Using Logical Time to Ensure Liveness in Material Handling Systems With Decentralized Control

Zäzilia Seibold , Kai Furmans, and Kevin R. Gue, *Member, IEEE*

***Abstract*— We describe a method for decentralized control of route-based material handling systems in which devices have no central controller (by definition), no common source of information, and no synchronized or common clocks with which to plan and execute their activities. The control scheme is based on the concept of logical time, which is a means of partially ordering events in computer operating systems. We modify the concept to the domain of material handling systems and prove system liveness. We conclude by describing GridSorter, a conveyance-based sorter that uses decentralized control and logical time to sort packages. A prototype has been successfully built and tested at the Institute for Material Handling and Logistics, Karlsruhe Institute of Technology.**

***Note to Practitioners*—Decentralized control is a means of distributing the control of a complex system away from a central computing source toward individual devices and subsystems. In material handling, complex systems can malfunction due to deadlock, livelock, or starvation. This article presents a new method for controlling decentralized material handling systems based on logical time, a method from computer operating systems describing a sequence of activities at each resource. We modify logical time for material handling and show that the system is deadlock-free, thus giving engineers an easy-to-implement method of controlling route-based material handling systems such as automated guided vehicles (AGVs) or conveyors.**

***Index Terms*— Decentralized control, logical time, material handling.**

I. Decentralized Control in Material Handling

An IMPORTANT feature of the Industry 4.0 concept in material handling systems is autonomy, the ability of devices to make independent decisions. When autonomous devices must cooperate to get things done without a central controller, we say that the system has decentralized control.

Manuscript received August 27, 2019; revised June 7, 2020; accepted July 22, 2020. This article was recommended for publication by Editor B. Vogel-Heuser upon evaluation of the reviewers' comments. This work was supported in part by the Institute for Material Handling and Logistics at Karlsruhe Institute of Technology and in part by several projects financed through "Zentrales Innovationsprogramm Mittelstand" of "Bundesministerium für Wirtschaft und Energie (BMWI)." *(Corresponding author: Zäzilia Seibold.)*

Zäzilia Seibold and Kai Furmans are with the Department of Mechanical Engineering, Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, Karlsruhe 76131, Germany (e-mail: zaezilia.seibold@partner.kit.edu; kai.furmans@kit.edu).

Kevin R. Gue is with the Department of Industrial Engineering, University of Louisville, Louisville, KY 40292 USA (e-mail: kevin@kevingue.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/TASE.2020.3029199, provided by the authors.

Digital Object Identifier 10.1109/TASE.2020.3029199

The problem of decentralized control in material handling has been addressed by several authors for automated guided vehicle (AGV)- or vehicle-based systems and recently by a few authors for conveyance-based systems. Most of these articles assume some form of route-based planning in which, for example, an AGV travels a predetermined route from origin to destination. By "route," we mean a path through a network of zones, usually represented by a graph. A fundamental challenge of operating such systems is timing the movement of entities (AGVs, boxes) among the shared resources (zones and conveyors) such that the entities are transported when and where required without sacrificing system liveness—that is, without causing deadlocks, livelocks, or starvation.

Tanenbaum and Bos [1] described four strategies for dealing with deadlocks in operating systems: ignoring the problem, detection and recovery, dynamic avoidance, or prevention. Of these possibilities, we address the structural prevention of deadlocks by applying and adapting the principle of logical time [2], which is a means of partial ordering of events in distributed computer operating systems. We modify the concept from computer science in order to sequence activities on shared physical resources.

This allows the system to be independent of synchronized (physical) clocks within the system, which represents a significant departure from prior work that assumes a central clock to coordinate activities between decentralized control units. By means of structural deadlock prevention, we create an environment in which the degree of decentralization is as high as possible and the communication load is low.

Our article is structured as follows. In Section II, we introduce articles in the field of material handling systems with decentralized control or route-based planning algorithms. Section III is a formal description of the system, and Section IV introduces the principle of logical time that serves as a base for Section V where we transfer logical time to material handling systems. After the proof in Section VI, we conclude in Section VII by describing a prototype system called GridSorter, a grid-based conveyor system that uses the logical time to sort packages. The functionality of its control algorithm has been shown and evaluated with simulation and on a real demonstration system.

II. Literature Review

We observe an equivalence between AGVs moving through a network of zones and boxes being transported by a network of conveyors: the general problem is routing entities in a network of resources under the assumption that a resource can possess only one entity at a time [3] unless it is transferring the entity to another resource, during which time both resources are busy with the same entity (this subtlety is usually unmentioned). Therefore, we divide the literature of decentralized control for material handling into articles addressing conveyor-based systems and articles addressing AGV- or vehicle-based systems. We focus only on articles that handle resource conflicts potentially resulting in deadlocks.

The term decentralized control, sometimes called autonomous control [4], is used differently in the literature. All articles assume decentralized decision making, but there are differences with respect to information sharing and time coordination. In this article, we address systems with decentralized decision-making with distributed, local synchronization, and no general broadcasting [5]. In most of the articles, the means of time coordination is implied rather than stated explicitly.

Methods for conveyor-based systems can be further divided into negotiation-based algorithms, which move items in a stepwise manner, and reservation-based algorithms, which determine a module-by-module path from source to destination. Mayer and Furmans [5] proposed a reservation-based algorithm to route boxes within a network of small conveyor modules (called FlexConveyors). Each module has its own control and communicates with its neighbors to make decisions. For each box entering the system, a route is reserved from source to destination in order to prevent opposing routes on bidirectional conveying modules. The system uses "deadlock tokens" to ensure deadlock-free operations, a property proven by the authors. Because FlexConveyor is intended for sparse conveyor networks and not for more general dense networks, the deadlock-avoidance mechanism does not address an unlimited number of interlocking circular routes.

FlexConveyors also form the basis of grid-based material handling systems such as GridStore [6], GridPick [7], and GridSequence [8], which embody the negotiation-based decentralized control scheme. Adjacent units in the grid negotiate by passing messages in order to determine what action to take next. GridStore was proven deadlock-free; GridPick was shown to be deadlock-free under certain conditions. These systems assume time synchronization.

Krühn *et al.* [9], [10] described small-scaled conveyor modules called "cognitive conveyors," which feature decentralized control and route-based planning. These systems are distinctive in that many modules work together to transport a single box. The system applies a deadlock-avoidance algorithm requiring a high number of messages but does not require synchronization of clocks.

Because our method is route-based, we provide a brief overview of route-based planning for AGV systems. Kim and Tanchoco [11] described a centralized algorithm using time-window-based routing. Maza and Castagna [12] presented a time-window-based algorithm based on [11] which they call sequence-based conflict-free routing because deadlocks are prevented as long as resources respect the sequence of vehicles visiting them. In [13], they propose different methods for changing this priority if delays occur in order to reduce the impact of such incidences. All the articles described in this paragraph use centralized control, but they highlight an important feature of route-based solutions that we also exploit; in the presence of variability, respecting the sequence of activities on a resource maintains deadlock-free operation.

To our knowledge, there exist only two control algorithms using decentralized decision-making with distributed, local synchronization, and no general broadcasting: the method of [5], which is only applicable to spare networks with low risk of deadlock, and that of [9], which uses a complex deadlock-avoidance algorithm with high communication effort before each transport and cannot easily be applied to different system conditions and tasks. Our method improves on prior work by addressing more general system architectures using structural deadlock prevention.

III. System Description and Assumptions

Because our most complex application domain is a dense grid of unit-sized conveyors being able to transport boxes to its four neighbors, we assume that resources are conveyors and entities are boxes for the rest of the discussion. Bigger conveyors that can hold more than one box concurrently are virtually divided into several unit-sized conveyors. The conveyor modules can be of linear or curved shape, as long as they can be modeled as unit-sized conveyors with one-to-four transport directions. Modules with nonopposing directions are able to perform direction changes.

A system of conveyor modules and boxes can be considered a distributed system as follows; the transport of a box from source to destination is a process. When the box is being conveyed, the corresponding process is in state *Transport*; otherwise, it is in state *Hold and Wait*. The process starts when the box is inserted in the system and the transport steps from one conveyor module to the next are events in the process. The process is finished when the box reaches its destination.

Conveyor modules are resources in the system. A box must be conveyed by two conveyor modules concurrently to be transported from one module to the next. Thus, the corresponding process in state *Transport* must hold both resources. The duration of this transport phase is not defined but finite. When the box is stationary, it remains on one conveyor module. Thus, a process in state *Hold and Wait* only needs to hold only one resource, although it will have already requested the resource it needs for the next transport step.

Each resource can host at most one entity at any time, a "tandem movement" is executed in order to increase system throughput. Having three aligned conveyor modules, the middle module can receive a box from one side and send a box to the opposite side concurrently. This resource is assigned to two processes.

Resources are allowed to communicate with their direct neighbors. One important difference between any material handling system and a distributed multiprocess system is that the time required for physical movement is many times higher than the time needed to share and process information. This makes it possible to finish a reservation process with a high number of sent messages before starting the first transport of a box.

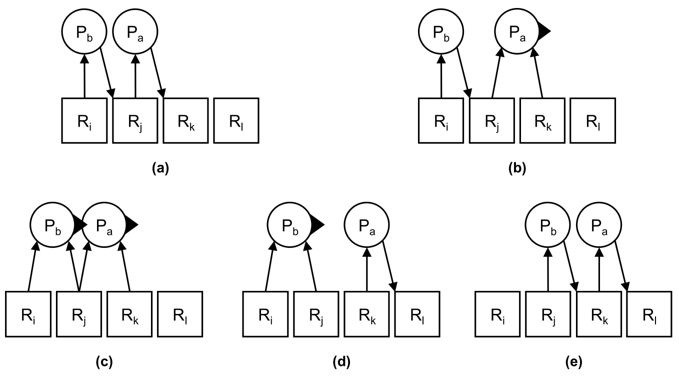

Fig. 1. Acquisition of resources for tandem transport. (a) Both boxes are waiting. (b) Box *a* is being transported. (c) Boxes *a* and *b* are transported in tandem. (d) Box *a* has arrived while box *b* is still being transported. (e) Both boxes have arrived.

The graphical representation of processes, resources, and possible relations between them is based on the representation of Tanenbaum and Bos [1]. They describe a sequence of actions required to use a resource as follows: request, use, and release the resource. In our system, resources need to be reserved before requesting them.

Fig. 1 shows the course of acquisition of resources for the more complex case of tandem transport. The resources have been reserved already. In Fig. 1(a), process P_b is requesting resource R_j, which is this time already held by process P_a. Process P_a, in turn, has requested resource R_k. Tandem transport is physically possible because the trio of resources R_i, R_j, and R_k is aligned. Once resource R_k is granted to P_a, P_a switches to state *Transport* [see Fig. 1(b)]. Resource R_j can now be granted to process P_b, while it is still held by P_a [see Fig. 1(c)]. P_b switches to state *Transport* as well. Transition to Fig. 1(d) is triggered when process P_a switches to *Hold and Wait* after a finite physical time interval. In addition, process P_b switches to *Hold and Wait* and requests the next resource after a finite physical time interval [see Fig. 1 (e)].

To summarize: the transition from state *Hold and Wait* to *Transport* takes place when granting conditions (GCs) are fulfilled. A resource can only be granted to a process if it is free, i.e., not held by another process, or if tandem transport conditions are fulfilled. The transition from *Transport* to *Hold and Wait* is time-triggered because the transport has a finite duration in physical time.

IV. Lamport's Logical Time

A distributed system consists of multiple parallel processes, each comprising an ordered set of events (graphically represented in Fig. 2). This order defines causal relations between events (dots) of one process (vertical lines). Lamport [2] referred to a causal relation as a "happening before" relation. Parallel processes are connected to each other by messages (wavy arrows) defining a causal relation between events of two different processes; the event of sending a message happens before the event of receiving it. The entire set of causal relations forms a partial ordering of the events in the system. In Fig. 2, event p_1 happens before q_3 because the events are connected through the sequence of causal relations via q_2. However, the ordering of events p_3 and q_3 cannot be determined because there is no sequence of causal relations.

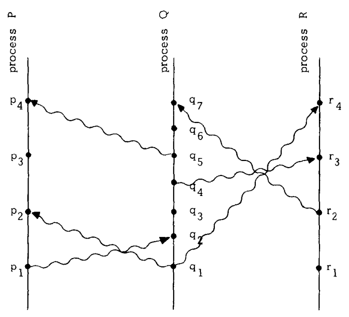

Fig. 2. Distributed system: parallel processes with multiple events and causal relations, from [2].

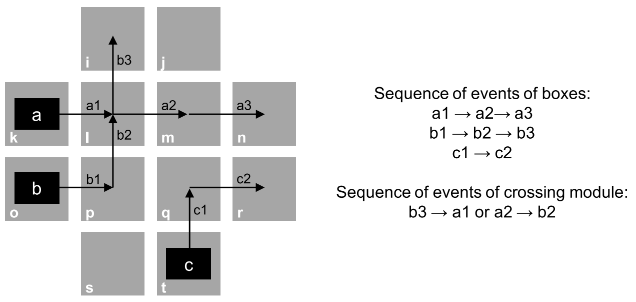

Fig. 3. Example routes for three boxes.

Lamport [2] assigned a number to each event representing the logical time at which the event occurs. C_i is the logical clock of process P_i and $C_i\langle a\rangle$ is the logical time of event a if it is an event of process P_i. The set of logical clocks of all processes is represented by C, where $C\langle a\rangle = C_i\langle a\rangle$ if a is an event of process P_i. Lamport formulates the clock condition as follows. For any events a and b, if $a \to b$, then $C\langle a\rangle < C\langle b\rangle$, where "$\to$" is the symbol for "happening before."

Lamport also defines timestamps on messages to update logical time and manage access to a common resource. Because in a distributed system there is no universal clock, modules only become aware of changes in time when executing events. A message sender indicates its logical time with the timestamp; if later than the receivers' logical time, the receiver updates its logical clock; the sender does likewise with the acknowledgment.

To see how logical time might be understood in the context of material handling, consider a small network of unit-sized conveyors arranged in a grid (see Fig. 3). Two different "happening before" relations exist (see the right of Fig. 3). The transport steps of each box can only take place in a predetermined order. On the module where the routes of handling unit *a* and *b* cross, the transport steps must occur in a certain order. Handling unit *a* must have left the module before unit *b* enters or vice versa. The route of box *c* does not overlap with another route. Therefore, no causal relations exist, and the transport of box *c* can be performed independently of the other boxes.

For the two kinds of "happening before" relations, the following conditions must be fulfilled in order to satisfy Lamport's clock condition: The transport steps of one box must

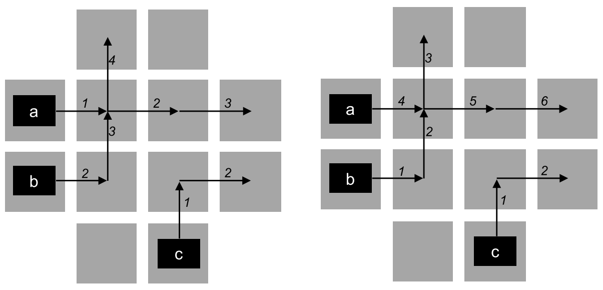

Fig. 4. Example for routes of three boxes with representative logical times.

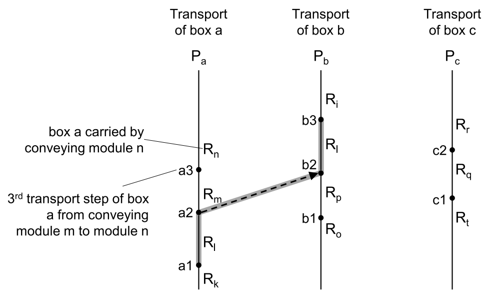

Fig. 5. Representation of the system in Fig. 4 as multiprocess system (inspired by [2]).

happen at advancing logical time. Consider one conveyor module. The outgoing transport of one box must take place at an earlier logical time than the incoming transport of the next box (in the example $C\langle b3\rangle < C\langle a1\rangle$ or $C\langle a2\rangle < C\langle b2\rangle$).

Two possible combinations of logical times of the events satisfying these conditions are shown in Fig. 4. Logical time increases by one unit in these examples.

V. Logical Time for Material Handling

There are two important differences between Lamport's conception of logical time and our model for material handling. First, in our model of material handling, it is resources that control the system by sending messages, not the processes. Second is the need to account for the physical (processing) time intrinsic to material handling events. In a pure expression of logical time, events have a sequence but no duration. In a world of events having nonzero time such as material handling, we must modify logical time appropriately.

A. Assigning Logical Clocks to Resources

Fig. 5 shows the example of Fig. 4 as a multiprocess system inspired by the representation of Lamport (compare to Fig. 2). The three processes P_a, P_b, and P_c (vertical lines) represent the transport of boxes with an uninterrupted sequence of resources to the destinations. With a transport step (dot), the box is transferred from one conveyor to another. For this transfer, both resources are needed. In the model, the physical duration of a transport step is irrelevant—the event simply includes the process switching from one resource to the next. A "happening before" relation between two processes (dotted arrow) is needed if a resource is used in common, i.e., if two routes cross. In Fig. 5, the common resource R_l of our example is highlighted in gray. The dotted arrow states that R_l must first be released by process P_a before being granted to P_b, which corresponds to the "happening before" relation $a2 \rightarrow b2$.

Even though Fig. 5 is similar to Fig. 2, it contains an additional dimension. Consider resource R_l highlighted in gray in Fig. 5. Following this resource in the horizontal direction, we observe that it forms a sequence of events like a process. Consequently, each event is related to one process and two resources that must be synchronized.

We now apply logical time: C is the set of all logical clocks in the system and $C\langle a1\rangle$ is the number assigned to event $a1$ by the set of all logical clocks. In [2], $C\langle a1\rangle = C_a\langle a1\rangle$ is assigned by the logical clock of process a if event $a1$ is part of process a. In our system, it is not the processes equipped with control, but the resources. Let C_i denote the logical clock of resource R_i. Consequently, the logical time of an event can only be assigned by the related resources that must have logical clocks of their own. For each event of a process, two resources are necessary. For example, in Fig. 5, the transport step $a1$ of box a is enabled by conveyors k and l. The logical clocks of both resources must assign the same logical time to the event. It must hold that $C\langle a1\rangle = C_k\langle a1\rangle = C_l\langle a1\rangle$ if $a1$ is an event for which resource R_k and R_l are needed.

Still, a valid sequence of resources for a process needs to be found and the resources need to agree on the logical time of an event. In order to find a route for a specific box to its destination, the conveyors send reservation requests to each other, assigning timestamps T to the transport steps of the box. By assigning a timestamp to each transport step, both conveyors agree on the logical time at which the transport should be performed. The entire set of reservations establishes a partial ordering of all transport steps for a box, as well as a partial ordering of all boxes on a resource. In order to satisfy the clock condition formulated in [2], the transport steps must be performed according to the partial ordering defined by the reservation timestamps. During transport of the boxes, each module sets its logical clock to the timestamp of the performed transport step and thereby synchronizes its logical clock with the neighboring conveyor module. The transport of boxes influences the reservation of modules because new reservations can only be accepted if they lie in future logical time in order to satisfy the clock condition.

B. Reserving Resources Using Logical Timestamps

For each process, all resources needed for its successful termination must be reserved before the process requests the first resource that could be commonly used. To use the principle of logical time for the control of a decentralized material handling system, the reservations must fulfill certain conditions that are introduced in this section. We do not present the algorithm showing how a set of possible reservations is found (i.e., the route reservation process) because it is not relevant for the principle of logical time. A possible implementation of the route reservation process is introduced in [14]. Here, we simply assume that a valid sequence of resources is known and reserved for each process. Each reservation is defined by a starting and an ending timestamp because the module is

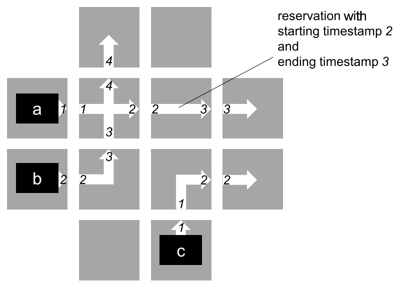

Fig. 6. Example for routes of three boxes with reservation (logical) times of resources.

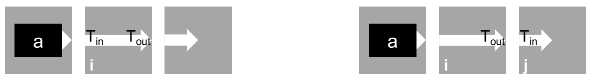

Fig. 7. Timestamps of (left) one reservation and (right) one transport step.

occupied by the box between the incoming and the outgoing transport. Two subsequent conveyor modules on the route of a box agree on the timestamp for this transport step. These reservations are distributively stored among the resources so that each resource only keeps its own reservations. A resource must be able to keep multiple reservations for different processes.

Fig. 6 shows the reservations for the previous example if the modules agree on the timestamps, as shown in Fig. 4 (left). The reservations are represented by white arrows. Each timestamp of a transport step is listed as the ending timestamp of the reservation of the sending module (head of arrow) and as the starting timestamp of the reservation of the receiving module (base of arrow). On the module where two routes are crossing, the reservations do not interfere with each other because the reservation for box *a* has the ending timestamp 2, whereas the reservation for box *b* has the starting timestamp 3.

Observe that the reservations are not necessarily received and accepted in the order of their timestamps. It is possible that a reservation is accepted even though another reservation exists for later timestamps, as long as the new request lies in the future logical time and does not interfere with other reservations. For example, the commonly used resource in Fig. 6 could have received the reservation of box *b* before the reservation for box *a*.

Let us now describe the reservation conditions formally. The reservation of a resource for a specific process is defined by two timestamps. Let $T_{\text{in}}(P_a, R_i)$ denote the timestamp of the incoming transport and $T_{\text{out}}(P_a, R_i)$ denote the timestamp of the outgoing transport. The reservation of resource R_i for process P_a is defined by the pair of timestamps

$$[T_{\text{in}}(P_a, R_i),\ T_{\text{out}}(P_a, R_i)]$$

with

$$T_{\text{in}}(P_a, R_i) < T_{\text{out}}(P_a, R_i) \tag{1}$$

representing that the incoming transport of a box must take place before its outgoing transport (see the left of Fig. 7). This condition guarantees that the clock condition for the causal relation between the events of one process is respected.

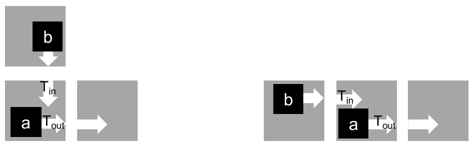

Fig. 8. Two boxes in (left) single movement and (right) tandem movement.

Two conveyors are needed to transport a box from one conveyor to the next one. If process P_a needs to use resource R_i and R_j subsequently, the condition

$$T_{\text{out}}(P_a, R_i) = T_{\text{in}}(P_a, R_j) \tag{2}$$

must be fulfilled (see the right of Fig. 7). This condition guarantees that both resources agree on the logical time of the related event.

As stated above, a resource can only be granted to one process at a time (with the exception of tandem movement). If process P_a uses resource R_i before process P_b, the reservations must fulfill the condition

$$T_{\text{out}}(P_a, R_i) < T_{\text{in}}(P_b, R_i) \tag{3}$$

if single movement is planned because box *a* must have left conveyor *i* before box *b* can enter (see the left of Fig. 8). This is being the case that the clock condition for the causal relation between the events of two different processes is respected. Single movement is necessary if the incoming direction of box *b* is perpendicular to the outgoing direction of box *a*.

Tandem movement is physically possible if the corresponding trio of resources is aligned, which is why box *a* leaves module *i* on the opposite side of box *b* entering (see the right of Fig. 8). In this case, resource R_i can be assigned to two processes, and instead of condition (3), the condition

$$T_{\text{out}}(P_a, R_i) = T_{\text{in}}(P_b, R_i) \tag{4}$$

must be fulfilled.

Let C_i be the logical clock of resource R_i defining its current logical time. Related to the logical clock, the reservation for process P_a must satisfy the condition

$$C_i \leq T_{\text{in}}(P_a, R_i) \tag{5}$$

in order to be in future logical time of the resource. As in any calendar system, reservations lying in the past cannot be accepted. An alternative, later time has to be found for the reservation.

C. Acquisition of Resources Using Logical Time

In Section III, we describe the course of acquisition of resources and state that a resource is only granted if the granting conditions are fulfilled. In this section, the conditions for granting and releasing a resource are defined using the timestamps of the related reservation. Each resource holds a list of all accepted reservations. Reservations are deleted once the corresponding transport steps are fulfilled.

Referring to box *b* in Fig. 1, the course of acquisition of resources with the additional information of the logical clocks works as follows: In Fig. 1(a) and (b), process P_b is holding resource R_i and its logical clock depicts the timestamp of the outgoing transport. If now resource R_j is granted to P_b, it sets its logical clock to the timestamp of the incoming transport

$$C_j := T_{\text{in}}(P_b, R_j). \tag{6}$$

In Fig. 1(c), P_b holds to resource R_i and R_j and switches to state *Transport*. The logical clocks of both resources show the same logical time. When the box is successfully transported to the next resource as shown in Fig. 1(e), the following actions take place.

1) The resource R_i is released.
2) R_i deletes the reservation of process P_b because all related transport steps have been performed.
3) Process P_b requests the next resource R_k and switches to state *Hold and Wait*.
4) R_j sets its logical clock to the timestamp of the outgoing transport

$$C_j := T_{\text{out}}(P_b, R_j). \tag{7}$$

The transition from the state *Transport* to the state *Hold and Wait* is only triggered by the end of the transport step, that is, it takes place after a finite physical time without any special conditions related to the reservation.

The following granting conditions must be fulfilled before starting a transport.

GC1: R_j is requested by P_b. This condition guarantees that the sending conveyor is ready for the transport.

GC2: R_j has performed all transport steps with lower timestamps and deleted the relevant reservations. Therefore, R_j can be granted to P_b if it does not keep a precedent reservation for any process P_a with $T_{\text{out}}(P_a, R_j) < T_{\text{in}}(P_b, R_j)$. (Within tandem movement, resource R_j keeps a reservation with $T_{\text{out}}(P_a, R_j) = T_{\text{in}}(P_b, R_j)$ that is allowed.)

GC3: The third condition depends on the transport type; for a single transport, R_j must be free (not held by any other process), for a tandem transport, R_j must be held by P_a which is in state *Transport*, and tandem transport must be physically possible (compare Fig. 1).

The granting conditions guarantee that transports are fulfilled in the order of their reservation timestamps so that the logical clock of the conveyor is always set forward and never back.

What does this set of logical clocks look like for an external observer? Each module has its own logical clock which is only set forward in discrete steps if a box is transported. If two neighboring modules perform a transport together, their logical clocks are synchronized, that is, their logical clocks are set forward to the timestamp reserved for the transport step. The set of logical clocks does not define one identical system time because all logical clocks could differ from each other. The logical time is completely independent of physical time. For example, it could happen that a conveyor module remains in the same logical time for a long period of physical time because it is not involved in the movement of any boxes.

Fig. 9 shows a conveyor network forming a crossing where boxes have only been transported from WEST to EAST several times. All other modules have remained in the logical time *0*.

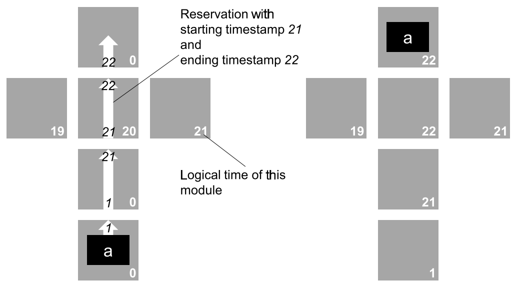
Reservation with starting timestamp 21 and ending timestamp 22
Logical time of this module
22
0
22
19
21
20
21
21
1
0
1
a
0
a
22
19
22
21
21
1

Fig. 9. Conveying network (left) before and (right) after the transport process of box *a*.

The left of the figure shows a valid reservation because all timestamps lie in the future logical time of the participating conveyor modules. When the transport of box *a* is finished, the conveyor modules have updated their logical time by synchronizing with the neighboring module when performing a transport step. The logical clocks have skipped all time steps in between. To put it more explicitly, one could say that the boxes bring the time to the conveyors.

VI. Proof of Liveness

To prove that a decentralized material handling system controlled with logical time is live, we must show that it is deadlock-free, livelock-free, and that no box will stay in the system indefinitely (no starvation). The latter two requirements are met by definition of route-based planning by reservation; boxes are disallowed from livelock behavior, and a valid route ensures that the box will depart eventually. To show that the system is deadlock-free, we first show that our modified algorithm satisfies the clock condition mentioned in [2] (a more detailed version can be found in [14]). We then prove that the only relevant deadlock condition (circular wait) is impossible.

Lemma 1: Our implementation of logical time for material handling satisfies Lamport's clock condition.

Proof: The "happening before" relation only exists between events of one process or between events on one resource. It is therefore sufficient to show that the events of one process take place in ascending order of their timestamps and that the logical clock of one resource is advancing, i.e., the related events take place in ascending order of their timestamps.

With the reservation conditions (1) and (2), we guarantee that the timestamps of reservations of one process are ascending. It is physically given that the transport steps are executed in this order. Therefore, the clock condition is satisfied.

Thus, if process P_b is holding to resource R_j and waiting for the next resource, no reservation exists for process P_b on any other resource $R_i \in R$ with

$$T_{\text{out}}(P_b, R_i) < C_j. \tag{8}$$

Consider now the events related to one resource. The logical clock is set to a new value if the resource is granted to a new process [(6)] or if the next resource is requested [(7)]. Granting condition **GC2** and reservation conditions (3) and (5) guarantee that the logical time is ascending when granted to a new process. Reservation condition (1) guarantees that logical time is ascending when the next resource is requested.

We conclude that the clock condition is also satisfied for the events happening on one resource. This means the following; if process P_b is holding to resource R_j, no reservation on resource R_j exists for any process $P_a \in P$ with

$$T_{\text{out}}(P_a, R_j) < C_j. \tag{9}$$

Tanenbaum and Bos [1] stated that a deadlock exists under the following conditions. First, all involved resources are held by a process. Second, resources cannot be preempted. Third, all involved processes are in state *Hold and Wait*. Fourth, the processes are waiting in a circular chain. The circular wait condition is the only one that can be negated in our system. To prove the absence of deadlocks, we show that the circular wait condition contradicts the described reservation and granting conditions.

Theorem 1: A system operated according to logical time for material handling is deadlock-free.

Proof: We know that a logical clock in our system is only set forward if an event takes place. In case of a deadlock with circular wait condition, no events take place in a set of resources held by processes. Therefore, the corresponding set of logical clocks would remain infinitely in their current logical time. We must show that the logical clocks of resources held by the process are set forward definitely.

Let us consider only the resources held by a process. If we can show that, out of this set of resources, the resources with the minimal value as logical time set their logical clock forward definitely; we can conclude that every event of every process will take place at some time because it will be the event related to the resource with the minimal value for the logical clock at some time.

Let us assume the worst case. All processes in the system are in state *Hold and Wait*. Let $C_{\min}$ denote the minimum logical time of all resources held by a process and $R_{\min} = \{R_i \mid C_i = C_{\min}\}$ the set of the held resources having the minimal logical time. Processes are only in *Hold and Wait* if granting conditions of Section V-C are not fulfilled:

1) **GC1** is fulfilled because the processes have already requested the next resource.
2) Because of (8) and (9), there is no resource $R_i \in R$ keeping a reservation for a process $P_a \in P$ with $T_{\text{out}}(P_a, R_i) < C_{\min}$. Thus, **GC2** is fulfilled for the set of resources $R_{\min}$.
3) Therefore, the processes holding these resources can only be in *Hold and Wait* because of **GC3**, meaning that the requested resource is not released by the previous process.

Suppose that the processes holding to the resources $R_{\min}$ are waiting in a chain, as shown in Fig. 11. In our system, the chain should include at least four different resources so that a deadlock in a loop is possible. Opposing deadlocks with only two resources are prevented by reservation condition (3).

It is not defined yet whether tandem transport is possible between these resources. From reservation conditions (2)–(4), we can deduce

$$T_{\text{out}}(P_i, R_i) = T_{\text{in}}(P_i, R_{i+1}) \geq T_{\text{out}}(P_{i+1}, R_{i+1})$$

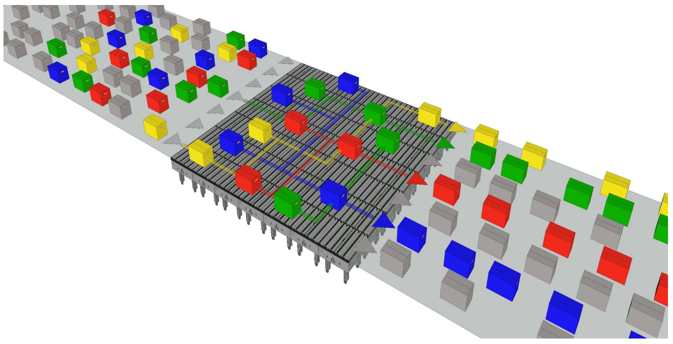
Fig. 10. GridSorter sorting packages from one side to the opposite side. Crossing routes lead to conflicts.

and consequently

$$T_{\text{out}}(P_1, R_1) \geq T_{\text{out}}(P_{1'}, R_{1'}). \tag{10}$$

For there to be a deadlock with circular waiting of these resources, it must be given $P_1 = P_{1'}$ and $R_1 = R_{1'}$. We can deduce

$$C_1 = T_{\text{out}}(P_1, R_1) = T_{\text{out}}(P_{1'}, R_{1'}) = C_{1'}. \tag{11}$$

Equation (10) is only consistent to (11) if tandem transport with reservation condition 4 is used exclusively in (10). For that, all trios of resources in this circular chain of *Hold and Wait* must satisfy the physical tandem transport condition by being aligned. This is only the case if all resources are aligned,[1] which makes it impossible that $R_1 = R_{1'}$. We can state that a circular waiting of resources connected with tandem transport reservations is physically impossible.

If $R_1 = R_{1'}$, there must be at least one single transport between resources in the chain and (10) becomes

$$T_{\text{out}}(P_1, R_1) > T_{\text{out}}(P_{1'}, R_{1'}) \tag{12}$$

which is in contradiction to (11). A circular waiting of processes is therefore impossible.

VII. Implementing Logical Time With GridSorter

GridSorter is a grid-based material handling system comprised of a grid of unit-sized FlexConveyors operating under decentralized control and the logical time scheme we describe earlier. Boxes enter the grid through any conveyor module on any side and exit through a specified destination module on any side. Thus, GridSorter could be used to sort boxes or totes in a distribution center or package handling facility. We have chosen the GridSorter for implementation because it is suited to the implementation of a control algorithm for the transport of goods to specific destinations in dense networks with high risk of deadlock.

Fig. 10 shows an exemplary GridSorter system. The topology of the network can take any shape and is recognized by a decentralized process based on the Link-State-Routing-Algorithm: After each module has requested the identifier of its neighbors, it distributes this information through the entire

[1]We assume that curved modules are modeled as conveyor modules with direction change and therefore without the ability of tandem movement.

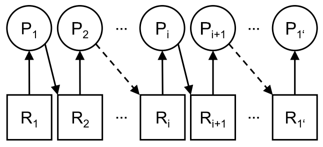

Fig. 11. Chain of processes in *Hold and Wait*.

network. With this information, modules can calculate shortest paths. Arriving boxes are identified by barcode or RFID before entering the grid. After receiving information about the destination of the box, the source conveyor begins the route reservation process via message passing to its immediate neighbors (see [14] for details). Once the route is determined, the transport starts according to the algorithm described in Section V-C.

The control algorithm described in this article has been implemented on a 5×5 demonstrator of GridSorter. For quantitative evaluation using multiagent simulation (see [14]), where several layout combinations have been analyzed regarding size, throughput, location of sources/destinations, and resulting synchronicity of logical clocks. We invite the reader to view a video of the GridSorter prototype at https://tinyurl.com/yxpr94m2. More detailed descriptions as well as a video showing GridSorter in operation can be found in the accompanying material of this publication.

VIII. Conclusion

Although originally conceived as a means of describing the partial ordering events among distributed computer operating systems, logical time is an effective means of routing physical entities in a network of resources such that entities reach their destinations without deadlock, livelock, or starvation.

A fair question with regard to our assumptions is, "Why assume there is no central clock, when any reasonable operating environment would make such a clock easily available?" We answer: eliminating the need for a central clock reduces the communication load on the system because the system can react flexibly to delayed or early transports without renegotiation of reservations. Unlike decentralized methods that rely on time windows with safety margins [15], execution with logical clocks introduces delays only in response to existing delays and then only as long as necessary. Liveness is guaranteed as long as the order of events is respected on each resource. In this sense, managing transport with logical time guarantees liveness in the same way as proposed in [12], but their algorithms assume centralized control, whereas we assume decentralized control with minimal requirements for information sharing and clock coordination.

We have shown by a formal proof that decentralized control with logical time is effective for a network of conveyors of any size and shape that can be modeled as unit-sized resources being able to transport boxes in one-to-four directions. Due to structural deadlock prevention resulting in highly local decision making, the algorithm is scalable in terms of network size. Therefore, we believe that logical time is a new way of thinking about and designing control algorithms for decentralized control.

Future research could extend the logical time to further applications, such as sequencing or buffering of boxes. The ability to address failures (lost messages and broke conveyors) should be included before industrial application because one conveyor remaining in its logical time forever would eventually result in a system with all conveyors in state *Hold and Wait*.

Finally, we observe that logical time is being used in practice: Gebhardt Fördertechnik GmbH has installed several instances of the FlexConveyor, which uses logical time as control principle.

References

[1] A. S. Tanenbaum and H. Bos, *Modern Operating System*, 4th ed. Boston, MA, USA: Pearson, 2015.

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[3] M. P. Fanti, A. M. Mangini, G. Pedroncelli, and W. Ukovich, "A decentralized control strategy for the coordination of AGV systems," *Control Eng. Pract.*, vol. 70, pp. 86–97, Jan. 2018.

[4] K. Windt, F. Böse, and T. Philipp, "Autonomy in production logistics: Identification, characterisation and application," *Robot. Comput.-Integr. Manuf.*, vol. 24, no. 4, pp. 572–578, Aug. 2008.

[5] S. Mayer and K. Furmans, "Deadlock prevention in a completely decentralized controlled materials flow systems," *Logistics Res.*, vol. 2, nos. 3–4, pp. 147–158, Dec. 2010.

[6] K. R. Gue, K. Furmans, Z. Seibold, and O. Uludag, "GridStore: A puzzle-based storage system with decentralized control," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 429–438, Apr. 2014.

[7] O. Uludag, "GridPick: A high density puzzle based order picking system with decentralized control," Ph.D. dissertation, Dept. Ind. Syst. Eng., Auburn Univ., Auburn, AL, USA, 2014.

[8] K. R. Gue, O. Uludag, and K. Furmans, "A high-density system for carton sequencing," in *Proc. 6th Int. Sci. Symp. Logistics*, 2012, pp. 1–5.

[9] T. Krühn, M. Radosavac, N. Shchekutin, and L. Overmeyer, "Decentralized and dynamic routing for a cognitive conveyor," in *Proc. Int. Conf. Adv. Intell. Mechatronics (AIM)*, 2013, pp. 436–441.

[10] T. Krühn, "Dezentrale, verteilte Steuerung flchiger Fördersysteme für den innerbetrieblichen Materialfluss," Ph.D. dissertation, Fakultät für Maschinenbau, Gottfried Wilhelm Leibniz Universität Hannover, Hannover, Germany, 2014.

[11] C. W. Kim and J. M. A. Tanchoco, "Conflict-free shortest-time bidirectional AGV routeing," *Int. J. Prod. Res.*, vol. 29, no. 12, pp. 2377–2391, Dec. 1991.

[12] S. Maza and P. Castagna, "A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles," *Comput. Ind.*, vol. 56, no. 7, pp. 719–733, Sep. 2005.

[13] P. Zítek, *Sequence Based Hierarchical Conflict-Free Routing Strategy Bi-Directional Automated Guided Vehicle*. Amsterdam, The Netherlands: Elsevier, 2005.

[14] Z. Seibold, "Logical time for decentralized control of material handling systems," Ph.D. dissertation, Karlsruhe Inst. Technol., Karlsruhe, Germany, 2016.

[15] S. Sohrt and L. Overmeyer, "Decentralized routing algorithm with physical time windows for modular conveyors," *Logistics Res.*, vol. 13, no. 1, Aug. 2020, doi: 10.23773/2020_8.