Options for Maintaining Weak FIFO in Parallel Queues

Kaan Kalkanci[1] and Christoph Roser[1]

[1] Karlsruhe University of Applied Sciences, Moltkestrasse 30, Karlsruhe, Germany,
Corresponding author: christoph.roser@hochschule-karlsruhe.de

Abstract: Maintaining a FIFO sequence is a common aspect of manufacturing, especially in lean manufacturing. This can easily be realized by having FIFO lanes, where items are added at one end and removed at the other end. This maintains their FIFO sequence. In industry, however, there is often the situation that the required quantity of items would result in a very long single FIFO lane. This creates subsequent problems with available floor space. Hence, a practical solution is often to have multiple parallel FIFO lanes. However, a separate system is now needed to maintain the FIFO sequence across multiple parallel FIFO lanes. Ideally this would not require much additional managerial overhead and high investment costs. This paper investigates different options on how to manage parallel FIFO lanes, and measures their sequencing performance.

Keywords: FIFO, pull production, storage, lean manufacturing

1 Introduction

The sequence of material flow in manufacturing and other processing systems is relevant for overall performance. About the most common method used is FIFO (First In, First Out). The oldest material is always used first. The primary advantage of FIFO is that it reduces the fluctuations of the lead time for individual items. All items will have a similar lead time in a FIFO lane. In contrast, a LIFO (Last In, First out) will create vastly different lead times for different items. The first item in will have the longest lead time, and the last item added will have the shortest lead time. LIFO is generally not advisable, but may sometimes be used due to the nature of storage. Both principles are shown in the following Figure 1.

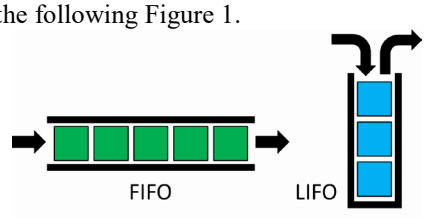

Figure 1: Simplified illustration of FIFO and LIFO

FIFO is most of all a buffer inventory to decouple fluctuations between the arrival and departure processes. By limiting the maximum inventory, it also prevents excess

build up of material. FIFO also has other advantages, as it allows better traceability. If there is a systematic product quality issue, or if there is a design change, it is much easier to track the items that have to be changed. It is also much easier to use up the items of the "old" design before switching over to the "new" design [1]. Furthermore, FIFO represents a transparent material storage, because the information flow is ensured through the materials in the FIFO lanes.

2 Problem Statement

This paper is based on a master thesis [2]. In this paper, we will discuss possibilities for managing parallel FIFO lanes. In theory, it is often assumed to have a single FIFO lane in a certain length that is needed. There are different academic methods to determine the FIFO size (for example [3], [4], [5], [6], [7]). These can be used to design capacities of FIFO lanes and production areas. In practice, however, limitations on the floor space and storage system often make it difficult or impossible to place all required items into one FIFO lane. One option is to have random-access storage and track the FIFO sequence using an overarching ERP system. Another option commonly seen in industry is to use multiple parallel FIFO lanes, what is being investigated in this article.

An example of multiple parallel FIFO lanes is shown in Figure 2. The items arrive in sequence. An adding logic decides in which of the multiple parallel FIFO lanes an item is stored. Naturally, this has to be a FIFO lane that is not yet completely full. In addition, there is also customer demand, where the customer requests items periodically. A removing logic decides from which of the multiple parallel FIFO lanes an item will be removed. Similar to the adding logic, this has to be a FIFO lane that is not yet completely empty. If all FIFO lanes would be full, no more items could be added. The system is blocked. Otherwise, if all FIFO lanes are empty, no more items could be removed. The system is starved.

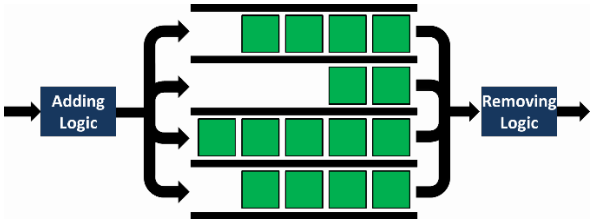

Figure 2: Example of four parallel FIFO lanes each with a capacity of five items

Hence through the adding and removing logic, it is possible to maintain FIFO [8]. Here we have to distinguish between two definitions. First, a strict FIFO where the items leave the system in exactly the sequence they arrived. Second, a weak FIFO sequence where the items leave the system approximately in the sequence they arrived [8]. While within this paper we analyze weak FIFO sequences in particular, we would first briefly introduce the approach for a strict FIFO sequence.

3 Strict FIFO Sequence Approach

There are three basic approaches in multiple parallel FIFO lanes that are able to maintain a strict FIFO sequence. The first is to create a digital or physical timestamp of the time when the item was added into the parallel FIFO system. Alternatively, a number could be used to indicate the sequence. If an item is needed, the item with the oldest timestamp is used. The disadvantage of this method is the effort in creating a timestamp and the effort in searching for the oldest item during removal. Nevertheless, this approach is used, in particular in combination with a digital MES (manufacturing execution system) or ERP (enterprise resource planning) system.

A second approach is to fill the parallel FIFO lanes sequentially (i.e. fill the first FIFO lane, then the second FIFO lane, etc.).This is followed by the removal of parts by emptying the first FIFO lane, then the second FIFO lane, etc. The major caveat of this approach is that the filling process must not overtake the emptying process. This is visualized in Figure 3, where for your understanding the parts are numbered in the sequence of arrival. On the left image the filling process is about to fill a lane before the lane assigned for emptying. In the right image this lane is now full, and the filling process switches to the next lane. This, however, breaks the sequence.

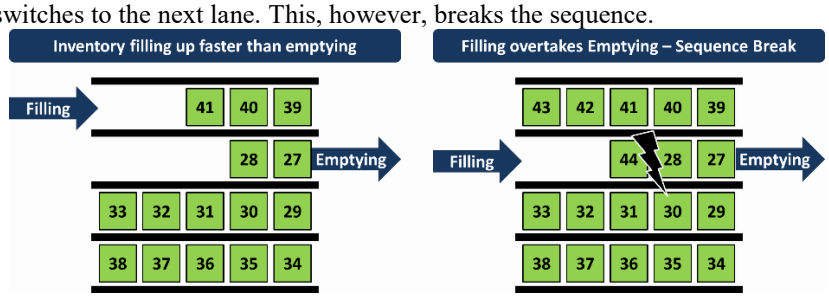

Figure 3: Problem of filling overtaking the emptying process

This breaking of the sequence not only breaks the strict FIFO sequence, but due to the nature of the break we now no longer even have a weak FIFO sequence. Depending on the subsequent arrivals and departures, the items in the other FIFO lanes may remain in the buffer for a very long time while filling and emptying use only one FIFO lane.

This problem can be avoided by blocking the filling of the FIFO lane that is currently emptied. In this case, however, the usable buffer space is less than the total buffer space, and the buffer capacity fluctuates. On the shop floor it would also require very stringent enforcement of standards to prevent, for example, a forklift driver from dropping a part in the only available space just because something indicates he should not place it there.

Another option is to have a signal (red sign, dummy part, etc.) in between the two non-sequential parts. This would be, for example, parts 28 and 44 in Figure 3. The challenge here is to know when to add this signal, which may not be obvious to the person adding the items. This does have potential for mistakes.

A third option is to always fill the next part into the next FIFO lane adjacent to the previous adding. For removal, the part from the next FIFO lane is removed adjacent to

the previous removal. This is also modeled below in the *Add Cyclic* & *Take Cyclic* combination.

Overall, it is possible to maintain a strict FIFO in parallel systems without the use of timestamps. However, this may require some effort. For example, it can be imagined that a forklift driver would have to get off his forklift to move a signal – and we all know that forklift drivers usually prefer not to get off their vehicle. Furthermore, this may not use the entire available space. Additionally, if a mistake is made, the items can potentially be very badly out of sequence. Especially the second approach above is hence not very robust, and small mistakes may have significant consequences.

In many situations, however, a weak FIFO sequence is sufficient (e.g., if the surrounding system does not include traceability, or in general the advantages of a strict FIFO are not utilized). In this case it may be better to maintain only a weak FIFO with less effort. In the following we compare possibilities for a weak FIFO sequence that are

- more robust against mistakes;
- do not require complex signals within the FIFO; and
- use all of the available space in the FIFO.

4 Methods for Adding and Removing

We will consider five different strategies for adding and removing items from multiple parallel FIFO lanes, which are presented below.

4.1 Adding Methods

One method for adding items is **Add Random**. In this case, the items arriving are randomly assigned to one of the empty FIFO lanes. The assignment is equally distributed, with only non-full FIFO lanes being used as addition possibilities.

Another method is **Add Max**, where the item is added to the FIFO lane with the largest non-full inventory. If more than one FIFO lane has the largest non-full inventory, the method prefers the FIFO lane that was used for the last addition. If this no longer has the largest non-full inventory, sequentially the next FIFO lane from these equally largest non-full inventories will selected for the addition process.

A further method represents **Add Min**, where the item is added to the FIFO lane with the smallest inventory. If more than one FIFO lane has the smallest inventory, the method prefers the FIFO lane that was used for the last addition. If this no longer has the smallest inventory, sequentially the next FIFO lane from these equally smallest inventories is selected.

The fourth method is **Add Repeat**, where items are repeatedly added in the same FIFO lane until it is full. After this FIFO lane is full, the next non-full FIFO lane is selected. Items are added into this new FIFO lane until it is also full. Then the process is repeated. Please note that if this method includes a signal in case of an overlap, then this *Add Repeat* and corresponding *Take Repeat* would maintain a strict FIFO as presented in section 3.

Finally, **Add Cyclic** is used as the fifth method, where each item is added to the next non-full FIFO lane from the previous one. Doing this, the first item is added to FIFO lane 1, the second to FIFO lane 2, and the third to FIFO lane 3, etc. This will continue until an item has been added to the last FIFO lane, after which the cycle begins again with FIFO lane 1. If during this procedure a FIFO lane is full, it is skipped.

4.2 Removing (Taking) Methods

Similar methods are used for removing items. One of these is the **Take Random** method, which takes an item randomly from one of the non-empty FIFO lanes. The random removal is with equal probability from the non-empty FIFO lanes.

The **Take Max** method removes the item from the FIFO lane with the largest inventory. If more than one FIFO lane has the largest inventory, the method prefers the FIFO lane that was used for the last removal. If this no longer has the largest inventory, sequentially the next FIFO lane from these equally largest inventories will be selected for the addition process.

The **Take Min** method removes an item from the FIFO lane, which contains the smallest non-zero inventory. If more than one FIFO lane has the smallest non-zero inventory, the method prefers the FIFO lane that was used for the last removal. If this no longer has the smallest non-zero inventory, sequentially the next FIFO lane from these equally smallest inventories.

The **Take Repeat** method removes items repeatedly from the same FIFO lane until it is empty. After this FIFO lane is empty, the next non-empty FIFO lane is selected, and items are removed from this FIFO lane until it is also empty. Then the process is repeated. The sequence of the FIFO lanes for the removal is identical for the sequence of the FIFO lanes for adding if the adding method has a sequence (*Add Repeat*, *Add Cyclic*).

The fifth method is **Take Cyclic**, where each item is removed from the next non-empty FIFO lane from the previous one. In this case, the first item is removed from FIFO lane 1, the second from FIFO lane 2, and the third from FIFO lane 3, etc. By analogy to *Add Cyclic*, this will continue until an item has been removed from the last FIFO lane. After this the cycle begins again with FIFO lane 1. If during this procedure a FIFO lane is empty, it is skipped. The sequence of the FIFO lanes for the removal is identical for the sequence of the FIFO lanes for adding if the adding method has a sequence (*Add Repeat*, *Add Cyclic*). Please note that the combination of *Add Cyclic* and *Take Cyclic* would give a strict FIFO sequence.

There would also be another method, **Take Oldest**, where the oldest item in the parallel FIFO lanes is removed. This would ensure a strict FIFO, where the removal of the items is exactly in the sequence they are added. This is regardless of the method in which they were added to the FIFO lanes. However, this would require a data system or a comparison of the first item in all parallel FIFO lanes to find the oldest one. While best for the sequence, it is not always practical due to the effort in determining the oldest item and the amount of investment costs incurred.

4.3 Combination Adding and Removing (Taking) Methods

Within the scope of selecting FIFO lanes, we have five methods for adding items and five for removing items (excluding *Take Oldest*). Hence, we have a total of 25 combinations of adding and removing methods as shown in Figure 4. Please note again that combination 25 gives a strict FIFO sequence.

	Add Random	Add Max	Add Min	Add Repeat	Add Cyclic
Take Random	1	2	3	4	5
Take Max	6	7	8	9	10
Take Min	11	12	13	14	15
Take Repeat	16	17	18	19	20
Take Cyclic	21	22	23	24	25

Figure 4: 25 combinations of adding and removing strategies, with #25 having a strict FIFO sequence

5 Simulation Approach

We simulated all 25, adding and removing combinations for a system with 10 parallel FIFO lanes with a capacity of 10 each. The inter-arrival and inter-departure times were exponentially distributed. The system was tested under three different load conditions. A low load had an on average 0.5% less items arriving than demanded (i.e., the customer frequently had to wait and the inventory was frequently empty). A high load had on average 0.5% more items arriving than needed, resulting in a mostly full inventory. Finally, a medium load had on average as many items arriving as consumed. We measured the sequencing quality as the root mean square error (RMSE) of the difference between the position in the arrival sequence and the position in the departure sequence. For each simulation this was calculated as shown in equation (1), where i is the i^{th} item in the arrival sequence out of n items in total, and T_i is the position of the item i in the departure sequence.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(i - T_i)^2} \quad (1)$$

Each simulation processed 10,000 items. Furthermore, each simulation was repeated 52 times for statistical accuracy. This resulted in almost 4,000 simulations to determine the mean RMSE and its confidence interval.

6 Simulation Results

Below are exemplary of the results of all 25 combinations for inter-arrival and inter-departure times with same parameters, including its 2.5% confidence interval. The results in the following Figure 5 are sorted by the removing methods. Full results can be found in [2].

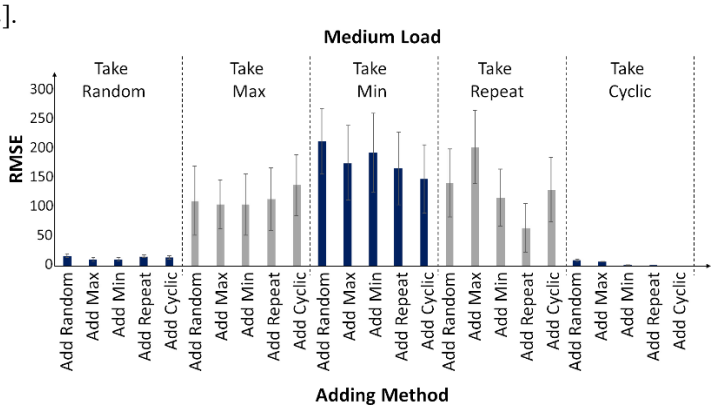

Figure 5: RMSE and confidence intervals of all 25 combinations for medium load

The differences between the removing methods indicates that the removal method has a significant effect on the sequence quality. Removing items using the *Take Min* approach results in the worst sequencing quality, having an average RMSE across all five adding strategies of 180. The *Take Repeat* and *Take Max* approaches are only marginally better with an average RMSE of 131 and 115. Removing items randomly using *Take Random* is actually a pretty good strategy with an average RMSE of 13. However, the best sequence quality was achieved by using *Take Cyclic*, with an average RMSE of only 5.6 across all five adding strategies. Comparable results were achieved with the low- and high-load conditions. A strict FIFO can be maintained by the combination of *Add Cyclic–Take Cyclic* strategy with a RMSE of 0. This combination brought overall the highest quality FIFO sequence. If a weak FIFO is sufficient, a *Take Cyclic* approach gives the smallest RSME regardless of the adding sequence, saving the additional effort of *Add Cyclic*.

7 Summary

Overall, a strict FIFO maintaining the correct sequence is of course the best approach. However, for parallel FIFO lanes that are not managed digitally, the effort of maintaining strict FIFO may exceed the benefit. In these cases, a weak FIFO can be achieved simply by using the *Take Cyclic* approach, where every item is taken from the next FIFO lane. This approach maintains an adequate FIFO sequence with acceptable additional effort, regardless of the logic used for adding items into the FIFO. It is also a robust approach. While this does not maintain a strict FIFO sequence, it is often the

appropriate solution for practical applications. Overall this gives a good trade-off between effort and FIFO quality if a strict FIFO sequence with parallel FIFO lanes would lead to high effort or additional investment costs.

8 References:

[1] W. A. Günthner, J. Durchholz, E. Klenk, J. Boppert, T. Knössl, and M. Klevers, *Schlanke Logistikprozesse: Handbuch für den Planer*, 2013th ed. Berlin: Springer Vieweg, 2013.

[2] K. Kalkanci, "Entwicklung und Simulation von Ein- und Auslagerungsstrategien zur Sicherstellung des FIFO-Prinzips bei dezentralen parallelen Materiallagerungen," Master Thesis, Karlsruhe University of Applied Sciences, Karlsruhe, Germany, 2019.

[3] C. M. Lutz, "Determination of buffer location and size in production lines using tabu search," *European Journal of Operational Research*, vol. 106, pp. 301–316, 1998.

[4] F. Altiparmak, B. Dengiz, and A. A. Bulgak, "Optimization of Buffer Sizes in Assembly Systems Using Intelligent Techniques," in *Winter Simulation Conference*, San Diego, CA., USA, 2002, pp. 1157–1162.

[5] A. A. Bulgak and J. L. Sanders, "Integrating a modified simulated annealing algorithm with the simulation of a manufacturing system to optimize buffer sizes in automatic assembly systems," in *Winter Simulation Conference*, Piscataway, NJ, USA, 1988, pp. 684–690.

[6] C. Roser, M. Nakano, and M. Tanaka, "Buffer Allocation Model Based on a Single Simulation," in *Winter Simulation Conference*, New Orleans, Louisiana, USA, 2003, pp. 1238–1246.

[7] C. Roser, M. Nakano, and M. Tanaka, "Single Simulation Buffer Optimization," *JSME International Journal Series C: Mechanical Systems, Machine Elements and Manufacturing*, vol. 48, no. 4, Dec. 2005, pp. 763–769.

[8] T. Atz and W. A. Günthner, "Integrierte Lagersystemplanung," *Logistics Journal: Proceedings*, vol. 07, no. 1, Aug. 2011.