}(P_{i,j-2})$, depending on which one is larger. Applying this argument to all $j \in \Lambda^P$ gives the first term in the right side of (5). The second term in the right side of (5) comes from the fact that from Lemma 2, the cycle time cannot be greater than $\mathrm{RCT}(R_i)$. Moreover, the

cycle time cannot be smaller than (5); otherwise, some resources cannot return to the original state. As a result, the cycle time must be equal to what is specified in (5). ■

Remark 2: The calculations of resource cycle time $\mathrm{RCT}(P_{ij})$ in (3) and the cluster tool cycle time $T_i(\pi_i)$ in (5) do not explicitly consider the cases of multiple PMs that conduct exactly the same processes, namely, parallel PMs. We can take the formulation given in [18] to extend the cycle time calculations in this section to include the cases of existence of the parallel PMs in $\mathbb{C}_i$. We omit the formulation here for brevity and assume no parallel PMs in each cluster in this paper.

IV. Optimal Scheduling and Minimal Cycle Time of Two-Cluster Tools

A. Wafer Distribution in a Multicluster Tool

We consider the number of wafers inside $\mathbb{C}_i$ during the steady-state operation after A_{i0} and before A_{ic_i}. The wafers (if any) inside the buffer P_{ib_i} are considered as those of $\mathbb{C}_i$ instead of $\mathbb{C}_{i+1}$. Obviously, the number of wafers in $\mathbb{C}_i$ varies because of the buffer sizes. A single-space buffer must hold no wafer during the steady state for otherwise deadlock will happen. A double-space buffer can hold zero, one, or two wafers without causing a deadlock.[1] We consider a two-cluster $(\mathbb{C}_1, \mathbb{C}_2)$ under schedule (π_1, π_2). Let N denote the total number of wafers in the tool. Suppose we decouple this two-cluster tool into two single-cluster tools $\mathbb{C}_1$ and $\mathbb{C}_2$, by treating the buffer module P_{1b_1} as a virtual PM of $\mathbb{C}_1$. Let n_i denote the number of wafers in $\mathbb{C}_i$ (when it is decoupled from the other one) under schedule π_i, $i = 1, 2$, respectively. Note that n_i is equal to the number of activity chains and is obtained by Lemma 1.

Lemma 4: If P_{1b_1} is single space, then the total number of wafers in the two-cluster tool $\mathbb{C}_1$ and $\mathbb{C}_2$ is $N = n_1 + n_2 - 1$ and is distributed within $\mathbb{C}_1$ and $\mathbb{C}_2$ as $(n_1 - 1, n_2)$; if P_{1b_1} is double space, then wafer distribution can be the following three cases:

a) $N = n_1 + n_2 - 1$ and distributed as $(n_1 - 1, n_2)$;
b) $N = n_1 + n_2$ and distributed as (n_1, n_2);
c) $N = n_1 + n_2 + 1$ and distributed as $(n_1 + 1, n_2)$.

Proof: If P_{1b_1} is single space, N cannot be less than or greater than $n_1 + n_2 - 1$; otherwise, deadlock will occur. When $N = n_1 + n_2 - 1$, after R_1 loads a wafer into P_{1b_1}, it cannot load another one until R_2 returns one to P_{1b_1}; otherwise, deadlock will also occur. In this case, the whole process in $\mathbb{C}_2$ becomes a part of the activity chain that includes A_{1b_1} in $\mathbb{C}_1$. The number of wafers in $\mathbb{C}_1$ decreases to $n_1 - 1$ after R_2 unloads a wafer from P_{20}, yielding distribution $(n_1 - 1, n_2)$, and increases back to n_1 after R_2 returns a wafer to P_{20}. Hence, the wafer distribution is $(n_1, n_2 - 1)$. Since the wafer distribution is defined during the time period after A_{20} and before A_{2c_2}, we shall only use $(n_1 - 1, n_2)$.

If P_{1b_1} is double space, N can be $n_1 + n_2 - 1$, which is the same as in the single-space case because only one of two spaces is used at any time. When $N = n_1 + n_2$, R_2 cannot unload a wafer from P_{20} until it returns one wafer. Since we do not consider the time during A_{2c_2} and A_{20}, the numbers of wafers in $\mathbb{C}_1$ and $\mathbb{C}_2$ are, respectively, n_1 and n_2. The case $N = n_1 + n_2 + 1$ is also possible because one of the two buffer spaces can always contain a wafer. Any other value of N will cause deadlock. This completes the proof. ■

[1]These facts will be shown in Lemma 4 and illustrated in Example 2.

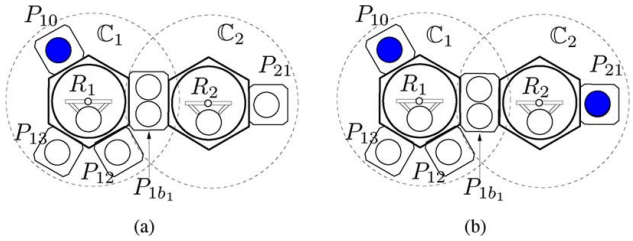

Fig. 4. An example showing two possible cycle times for a given schedule.

The following example shows that the wafer distribution is one factor that captures the interactions among clusters.

Example 2: Consider a two-cluster tool with a double-space buffer P_{1b_1} as shown in Fig. 4. Suppose that $\mathbb{C}_1$ and $\mathbb{C}_2$ follow schedules $\pi_1 = (A_{10}A_{11}A_{12})$ and $\pi_2 = (A_{20}A_{21})$, respectively. Fig. 4(a) and (b) show two possible cycle times under the same schedule (π_1, π_2). In Fig. 4(a), there is only one wafer (the highlighted circular disk) in the tool, and whenever R_1 loads a wafer into P_{1b_1}, it has to wait until $\mathbb{C}_2$ finishes processing the wafer. However, in Fig. 4(b), because there are two wafers in the tool, when R_1 loads a job into P_{1b_1}, it is possible to unload the other wafer from the other space of P_{1b_1} if the processing time at $\mathbb{C}_2$ is fast enough. Therefore, the cycle times of the two situations are different under (π_1, π_2). It seems intuitively that the cycle time of the cluster tool shown in Fig. 4(b) is shorter because of the extra wafer inside $\mathbb{C}_2$. We will formally prove such a intuition in Theorem 3.

B. Minimal Cycle-Time Calculation

In this section, we derive a formulation for computing the cycle time for a given schedule under different wafer distributions. We define the *minimal cycle time* as the smallest cycle time over all possible wafer distributions under the same given schedule in a multicluster tool.

Let $T_i^0(\pi_i)$, $i = 1, 2$, denote the cycle times of the decoupled single-cluster tools with zero time delay at P_{ib_i}. $T_1^0(\pi_1)$ has the following properties. If the wafer distribution is $(n_1 - 1, n_2)$, the wafer upon entering P_{1b_1} becomes immediately ready for pickup and therefore the number of wafers in $\mathbb{C}_1$ can be viewed as n_1. Using the same argument, if the wafer distribution is $(n_1 + 1, n_2)$, the two wafers residing in the two-space buffer can be considered as one wafer when computing the cycle time, and thus the number of wafers in $\mathbb{C}_1$ can again be viewed as n_1. Therefore, regardless of the wafer distributions, $T_1^0(\pi_1)$ remains unchanged.

It is straightforward to observe that in a multicluster tool, the collaborative robot movement cannot further reduce any single-cluster's cycle time $T_i(\pi_i)$ more than what the decoupled cluster with $t_{v_i} = 0$ can reach for a given π_i, namely, $T_i^0(\pi_i) \leq T_i(\pi_i)$, where $T_i(\pi_i)$ denotes the minimal cycle time of $\mathbb{C}_i$ assuming P_{ib_i} has processing time t_{v_i}. Moreover, if we attempt to schedule the multicluster tool based on the two decoupled single clusters, it is unlikely to obtain a cycle time equal to the lower-bound due to the cluster interactions.

Before formally presenting the cycle-time calculations, we shall give an intuitive explanation of how to obtain these results. Basically, there are three parts that we need to consider in the cycle-time calculation:

Part 1. The cycle time of $\mathbb{C}_1$ with the buffer module P_{1b_1} treated as a virtual PM,

Part 2. The cycle time of $\mathbb{C}_2$ when decoupled from $\mathbb{C}_1$, and

Part 3. The delay in the cycle time due to the interaction between $\mathbb{C}_1$ and $\mathbb{C}_2$. This time delay is unique in multicluster tool and will be described next.

Part 2 can be computed by using Theorem 1. Part 1 can be determined by using Theorem 1 and the virtual processing time t_{v_1} of P_{1b_1}, which is equal to the time duration for R_2 to return a wafer to P_{1b_1} after unloading one from it (see the proof of Theorem 2 for the calculation). The time delay in Part 3 can happen when long processing times in $\mathbb{C}_2$ propagate to $\mathbb{C}_1$. In particular, when the sum of all processing times in $\mathbb{C}_2$ is large enough, the entire two-cluster tool cannot return to its original state in one-unit cycle no matter how to collaborate scheduling of two robots. Multiple identical cycles are needed to return the tool to its original state. This is the unique phenomenon in scheduling multicluster tool. As we mentioned before, we have adopted the terminology n-CRM to denote concatenation of the n one-unit robot-move sequences. In the end of an n-CRM cycle, exactly n wafers are produced. Therefore, to compare an n-CRM with a one-unit cycle, we divide the n-CRM period by n. We must take into account the n-CRM cycles when we calculate the cycle time of cluster tools.

Theorem 2: For a two-cluster tool $(\mathbb{C}_1, \mathbb{C}_2)$ connected by a single-space buffer P_{1b_1} and under schedule $\boldsymbol{\pi} = (\pi_1, \pi_2)$ and wafer distribution $(n_1 - 1, n_2)$, the minimal cycle time is

$$T_{12}(\pi_1, \pi_2) = \max\{T_1(\pi_1), T_2(\pi_2), K_{12}\}, \tag{6}$$

where K_{12} is shown in the equation at the bottom of the page, $T_2^F(\pi_2) = \beta_2^{\delta_2} + \sum_{j=1}^{c_2} \alpha_{2j}$, and k_L (k_R) is the largest (smallest) PM indexes (if exist) in Λ_1^P smaller (greater) than b_1 in π_1. In (6), the processing time of P_{1b_1} is $t_{v_1} := \beta_2^{\delta_2} + \sum_{l=1}^{p} \alpha_{2l} + \beta_2 + \sum_{l=q+1}^{c_2} \alpha_{2l}$, where

$$p = \max\{i : \text{activity sequence } (A_{20}A_{21}\cdots A_{2i}) \text{ is contained in } \pi_2\},$$

$$q = \min\{j : \text{activity sequence } (A_{2j}A_{2,j+1}\cdots A_{2,c_2-1}A_{2,c_2}) \text{ is contained in } \pi_2\}.$$

If $p = 0$ ($q = c_2$), then the first (second) summation in t_{v_1} is zero.

Proof: The proof is by a resource-based argument. The first term in the expression is the cycle time of $\mathbb{C}_1$, in which P_{1b_1} is treated as a virtual PM with processing time t_{v_1}. This cycle time is $T_1(\pi_1) = \max\{\max_{j \in \Lambda_1^P}\{\mathrm{RCT}(P_{1j}), \mathrm{RCT}(R_1)\}\}$, in which any resource cycle that includes P_{1b_1} should use t_{v_1} as its process time. We now calculate t_{v_1}. It includes the time for R_2 to perform the following three sets of actions: 1) unload a wafer from P_{1b_1} (P_{20}), move and load the wafer to P_{21} (a total time of $\beta_2^{\delta_2}$); 2) complete the activities $A_{21}, \ldots, A_{2p}$ (if any) (a total time of $\sum_{l=1}^{p} \alpha_{2l}$); 3) move to P_{2q}, unload a wafer from it, and move and load the wafer to $P_{2,q+1}$ (a total time of β_2); and 4) complete the activities $A_{2,q+1}, \ldots, A_{2,c_2-1}$ (if any), and A_{2c_2}, which returns a wafer to P_{20} (a total time of $\sum_{l=q+1}^{c_2} \alpha_{2l}$). Hence, $t_{v_1} = \beta_2^{\delta_2} + \sum_{l=1}^{p} \alpha_{2l} + \beta_2 + \sum_{l=q+1}^{c_2} \alpha_{2l}$. The second term in the expression is the cycle time of $\mathbb{C}_2$ when decoupled from $\mathbb{C}_1$.

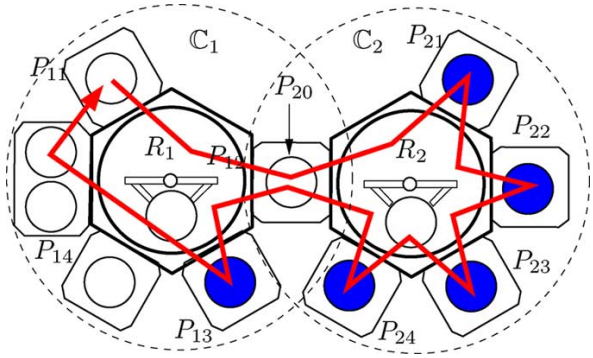

Fig. 5. An example of a two-cluster tool with wafer distribution.

When the overall processing time in $\mathbb{C}_2$ is long enough compared with the cycle time of $\mathbb{C}_1$ and the decoupled cycle time of $\mathbb{C}_2$, congestion will occur in $\mathbb{C}_2$ and propagate to $\mathbb{C}_1$, creating a critical path that spans multiple one-unit cycles, namely, n-CRM exists. Specifically, the time for a wafer to go through $\mathbb{C}_2$ (denoted by T_2^F) is equal to the time to finish the following activities: 1) unloaded from P_{20}, moved, loaded, and processed at P_{21} (a total time of α_{21}); 2) repeat the same set of actions from P_{21} to P_{2c_2} (a total time of $\sum_{j=2}^{c_2} \alpha_{2j}$); and 3) unloaded from P_{2c_2}, moved and loaded to P_{20} (a total time of $\beta_2^{\delta_2}$). Hence, $T_2^F = \beta_2^{\delta_2} + \sum_{j=1}^{c_2} \alpha_{2j}$. This time delay T_2^F propagates to $\mathbb{C}_1$ by adding to the resource cycle time of P_{1b_1}. That is, if the overall processing time in $\mathbb{C}_2$ is long, it will take $\mathrm{RCT}^0(P_{1b_1}) + T_2^F$ (if $b_1 \in \Lambda_1^P$) for the two-cluster tool to return to its original state, in particular, the state of P_{1b_1} relative to the states of other PMs. If $b_1 \in \Lambda_1^R$, then T_2^F will be added to the robot resource cycle as well as the resource cycles of the two neighboring PMs (P_{1k_L} and P_{1k_R}). Moreover, after this T_2^F time delay, all the n_2 wafers (by Lemma 1) originally appeared in $\mathbb{C}_2$ at the beginning of T_2^F have left $\mathbb{C}_2$. As a result, this time delay has to be divided by n_2 to be compared with the one-unit cycles. All together, we obtain the expression for K_{12}. This completes the proof. ■

The following example illustrates a 4-CRM and its cycle time calculation.

Example 3: Consider the two-cluster tool shown in Fig. 5 under schedule

$$\boldsymbol{\pi} = (\pi_1, \pi_2) = ((A_{10}A_{13}A_{14}A_{11}A_{12}), (A_{20}A_{24}A_{23}A_{22}A_{21}))$$

$$K_{12} = \begin{cases} \dfrac{\mathrm{RCT}^0(P_{1b_1}) + T_2^F}{n_2}, & b_1 \in \Lambda_1^P \\ \max\left\{\dfrac{\mathrm{RCT}^0(Q) + T_2^F}{n_2} : Q \in \{P_{1k_L}, P_{1k_R}, R_1\}\right\}, & b_1 \in \Lambda_1^R \end{cases}$$

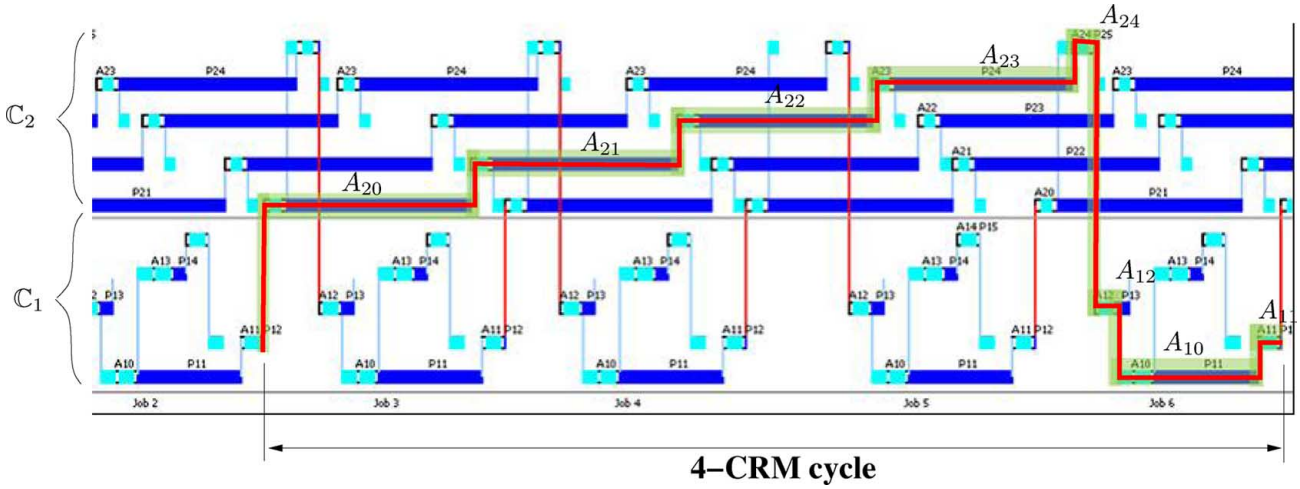

Fig. 6. Gantt chart that shows a 4-CRM cycle (highlighted in red line). The 4-CRM is also the critical path.

and wafer distribution $\boldsymbol{n} = (n_1 - 1, n_2) = (1, 4)$. The processing times (in s) for each cluster are given as

$$\mathbf{t}_1 = (t_{11}, t_{12}, t_{13}, t_{14}; \epsilon_1, \delta_1) = (45, 0, 5, 5; 2, 6),$$
$$\mathbf{t}_2 = (t_{21}, t_{22}, t_{23}, t_{24}; \epsilon_2, \delta_2) = (80, 80, 75, 77; 3, 4).$$

In this example, $\mathbb{C}_2$ has an overall long processing time in each PM. In the Gantt chart shown in Fig. 6, the highlighted path is the 4-CRM that must be fully executed before the tool can return to its original state. Since all the activity times on this path are long, they add up and propagate to $\mathbb{C}_1$ to create a cycle that cannot be fitted equally into several one-unit cycles. Therefore, the two-cluster tool cannot repeat its state in a one-unit cycle. Thereby, the 4-CRM cycle repeats the state of the two-cluster tool every four one-unit cycles and is the critical path. This critical path is also illustrated by the solid flow line in Fig. 5. Following Theorem 2, we have:

$$\begin{aligned}
&T_{12}(\pi_1, \pi_2) \\
&\quad = \max\{T_1(\pi_1), T_2(\pi_2), K_{12}\} \\
&\quad = \max\{\max\{\mathrm{RCT}(P_{11}), \mathrm{RCT}(P_{13}), \mathrm{RCT}(R_1)\}, \\
&\qquad \max\{\mathrm{RCT}(P_{21}), \mathrm{RCT}(P_{22}), \mathrm{RCT}(P_{23}), \\
&\qquad \mathrm{RCT}(R_{24}), \mathrm{RCT}(R_2)\}, \\
&\qquad \max\left\{\frac{\mathrm{RCT}^0(P_{11}) + T_2^F}{4}, \frac{\mathrm{RCT}(R_1) + T_2^F}{4}, \right. \\
&\qquad \left. \frac{\mathrm{RCT}^0(P_{13}) + T_2^F}{4}\right\}\Big\} \\
&\quad = \max\{\max\{105, 80, 97\}, \\
&\qquad \max\{104, 104, 99, 101, 70\}, \\
&\qquad \max\{110.75, 104.5, 108.75\}\} = 110.75\,\mathrm{s}.
\end{aligned}$$

We note that because $b_1 \in \Lambda_1^R$, $b_1 = 2$, $k_L = 1$, $k_R = 3$, $\mathrm{RCT}^0(P_{1b_1}) = \mathrm{RCT}(R_1)$. Also, the virtual processing time $t_{v_1} = \beta_2^\delta + \beta_2$.

If buffer P_{1b_1} is double space, there could be three possible cycle times according to the wafer distribution given in Lemma 4.

Theorem 3: For a two-cluster tool $(\mathbb{C}_1, \mathbb{C}_2)$ connected by a double-space buffer P_{1b_1} and under schedule $\boldsymbol{\pi} = (\pi_1, \pi_2)$, the cycle time has three possibilities.

a) For wafer distribution $(n_1 - 1, n_2)$, $T_{1,2}(\pi_1, \pi_2)$ is given in Theorem 2;
b) For wafer distribution (n_1, n_2), $T_{1,2}(\pi_1, \pi_2) = \max\{T_1(\pi_1), T_2(\pi_2)\}$; and
c) For wafer distribution $(n_1 + 1, n_2)$ $T_{1,2}(\pi_1, \pi_2) = \max\{T_1(\pi_1), T_2(\pi_2), R_{12}\}$, where $R_{12} = \mathrm{RCT}(R_1) + \mathrm{RCT}(R_2) + 2(\epsilon_1 + \epsilon_2) - \mathrm{RCT}(P_{1b_1}) - \mathrm{RCT}(P_{20})$.

Proof: See Appendix A. ■

Theorems 2 and 3 imply that given a schedule $\boldsymbol{\pi} = (\pi_1, \pi_2)$, the best wafer distribution is $\boldsymbol{n} = (n_1, n_2)$ if the buffer is double space and $\boldsymbol{n} = (n_1 - 1, n_2)$ if the buffer is single space, where n_1 and n_2 are determined by Lemma 1. Note that such an observation is also obtained for the simulation and graph model-based approach for scheduling multicluster tools in [17].

C. Optimal Robot Scheduling

The following theorem states that the optimal schedules for a two-cluster tool can be found in polynomial time. The proof and the scheduling algorithm are given in Appendix B.

Theorem 4: An optimal schedule for a two-cluster tool can be found in polynomial time.

V. Conclusion

In this paper, we presented optimal scheduling of two-cluster tools with single-blade robots and constant robot moving times. Scheduling multicluster tools is much more difficult than just simply combining the optimal schedules of individual single clusters by aligning these schedules. It has been shown that the cycle time of the two-cluster tools is determined by not only the robot moving sequences within single clusters but also the wafer distributions and the interactions between two clusters. These interactions introduce dependencies that create the phenomenon of n-CRM cycles. As a result, both single clusters must repeat their one-unit schedules multiple times to obtain a shorter full cycle. We proposed a resource-based approach to analytically capture the dependencies between two single clusters. A closed-form cycle time expression was obtained and a polynomial-time algorithm to find the optimal schedule was also provided. We illustrated the analysis and methodologies through

several examples. The analyses and results in this paper will be served as a foundation that will be further extended to schedule tree-like topological multicluster tools in the companion paper [21].

APPENDIX A
PROOF OF THEOREM 3

Part (a): It suffices to show that the two-space buffer functions similar to a single-space buffer. It has been shown that scheduling of cluster tools can be formulated as linear programs whose constraints represent the precedence relationships among the activities [25]. Let S^U_{ijl} and S^L_{ijl} denote the start times of the lth executions of the unload and load actions of A_{ij}, $i = 1, 2, j = 1, \ldots, c_i$, respectively. For any (π_1, π_2) and l, there are four constraints governing the activities performed in the double-space buffer

$$S^U_{20l} \geq S^L_{1,b_1-1,l} + \epsilon_1, \quad S^L_{1,b_1-1,l+1} \geq S^U_{20l} + \epsilon_2,$$
$$S^U_{1b_1l} \geq S^L_{2c_2l} + \epsilon_2, \quad S^L_{2,c_2,l+1} \geq S^U_{1b_1l} + \epsilon_1.$$

The four constraints that enforce the single-space buffer are identical to the above four constraints. All the rest of the constraints enforcing the actions of other activities are also equivalent for both cases under the same schedule. As a result, the cycle time of the double-space buffer under $(n_1 - 1, n_2)$ is the same as that of the single-space buffer case.

Part (b): In the case of n_1 wafers in $\mathbb{C}_1$, an important fact is that when R_1 loads a wafer into $P_{1b_1:1}$, R_2 cannot unload this wafer unless it loads another finished wafer into $P_{1b_1:2}$. Therefore, the cycle time can be computed based on the following possible bottlenecks: (i) $\mathbb{C}_1$ (but not $P_{1b_1:1}$ or $P_{1b_1:2}$); (ii) $\mathbb{C}_2$; (iii) $P_{1b_1:1}$ or $P_{1b_1:2}$; and (iv) the n-CRM cycle. For (i), R_1 never waits at P_{1b_1}. Therefore, the processing time at P_{1b_1} in all resource cycles can be replaced by zero, that is, $T_1(\pi_1)$. In (ii), $T_2(\pi_2)$ dominates all resource cycle times of $\mathbb{C}_1$, including those that contain A_{1b_1}. To analyze (iii), first note again that since R_2 cannot unload a wafer from $P_{1b_1:1}$ (P_{20}) until it loads another finished wafer into $P_{1b_1:2}$. If R_1 needs to wait to unload a wafer from $P_{1b_1:2}$, then this is the same case in (ii). Therefore, under (iii), after R_1 loads a wafer to $P_{1b_1:1}$, there is already a wafer in $P_{1b_1:2}$ ready for pick up. Similarly, after R_1 unloads a wafer from $P_{1b_1:2}$, there is already a space in $P_{1b_1:1}$ ready for holding a wafer. As a result $t_{v_1} = 0$. Finally, we show that the n-CRM cycle never dominates. As discussed in the proof of Theorem 2, the n-CRM cycle time is the total time for a wafer to go through $\mathbb{C}_2$ in addition to the time it takes for R_1 to return a wafer to $P_{1b_1:1}$ after it unloads one from $P_{1b_1:2}$. However, after R_2 loads a wafer to $P_{1b_1:2}$, there is already a wafer in $P_{1b_1:1}$ ready for pick up [otherwise, it becomes either Cases (i) or (iii)]. Therefore, unlike Theorem 2, the time it takes for P_{1b_1} to have a wafer back after a wafer departs is zero. Therefore, the n-CRM cycle time cannot dominates.

Part (c): The n_1+1 wafers in $\mathbb{C}_1$ also results in $t_{v_1} = 0$ by the similar proofs for Cases (iii) and (iv) in Part (b). Therefore, all the resource cycle times are equal to those of the (n_1, n_2)-distribution case, except one additional possible bottleneck resource cycle created by the congestion at the buffer. This bottleneck can also be identified using the Gantt chart. Since there are $n_1 + 1$ wafers in $\mathbb{C}_1$, there is at least one wafer, say wafer A, in P_{1b_1}. Without loss of generality, we assume wafer A is in $P_{1b_1:2}$; see Fig. 7. The wafer flow congestion starts when R_1 loads a wafer, say wafer B, to $P_{1b_1:1}$ and at the same time R_2 is waiting to load a finished wafer, say wafer C, into $P_{1b_1:2}$. The time for $P_{1b_1:2}$ to be cleared (wafer A) starting from the time at which R_1 begins to load wafer B into $P_{1b_1:1}$ is equal to the time for R_1 to perform the following activities: 1) load wafer B to $P_{1b_1:1}$ (ϵ_1); 2) all activities between A_{1b_1-1} to A_{1b_1} (excluding A_{1b_1-1} and A_{1b_1}); and 3) unload wafer A from $P_{1b_1:2}$ (ϵ_1). Suppose the schedule is $\pi_1 = (A_{10}A_{1j_1}, \ldots, A_{1b_1}, \ldots, A_{1,b_1-1}, \ldots, A_{1j_{c_1}})$ (the case where A_{1,b_1-1} precedes A_{1b_1} is the same since the schedule repeats itself), then the time for the second part above is equal to the time for R_1 to perform all activities (from A_{10} to A_{1c_i}) minus the time to perform the activities in the resource cycle of P_{1b_1} (from A_{1b_1} to A_{1,b_1}, with $t_{v_1} = 0$ since wafer A is already finished at $P_{1b_1:2}$ when the cycle starts; otherwise the cycle cannot repeat), that is, $\mathrm{RCT}(R_1) - \mathrm{RCT}^0(P_{1b_1})$; see the solid flow line in Fig. 7.

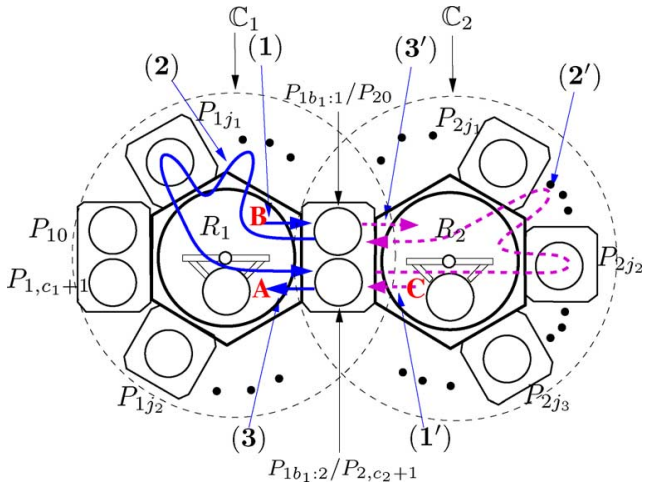

Fig. 7. Congestion under the wafer distribution $(n_1 + 1, n_2)$.

This congested resource cycle continues when R_2 starts loading wafer C to $P_{1b_1:2}$ and ends when it unloads wafer B from $P_{1b_1:1}$ (so that the cycle can repeat). The time for $P_{1b_1:1}$ to be cleared (wafer B) starting at the time at which R_2 begins to load wafer C to $P_{1b_1:2}$ is equal to the time for R_2 to perform the following activities: $(1')$ load the wafer to $P_{1b_1:2}$ (ϵ_2), $(2')$ all activities between A_{2c_2} to A_{20} (excluding A_{2c_2} and A_{20}), and $(3')$ unload the wafer from $P_{1b_1:1}$ (ϵ_2). Suppose the schedule is $\pi_2 = (A_{20}A_{2j_1}, \ldots, A_{2c_2}, \ldots, A_{2j_{c_2}})$, then the time for the second part $(2')$ is equal to the time for R_2 to perform all activities (from A_{20} to $A_{2j_{c_2}}$) minus the time to perform the activities in the resource cycle of P_{20} (from A_{20} to A_{2c_2}, excluding the moving time to P_{2c_2} since R_2 is already there when the cycle starts; otherwise, the cycle cannot repeat), that is, $\mathrm{RCT}(R_2) - \mathrm{RCT}(P_{20})$; see the dash flow line in Fig. 7. In summary, the total time of this resource cycle is $\mathrm{RCT}(R_1) - \mathrm{RCT}^0(P_{1b_1}) + \mathrm{RCT}(R_2) - \mathrm{RCT}(P_{20}) + 2(\epsilon_1 + \epsilon_2)$. This completes the proof. ■

APPENDIX B
PROOF OF THEOREM 4

The proof is by constructing a polynomial-time algorithm, called `OptTwoCluster`. The algorithm finds an optimal schedule for a two-cluster tool by iteratively searching optimal schedules for the two single clusters individually until a best combined schedule is found. During the search process, `OptTwoCluster` uses the optimal algorithm developed in [7] to find the optimal schedule for each of the two single clusters. For convenience, we call this optimal one-unit cycle algorithm in [7] as `OptSingleCluster`. Moreover, same as in [7], we assume that all timing data are integers.

According to Theorem 3, the double-space buffer case can be easily solved by finding the optimal schedule for each single cluster independently using `OptSingleCluster` and setting the wafer distribution as (n_1, n_2). Therefore, we focus on the single-space buffer case in the following.

As in Theorem 2, we define two sets of consecutive activity indexes, $\mathrm{CS}_a(p) = \{1, 2, \ldots, p\}$ and $\mathrm{CS}_b(q) = \{q, q+1, \ldots, c_2\}$. If $p = 0$, the $\mathrm{CS}_a(p) = \varnothing$. We also define two decision questions.

Decision Question 1 (DQ1): Given L, n', and T_2^F, does there exist a one-unit cycle π_1 with $T_1(\pi_1) \le L$:

a) if $b_1 \in \Lambda_1^P$, $\mathrm{RCT}(P_{1b_1}) \le n'L - T_2^F$, or,
b) if $b_1 \in \Lambda_1^R$, $\max\{\mathrm{RCT}(P_{1k_L}), \mathrm{RCT}(P_{1k_R}), \mathrm{RCT}(R_1)\} \le n'L - T_2^F$?

Decision Question 2 (DQ2): Given L, n', $\mathrm{CS}_a(p)$, and $\mathrm{CS}_b(q)$, does there exist a one-unit cycle π_2 such that $T_2(\pi_2) \le L$ and $n_2(\pi_2) = n'$?

These two decision questions are modifications of the decision question (DQ) in [7]: *Given L, does there exist a one-unit cycle π with $T(\pi) \le L$?* In [7], DQ is addressed by solving a sequence of shortest path problems on some directed acyclic graphs (DAGs). The DAGs are constructed based on the timing data of cluster tools. DQ1 and DQ2 can be addressed by modifying the construction process of those DAGs and solving the shortest path problems in the modified DAGs. Here, we only provide the modifications of those DAGs.

For DQ1, the data used to construct the DAGs is $\mathbf{t}_1 = (t_{11}, \ldots, t_{v_1}, \ldots, t_{1c_1}; \epsilon_1, \delta_1)$. Similar to [7], let $D_{\delta_1} = \{i \,|\, i \in \{1, \ldots, c_1\}, t_{1i} \ge \delta_1\}$ denote the set of PMs with a processing time greater than the robot moving time. If $t_{v_1} \ge \delta_1$, then $b_1 \in D_{\delta_1}$ and the construction follows Case 1 in the following. If $t_{v_1} < \delta_1$, then $b_1 \notin D_{\delta_1}$ and P_{1b_1} could be selected in Λ_1^P or Λ_1^R (both Cases 1 and 2 are needed). We construct a DAG for each of these two cases. We use and refer to Construction Steps 1–4 in [7] in the following.

Case 1: P_{1b_1} is selected in Λ_1^P. Add condition $\mathrm{RCT}(P_{1b_1}) \le n'L - T_2^F$ when adding edges in Construction Steps 2–4.

Case 2: P_{1b_1} is selected in Λ_1^R. Add condition $\max\{\mathrm{RCT}(P_{1k_L}), \mathrm{RCT}(P_{1k_R}), \mathrm{RCT}(R_1)\} \le n'L - T_2^F$ when adding edges in Construction Steps 2–4.

These modified conditions ensure that not only the selected schedule has a cycle time less or equal to L, but also the n-CRM cycle term is less than or equal to L. To find such a schedule, we solve a shortest path problem on each of the DAGs. It is straightforward (and omitted here) to extend the proof in [7] to verify Procedure `AddressingDQ1` given here for finding a schedule satisfying the condition in DQ1.

Procedure `AddressingDQ1`**.**

1. Construct the DAGs as in DQ under Case 1. Answer the DQ by solving the shortest path problem on each of the DAGs. If yes, exit and return "yes";
2. If $t_{v_1} < \delta_1$, construct DAGs as in DQ under Case 2. Answer the DQ by solving the shortest path problem on each of the DAGs. If yes, exit and return "yes"; otherwise, return "no."

For DQ2, only one DAG, called Full_DAG, is needed and the data used to create it is $\mathbf{t}_2 = (t_{21}, \ldots, t_{2c_2}; \epsilon_2, \delta_2)$. The construction of the DAGs is the same as in [7] except that we change the length of all edges (except the edges to the sink node) corresponding to the nodes in $\mathrm{CS}_a(p)$ and $\mathrm{CS}_b(q) \setminus \{q\}$ to L. This modification ensures that `OptSingleCluster` will not select the nodes in $\mathrm{CS}_a(p)$ and $\mathrm{CS}_b(q) \setminus \{q\}$ to be in Λ_1^P. Since there is only one DAG, the initial set of Λ_2^P and Λ_2^R is set to $\{q\}$ and $\{1, \ldots, c_2\} \setminus \{q\}$, respectively.

The following Procedure `AddressingDQ2` for answering DQ2 is to find a schedule π_2 that satisfies $T_2(\pi_2) \le L$ and $n(\pi_2) = n_2'$ by solving a constrained (n-stop) shortest path problem on Full_DAG. The n-stop shortest path problem is to find the shortest path containing exactly n intermediate nodes between a source and a destination nodes.

Procedure `AddressingDQ2`**.**

1. If $n_2' = 1$, exit and return "yes" if $T_2^F \le L$ and "no" otherwise;
2. Construct Full_DAG as described above. Let $n = n_2' + 1$. Solve the n-stop shortest path problem for Full_DAG. If the length of the n-stop shortest path is less than or equal to $L - \mathrm{RCT}(R_2)$, where $\mathrm{RCT}(R_2)$ is computed using $\Lambda_2^P = \{q\}$ and $\Lambda_2^R = \{1, \ldots, c_2\} \setminus \{q\}$, then exit and return "yes"; otherwise, return "no."

As shown in [7], selecting a node in the shortest path means changing the corresponding PM from Λ_2^R to Λ_2^P. Since initially $\Lambda_2^P = \{q\}$ and there are one source node and one destination nodes, $n = n_2' - 1 + 2 = n_2' + 1$ is the number of additional nodes we need to have n_2' wafers in $\mathbb{C}_2$. If there is an n-stop shortest path of length less than or equal to $L - \mathrm{RCT}(R_2)$, then DQ2 has a positive answer and the schedule π_2 includes $\Lambda_2^P = \{q\} \cup \mathbf{N}$ and $\Lambda_2^R = \{1, \ldots, c_2\} \setminus \Lambda_2^P$, where $\mathbf{N}$ is the set of nodes (PMs) contained in the n-stop shortest path. The proof of this argument is similar to that in [7] and omitted here. The

algorithm for finding the optimal one-unit schedule (π_1^*, π_2^*) for the two-cluster tool is given in the following procedure.

Procedure `OptTwoCluster`**.**

1. Use `OptSingleCluster` to find the optimal schedules for decoupled $\mathbb{C}_1$ and $\mathbb{C}_2$ with $t_{v_1} = T_2^F$, yielding π_1^1 and π_2^1 with cycle times $T_1(\pi_1^1)$ and $T_2(\pi_2^1)$, respectively. Set $\mathrm{UB} = \max\{T_1(\pi_1^1), T_2(\pi_2^1), K_{12}\}$;
2. Use `OptSingleCluster` to find the optimal schedule for decoupled $\mathbb{C}_1$ with $t_{v_1} = \beta_2^{\delta_2} + \beta_2$, yielding π_1^2 and $T_1(\pi_1^2)$. Set $\mathrm{LB} = \max\{T_1(\pi_1^2), T_2(\pi_2^1)\}$ and $L = (\mathrm{LB} + \mathrm{UB})/2$;
3. Set $n_2' = 1$ and $p = c_2$. Answer DQ1 for L, n_2', T_2^F and $t_{v_1} = \beta_2^{\delta_2} + \sum_{l=1}^{p} \alpha_{2l}$. If yes, set π_1^* to the schedule found in `AddressingDQ1`; otherwise goto 5;
4. Answer DQ2 for $L, n_2', \mathrm{CS}_a(p)$ ($\mathrm{CS}_b(q)$ is not needed for $n_2' = 1$). If yes, set π_2^* to the schedule found in `AddressingDQ2` (and $n_2' = n_2(\pi_2^*)$) and goto 11;
5. Set $n_2' = n_2' + 1$, $p = 0$, and $q = n_2' + p$. If $n_2' > c_2$, goto 12;
6. Answer DQ1 for L, n_2', T_2^F, with $t_{v_1} = \beta_2^{\delta_2} + \sum_{l=1}^{p} \alpha_{2l} + \beta_2 + \sum_{l=q_2+1}^{c_2} \alpha_{2l}$. If yes, set π_1^* to the schedule found in `AddressingDQ1` and goto 9;
7. Set $q = q + 1$. If $q \leq c_2$, goto 6.;
8. Set $p = p + 1$ and $q = n_2' + p$. If $p \leq c_2 - n_2'$, goto 6; otherwise, goto 5;
9. Answer DQ2 for $L, n_2', \mathrm{CS}_a(p), \mathrm{CS}_b(q)$. If yes, set π_2^* to the schedule found in `AddressingDQ2` (and $n_2' = n_2(\pi_2^*)$), goto 11;
10. Set $q = q + 1$. If $q \leq c_2$, goto 9; otherwise, goto 8;
11. Set $\mathrm{UB} = \max\{T_1(\pi_1^*), T_2(\pi_2^*), K_{12}\}$ and $L = (\mathrm{LB} + \mathrm{UB})/2$. If $\mathrm{UB} - \mathrm{LB} < (1/c_2)$, terminate; otherwise, goto 3;
12. Set $\mathrm{LB} = L$ and $L = (\mathrm{LB}+\mathrm{UB})/2$. If $\mathrm{UB}-\mathrm{LB} < (1/c_2)$, terminate; otherwise, goto 3.

The validity of `OptTwoCluster` is illustrated as follows. We first find the optimal schedules π_1^1 and π_2^1 for $\mathbb{C}_1$ and $\mathbb{C}_2$, respectively, with $t_{v_1} = T_2^F$ and ignoring K_{12}. Since $t_{v_1} = T_2^F$ is the maximum virtual processing time, the maximum of the three terms in Step 1 gives an upper bound on the whole cycle time. Similarly, Step 2 finds the lower bound by using the minimum virtual processing time. If there exists an optimal cycle time less than the upper bound, it must be greater than $\max\{T_1(\pi_1^2), T_2(\pi_2^1)\}$ because $T_1(\pi_1^2)$ and $T_2(\pi_2^1)$ are the minimum of $T_1(\pi_1)$ and $T_2(\pi_2)$, respectively. Next, a binary search is performed over the interval [LB, UB]. The interaction between $\mathbb{C}_1$ and $\mathbb{C}_2$ is realized in three factors: p, q, and n_2. Therefore, Steps 5–8 find a triplet (p, q, n_2) that gives a schedule π_1 that satisfies $T_1(\pi_1) \leq L$ and $K_{12} \leq L$. When a (p, q, n_2) is found, it must be checked (Steps 9 and 10) to give a π_2 such that $T_2(\pi_2) \leq L$ and $n_2(\pi_2) = n_2$. If this is not satisfied, search again on the remaining values of the triplet (p, q, n_2). Observe that incrementing q reduces t_{v_1}; therefore, keeping $T_1(\pi_1) \leq L$. Finally, since all data is integer, the minimum difference between two cycle times is bounded by $(1/c_2)$, the smallest difference between two n-CRM cycle times. Therefore, the binary search will terminate if the search interval is less than this bound.

The complexity of `OptSingleCluster` is polynomial [7]. DQ1 involves at most two executions of DQ, each can be solved using a shortest path algorithm, for example, the Dijkstra's algorithm in $O(c_1^2)$. DQ2 can be solved using an n-stop shortest path algorithm, for example, [27] in a polynomial running time. The loop of Steps 5–10 has at most c_2^3 iterations. Finally, the number of binary search is bounded by the logarithm of the minimum difference between the upper and lower bounds of the cycle times, which is $1/c_2$. Consequently, `OptTwoCluster` is polynomial. This completes the proof.

REFERENCES

[1] D. Jevtic, "Method and aparatus for managing scheduling a multiple cluster tool," European Patent 1,132,792 (A2), Dec. 2001.

[2] 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, 1994.

[3] 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, 1997.

[4] Y. Crama and J. van de Klundert, "Cyclic scheduling of identical parts in a robotic cell," *Opers. Res.*, vol. 45, no. 6, pp. 952–965, 1997.

[5] Y. Crama, V. Kats, J. van de Klundert, and E. Levner, "Cyclic scheduling of robotic flowshops," *Anns. Oper. Res.*, vol. 96, no. 1, pp. 97–124, 2000.

[6] M. Dawande, H. Geismar, S. Sethi, and C. Sriskandarajah, *Throughput Optimization in Robotic Cells*. New York: Springer, 2007.

[7] M. Dawande, C. Sriskandarajah, and S. Sethi, "On throughtput maximization in constant travel-time robotic cells," *Manuf. Serv. Oper. Manage.*, vol. 4, no. 4, pp. 296–312, 2002.

[8] 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.

[9] Q. Su and F. Chen, "Optimal sequencing of double-gripper granty robot moves in tightly-coupled serial production systems," *IEEE Trans. Robot. Automat.*, vol. 12, no. 1, pp. 22–30, 1996.

[10] S. Sethi, J. Sidney, and C. Sriskandarajah, "Scheduling in dual gripper robotic cells for productivity gains," *IEEE Trans. Robot. Automat.*, vol. 17, no. 3, pp. 324–341, 2001.

[11] C. Sriskandarajah, I. Drobouchevitch, S. Sethi, and R. Chandrasekaran, "Scheduling multiple parts in a robotic cell served by a dual-gripper robot," *Oper. Res.*, vol. 52, no. 1, pp. 65–82, 2004.

[12] M. Dawande, H. Geismar, S. Sethi, and C. Sriskandarajah, "Sequencing and scheduling in robotic cells: Recent developments," *J. Scheduling*, vol. 8, pp. 387–426, 2005.

[13] N. Geismar, M. Dawande, and C. Sriskandarajah, "Approximation algorithms for k-unit cyclic solutions in robotic cells," *Eur. J. Oper. Res.*, vol. 162, pp. 291–309, 2005.

[14] S. Kuma, N. Ramanan, and C. Sriskandarajah, "Minimizing cycle time in large robotics cell," *IIE Trans.*, vol. 37, no. 2, pp. 123–136, 2005.

[15] D. Jevtic and S. Venkatesh, "Method and apparatus for scheduling wafer processing within a multiple chamber semiconductor wafer processing tool having a multiple blade robot," U.S. Patent 6,224,638, May 2001.

[16] 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, 2004.

[17] S. Ding, J. Yi, and M. T. Zhang, "Multi-cluster tools scheduling: An integrated event graph and network model approach," *IEEE Trans. Semiconduct. Manufact.*, vol. 19, no. 3, pp. 339–351, 2006.

[18] J. Yi, S. Ding, D. Song, and M. T. Zhang, "Steady-state throughput and scheduling analysis of multi-cluster tools: An decomposition approach," *IEEE Trans. Automat. Sci. Eng.*, vol. 5, no. 2, pp. 321–336, 2008.

[19] W.-K. Chan, J. Yi, and S. Ding, "On the optimality of one-unit cycle scheduling of multi-cluster tools with single-blade robots," in *Proc. IEEE Int. Conf. Auto. Sci. Eng.*, Scottsdale, AZ, 2007, pp. 392–397.
[20] W.-K. Chan, J. Yi, S. Ding, and D. Song, "Optimal scheduling of k-unit production of cluster tools with single-blade robots," in *Proc. IEEE Int. Conf. Auto. Sci. Eng.*, Washington, DC, 2008, pp. 335–340.
[21] W.-K. Chan, S. Ding, J. Yi, and D. Song, "Optimal scheduling of multi-cluster tools with constant robot moving times, Part II: Tree-like topology configurations," *IEEE Trans. Automat. Sci. Eng.*, vol. 8, no. 1, pp. 17–28, Jan. 2011.
[22] W. Zuberek, "Timed Petri nets in modeling and analysis of cluster tools," *IEEE Trans. Robot. Automat.*, vol. 17, no. 5, pp. 562–575, 2001.
[23] N. Wu, C. Chu, F. Chu, and M. Zhou, "A Petri net method for schedulability and scheduling problems in single-arm cluster tools with wafer residency time constraints," *IEEE Trans. Semiconduct. Manufact.*, vol. 21, no. 2, pp. 224–237, 2008.
[24] J.-H. Kim and T.-E. Lee, "Schedulability analysis of time-constrained cluster tools with bounded time variation by an extended Petri net," *IEEE Trans. Automat. Sci. Eng.*, vol. 5, no. 3, pp. 490–503, 2008.
[25] W.-K. Chan, "Mathematical programming representations of discrete-event system dynamics," Ph.D. dissertation, Dept. Ind. Eng. Oper. Res., Univ. California, Berkeley, CA, 2005.
[26] A. Che and C. Chu, "Cyclic hoist scheduling in large real-life electroplating lines," *OR Spectrum*, vol. 29, pp. 445–470, 2007.
[27] M. Terrovitis, S. Bakiras, D. Papadias, and K. K. Mouratidis, "Constrained shortest path computation," in *Proc. Int. Symp. Spat. Temp. Databases (SSTD)*, C. B. Medeiros, M. J. Egenhofer, and E. Bertino, Eds., Angra dos Reis, Brazil, 2005, pp. 181–199.

Wai Kin Victor Chan (M'05) received the B.Eng. degree from Shanghai Jiao Tong University, Shanghai, China, the M.Eng. degree in electrical engineering from Tsinghua University, Beijing, China, and the M.S. and Ph.D. degrees in industrial engineering and operations research from the University of California, Berkeley, in 2001 and 2005, respectively.

He is an Assistant Professor with the Department of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY. His research interests include discrete-event simulation, agent-based simulation, and their applications in energy markets, social networks, service systems, and manufacturing.

Jingang Yi (S'99–M'02–SM'07) received the B.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 1993, the M.Eng. degree in precision instruments from Tsinghua University, Beijing, China, in 1996, and the M.A. degree in mathematics and the Ph.D. degree in mechanical engineering from the University of California, Berkeley, in 2001 and 2002, respectively.

He is currently an Assistant Professor of Mechanical Engineering at Rutgers University. Prior to joining Rutgers University in August 2008, he was an Assistant Professor of Mechanical Engineering at San Diego State University since January 2007. From May 2002 to January 2005, he was with Lam Research Corporation, Fremont, CA, as a member of Technical Staff. From January 2005 to December 2006, he was with the Department of Mechanical Engineering, Texas A&M University, as a Visiting Assistant Professor. His research interests include autonomous robotic systems, dynamic systems and control, mechatronics, automation science and engineering, with applications to semiconductor manufacturing, intelligent transportation and biomedical systems.

Dr. Yi is a member of the American Society of Mechanical Engineers (ASME). He is a recipient of the NSF Faculty Early Career Development (CAREER) Award in 2010. He has coauthored papers that have been awarded the Best Student Paper Award Finalist of the 2008 ASME Dynamic Systems and Control Conference, the Best Conference Paper Award Finalists of the 2007 and 2008 IEEE International Conference on Automation Science and Engineering, and the Kayamori Best Paper Award of the 2005 IEEE International Conference on Robotics and Automation. He currently serves as an Associate Editor of the ASME Dynamic Systems and Control Division and the IEEE Robotics and Automation Society Conference Editorial Boards. He also serves as a Guest Editor of IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING.

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

He is currently with the Direct Store Delivery (DSD) Division of Nestle USA as a member of the supply chain integration (SCI) team. From 2007 to 2009, he worked at Leachman and Associates LLC, a firm providing consulting and software for operations management to semiconductor manufacturers and other corporations. Prior to that, he was a Postdoctoral Fellow at the University of California, Berkeley. His research interests include optimization, queueing models, simulation, planning and scheduling, production management, and semiconductor manufacturing.