Computing Collision Probability in a Series-Parallel Machines Model

Eishi CHIBA*,† Taiki OTSUKA*

*Department of Industrial and Systems Engineering, Hosei University, 3-7-2 Kajino-cho, Koganei-shi, Tokyo, 184-8584 Japan

Abstract

In this paper, we consider collision probability in a manufacturing line model. Collision probability is the probability of jobs colliding in a manufacturing line and is a comparatively new evaluation item in systems in an unsteady state, having real-world applications in, for example, flat panel display and semiconductor manufacturing systems. Previous work in this area includes efficient methods to compute collision probability in an in-line machines model, a parallel machines model, and a generalized in-line machines model with consideration of delivery time and buffers. Two optimization problems with collision probability also exist for the generalized in-line machines model which address the minimization of tact time and total number of buffers. In an in-line machines model, when the processing time on a particular machine is longer than on others, this machine becomes a bottleneck in the production process and may lengthen the total production completion time. However, if two such machines are placed in parallel, then a reduction in the total production completion time is expected. In this paper, we generalize the in-line machines model to a series-parallel machines model and present a method which efficiently computes collision probability using computer simulation. The key idea regarding efficiency is the use of a heap data structure when searching for an idle machine among parallel machines. We also implement the method presented and show the findings obtained from computational experimentation.

Key words: production scheduling, operations research, series-parallel machines model, collision probability

1 Introduction

In the fields of scheduling research and queueing theory, production scheduling is a widely researched area. For example, Refs. [1, 13] give algorithms for parallel machine scheduling problems and flowshop scheduling problems, and Refs. [8, 9, 10] give analysis results for production models from the viewpoint of queueing theory.

This paper focuses on collision probability in a series-parallel machines model. Collision probability is the probability of jobs colliding in a manufacturing line and is a comparatively new evaluation item in systems in an unsteady state, having real-world applications in, for example, flat panel display and semiconductor manufacturing systems.

All previous results relating to collision probability can be found in Refs. [2, 3, 4, 5, 6, 12]. The concept of collision probability was first introduced in article format in Ref. [3], and the analytical results pertaining to the approximate formula of collision probability were given in Refs. [2, 3, 4]. In Ref. [3], when the processing time followed a general distribution, an approximate expression of the collision probability was given, which was expressed by a multiple integration. In Ref. [4], when the processing time followed an Erlang distribution, a closed-form formula of the approximate collision probability was derived. Ref. [5] focused on a machines model with buffers, in which jobs can temporarily wait, and presented a computer simulation method that computed collision probability. It also offered a minimization problem of the total number of buffers, and a heuristics to solve it. In Ref. [6], it was found that consideration of delivery time between machines did not affect collision probability, and was referred to as the redundancy of delivery time. Note that Refs. [2, 3, 4, 5, 6] considered collision probability in an in-line machines model, whereas Ref. [12] ad-
dressed collision probability in a parallel machines model. Concretely, Ref. [12] presented a framework for computing collision probability using a flag search, which is a method that searches for a zero in a 0/1 sequence of any length.

In an in-line machines model, when the processing time on a particular machine is longer than on others, this machine becomes a bottleneck in the production process and may lengthen the total production completion time. However, if two such machines are placed in parallel, then a reduction in the total production completion time is expected. Therefore, in this paper, by generalizing the in-line machines model presented in Refs. [2, 3, 4], we offer a new series-parallel machines model with consideration of collision probability, and then present an efficient method to compute collision probability in this model. Note that this series-parallel machines model is also a generalization of the parallel machines model proposed in Ref. [12].

The remainder of this paper is organized as follows. In Section 2, we describe a formal series-parallel machines model with consideration of collision probability. In Section 3, we present a method to decide whether a collision occurs or not in the proposed model. Based on this method, we describe a computer simulation method to evaluate exact collision probability. In Section 4, we implement the method presented in Section 3 and show the findings obtained from computational experimentation. Finally, Section 5 concludes this paper.

2 Series-Parallel Machines Model

In this section, we describe a series-parallel machines model. This is a model which generalizes each machine in the in-line machines model presented in Ref. [3] to parallel identical machines. The following notations are used:

- $s$: the number of processing stages.
- $m_l$: the number of machines in stage $l$.
- $M^{(l)}_1, M^{(l)}_2, \ldots, M^{(l)}_{m_l}$: $m_l$ identical parallel machines in stage $l$.
- $J_1, J_2, \ldots, J_n$: $n$ jobs to be processed.
- $t_{tact}$: tact time, that is, the time difference between the points in time when $J_i$ and $J_{i+1}$ are fed into the line at the entrance for all $1 \leq i \leq n - 1$.
- $t_{ti}^{(l)}$: processing time of job $J_i$ at stage $l$.

The series-parallel machines model is illustrated in Fig. 1. There are $s$ processing stages on identical parallel machines, i.e. each job can be processed on any machine in each stage. The number of machines in stage $l = 1, 2, \ldots, s$ is denoted by $m_l$, and $m_l$ machines in stage $l$ are denoted by $M^{(l)}_1, M^{(l)}_2, \ldots, M^{(l)}_{m_l}$, where these $m_l$ machines are identical.

Given $n$ jobs $J_1, J_2, \ldots, J_n$ to be processed, these jobs are fed one by one into the line at the entrance with the same time interval. This time interval is referred to as tact time, which is denoted by $t_{tact}$. The feeding order of jobs into the line is $J_1, J_2, \ldots, J_n$.  

![Figure 1: A series-parallel machines model.](image-url)
Each job is first processed on an idle machine in stage 1. It is then automatically delivered to an idle machine in stage 2 after it has been finished in stage 1. As soon as the machine in stage 2 receives the job, it starts processing. In this manner, each job is processed on an idle machine in each stage, and then sent to the exit. It is assumed, for simplicity, that the delivery time of each job is nil. The processing time of job $J_i$ on any machine in stage $l$ is denoted by $T^{(l)}_i$. All processing times $T^{(l)}_i$ $(1 \leq i \leq n, 1 \leq l \leq s)$ are assumed to be random variables and independent of each other.

When a job arrives at stage $l$, the system dynamics are as follows. If an idle machine in stage $l$ exists, it starts processing the job. Otherwise, a collision at stage $l$ occurs. The collision probability is the probability that there will be at least one collision at any stage. If the number of machines in stage $l$ is no less than the number of jobs, then a collision never occurs at stage $l$ and so, from now on, we assume that $m_l < n$ for $1 \leq l \leq s$.

### 3 Computing Collision Probability

In this section, we first present a method to decide whether a collision occurs or not in the series-parallel machines model when the processing times of jobs are given. Next, based on this method, we present a simulation method to compute the collision probability in the model.

#### 3.1 Collision Decision Method

The random variables $T^{(l)}_i$ $(1 \leq i \leq n, 1 \leq l \leq s)$ are generated randomly from probability distribution. The variable $t^{(l)}_i$ is assumed to be the generated value derived from $T^{(l)}_i$ for $1 \leq i \leq n, 1 \leq l \leq s$. Given the number of stages $s$, the number of machines $m_l$ in each stage $l$, a tact time $t_{\text{act}}$, and $n$ jobs with processing times $t^{(l)}_i$ $(1 \leq i \leq n, 1 \leq l \leq s)$, we consider the computation to decide whether a collision occurs or not.

Throughout the computation, we manage the points in time when machines become available for processing. To manage such points in time, we apply a heap data structure, which satisfies the property that the value of every node other than the root is at least the value of its parent. From this property, assuming that $H$ is an array that represents a heap, $H[1]$ is the smallest value in the heap. Therefore, when a job arrives at a stage, an idle machine exists from the point $H[1]$ in time. Moreover, an attribute $H.\text{heap-size}$ is assumed to represent how many values in the heap are stored within array $H$. A heap supports the following operations.

- $\text{PushHeap}(H, x)$ inserts the value $x$ into the array $H$.
- $\text{DeleteMin}(H)$ removes the smallest value from the array $H$.

The variable $st_i$ is introduced in order to store the point in time when a job $J_i$ arrives at a stage. When a job $J_i$ arrives at a stage, we need to search for an idle machine in the stage. If $st_i$ is earlier than the point $H[1]$ in time, then an idle machine does not exist in the stage, meaning that a collision occurs. Otherwise, an idle machine exists, and then $J_i$ is processed on the machine. As a result, the point in time when the machine becomes available for processing is updated. Concretely, the smallest value $H[1]$ is removed from the heap, and then the point in time obtained by adding $J_i$’s processing time at the stage to $st_i$ is inserted into the heap.

The processing order of $n$ jobs at a stage can be represented by a permutation, i.e. a bijection

$$\pi: \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}.$$  

Note that the processing order of jobs at a stage is the same as the order in which jobs arrive at the stage. Although $\pi(i) = i$ for $1 \leq i \leq n$ at the first stage, the processing order of $n$ jobs may change at a subsequent stage. We decide whether a collision occurs or not at each stage in the order from the first stage to the last.

From the above discussion, the method to decide whether a collision occurs or not is described more precisely in Procedure COLLISION_DECISION. Here, lines 1–3 compute the points in time when jobs arrive at the first stage. The $l$-th iteration of the for loop of lines 4–19 computes whether a collision occurs or not at stage $l$. Line 5 computes the processing order of jobs at stage $l$. The points in time when $m_l$ machines in stage $l$ become available for processing are initialized to zero in lines 6–9. The $i$-th iteration of the for loop of lines 10–18 decides whether a collision at stage $l$ involving the job $J_{\pi(i)}$ occurs or not. Line 14 computes the point in time when $J_{\pi(i)}$ arrives at the next stage.

Assuming $m^* = \max_{1 \leq i \leq n} m_i$, we can analyze the time complexity of Procedure COLLISION_DECISION as follows. The sorting of $n$ jobs (in non-decreasing order of their arrival times at stage $l$) in line 5 takes $O(n \log n)$ time. The initialization of heap in lines 6–9 takes $O(m^*)$ time, since the call of PushHeap in line 8 takes $O(1)$ time. Deciding whether a collision occurs or not at stage $l$ in lines 10–18 takes $O(n \log m^*)$ time, since each call of DeleteMin and PushHeap in lines 12–13 takes $O(\log m^*)$ time. Therefore, the computation in lines 5–18 takes $O(n \log n + m^* + n \log m^*)$ time, which is simplified to $O(n \log n)$ from the assumption $m_l < n$ for $1 \leq l \leq s$.  

Vol. 11 No. 1 (2019)
As a result, Procedure COLLISION_DECISION runs in time $O(ns \log n)$.

**Procedure 1 COLLISION_DECISION**

**Input:** The number of stages $s$, the number of machines $m_l$ in each stage $l$, a tact time $t_{tact}$, and $n$ jobs with processing times $t_i^l$ ($1 \leq i \leq n, 1 \leq l \leq s$).

**Output:** 1 if a collision occurs, and 0 otherwise.

1: for $i = 1$ to $n$ do
2:  $st_i \leftarrow (i-1)t_{tact}$
3: end for
4: for $l = 1$ to $s$ do
5:  Compute a permutation $\pi$ such that $st_{\pi(i)} \leq st_{\pi(i+1)}$ for $i = 1, 2, \ldots, n - 1$.
6:  $H$.heap-size $\leftarrow 0$ {Make heap empty}
7: for $j = 1$ to $m_l$ do
8:  PushHeap($H, 0$)
9: end for
10: for $i = 1$ to $n$ do
11:  if $H[1] \leq st_{\pi(i)}$ then
12:    DeleteMin($H$)
13:    PushHeap($H, st_{\pi(i)} + t_i^l$)
14:  else
15:    $st_{\pi(i)} \leftarrow st_{\pi(i)} + t_i^l$
16:  return 1 {Collision occurs}
17: end if
18: end for
19: return 0 {Collision does not occur}

**Procedure 2 SIMULATE**

**Input:** The number of jobs $n$, the number of stages $s$, the number of machines $m_l$ in each stage $l$, a tact time $t_{tact}$, the parameters of a probability distribution for each stage, and a positive integer $c$.

**Output:** A collision probability.

1:  cnt $\leftarrow 0$
2:  for loop = 1 to $c$ do
3:    Generate processing times $t_i^l$ ($1 \leq i \leq n, 1 \leq l \leq s$) randomly from probability distribution.
4:    flag $\leftarrow$ COLLISION_DECISION
5:    if flag = 1 then
6:      cnt $\leftarrow$ cnt + 1
7:    end if
8:  end for
9: return cnt/c

Throughout all simulations, we use Mersenne Twister [11] as the pseudorandom generator, and the number of jobs and the number of iterations are set to $n = 1,000$ and $c = 10,000$, respectively. Moreover, since Erlang distribution is flexible enough to represent actual processing times [4], the processing times $T_i^l$ ($1 \leq i \leq n$) at stage $l$ are assumed to follow the Erlang distribution with parameters $\lambda_l$ and $k_l$. The probability density function of the Erlang distribution is defined as follows:

$$f(x; k_l, \lambda_l) = \frac{\lambda_l^{k_l}x^{k_l-1}e^{-\lambda_l x}}{(k_l-1)!}$$

for $x \geq 0$,

where two parameters $\lambda_l$ and $k_l$ are a positive real number and a positive integer, respectively. The expectation and variance of the random variable that follows the Erlang distribution are given by $k_l/\lambda_l$ and $k_l/\lambda_l^2$, respectively.

Firstly, assuming that $s = 1$, $k_1 = 1,000$, and $\lambda_1 = 1,000$, we compute the collision probabilities for each case of $m_1 = 1, 2, \ldots, 5$. The results are shown in Fig. 2, where the horizontal axis denotes the tact time $t_{tact}$. It shows that for each case of $m_1 = 1, 2, \ldots, 5$, as the tact time passes a certain threshold, the collision probability decreases rapidly, clearly exhibiting the trade-off between the tact time and the collision probability. Moreover, from Fig. 2 we can confirm that the collision probability decreases with an increasing value of $m_1$ when the value of $t_{tact}$ is fixed.

Note, in particular, that when $s = 1$ and $m_1 = 1$, the theoretical formula for collision probability can be found in Ref. [4], which is as follows:

$$1 - \left(1 - \sum_{h=0}^{k_1-1} \frac{(\lambda_1 t_{tact})^h}{h!} e^{-\lambda_1 t_{tact}}\right)^{n-1}.$$ (1)
Figure 2: The relationship between the tact time and the collision probability when $m_1 = 1, 2, \ldots, 5$. Each marker corresponds to the result obtained by the simulation method.

Table 1: Minimum tact time when collision probability is zero.

| $m_1$ | $t^*_{tact}$ |
|-------|--------------|
| 1     | 1.210        |
| 2     | 0.605        |
| 3     | 0.404        |
| 4     | 0.303        |
| 5     | 0.242        |

Following comparisons between theoretical values of collision probability using the formula and simulated values, we have confirmed that they are almost identical when $s = 1$ and $m_1 = 1$.

We consider the optimization problem presented in Ref. [3], which minimizes tact time by including collision probability as part of the input. Therefore, the minimum tact time under the constraint that the collision probability is zero is denoted by $t^*_{tact}$. Table 1 shows $t^*_{tact}$ for each case of $m_1 = 1, 2, \ldots, 5$ in Fig. 2, where the value of $t^*_{tact}$ decreases with an increasing value of $m_1$, and the value of $t^*_{tact}$ for $m_1 = j$ ($2 \leq j \leq 5$) is about the value of $t^*_{tact}$ for $m_1 = 1$, i.e. 1.22 divided by $j$.

Next, we carry out the experiments for other $k_l$s and $\lambda_l$s for $l = 1, 2$. Concretely speaking, when $s = 1$, $k_1 = \lambda_1 = 200, 400, \ldots, 2,000$; when $s = 2$, $k_1 = \lambda_1 = k_2 = \lambda_2 = 200, 400, \ldots, 2,000$. For each setting on $k_1, \lambda_1, k_2,$ and $\lambda_2$, although the expectation of the random variable is one, the variance depends on the setting. The results for $s = 1$ and $m_1 = 5$ are shown in Fig. 3. The results for $s = 2$ and $m_1 = m_2 = 5$ are shown in Fig. 4. From Figs. 3 and 4 we can confirm that, as the variance
The number of processing stages $s$ increases, both the collision probability and $t^*_tact$ increase.

Finally, we carry out experimentation for various values of $s > 2$. For this, the parameters are set to $k_l = \lambda_l = 1,000$ for any $l$, and $m_l = 5$ for any $l$. The results on the relationship between $s$ and $t^*_tact$ are shown in Fig. 5. From Fig. 5, we can confirm that $t^*_tact$ tends to increase with an increasing value of $s$. Moreover, we check the computational time to compute collision probability when the tact is set to $t^*_tact$. The results are shown in Fig. 6. From Fig. 6, it can be confirmed that CPU time increases with an increasing value of $s$, meaning collision probability can be computed in a practicable time for values of $s \leq 20$.

5 Conclusions

In this paper, we presented the simulation method to compute the collision probability in a new series-parallel machines model, which is a generalized model of the known models in Refs. [2, 3, 4, 12]. The simulation method is based on the collision decision method, which, for efficiency, applies the heap data structure. We also analyzed the time complexity of the simulation method. Moreover, we implemented the simulation method and made observations based on the computational results of collision probability.

Future work in this area could extend to a network model. Theoretical analysis of collision probability in the series-parallel machines model is another area worthy of further research.

ACKNOWLEDGMENTS

The authors would like to thank two anonymous reviewers for their indications regarding computational experimentation which have helped to reinforce the value of this paper.

References

[1] J. Blażeewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz: Scheduling Computer and Manufacturing Processes, Second Edition (Springer-Verlag, Berlin, 2001).

[2] E. Chiba, H. Fujiwara, and T. Ibaraki: A New Approach to Approximate the Collision Probability in an Automated Production Line. Information Technology Letters, 6 (2007), 1-4.

[3] E. Chiba, T. Asano, T. Miura, N. Katoh, and I. Mitsuaka: Collision Probability in an In-Line Machines Model. Transactions on Computational Science XIII, LNCS 6750 (2011), 1-12.

[4] E. Chiba, H. Fujiwara, Y. Sekiguchi, and T. Ibaraki: Collision Probability in an In-Line Equipment Model under Erlang Distribution. IEICE Transactions on Information and Systems, E96-D (3) (2013), 400-407.

[5] E. Chiba: Heuristics for the Buffer Allocation Problem with Collision Probability Using Computer Simulation. Mathematical Problems in Engineering, vol. 2015, Article ID 424370 (2015), 7 pages.
Computing Collision Probability in a Series-Parallel Machines Model

[6] E. Chiba: On the Redundancy of Delivery Time in an In-Line Machine Model. Journal of Information Processing, 24 (2016), 83-87.

[7] E. Chiba and T. Otsuka: Computing Collision Probability in a Series-Parallel Machines Model. Proceedings of the 19th Asia Pacific Industrial Engineering and Management Systems, Paper ID 161 (2018), 5 pages.

[8] D. P. Heyman and M. J. Sobel, ed.: Stochastic Models: Handbooks in Operations Research and Management Science, Vol. 2 (North-Holland, Amsterdam, 1990).

[9] L. Kleinrock: Queueing Systems, Volume I: Theory (Wiley-Interscience, 1975).

[10] K. Kunisawa and T. Honma, ed.: Applied Queueing Dictionary (Hirokawa Shoten, 1971).

[11] Mersenne Twister HP: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

[12] T. Otsuka and E. Chiba: A Framework for Computing Collision Probability in a Parallel Machines Model. International Journal of Japan Association for Management Systems, 10 (1) (2018), 117-124.

[13] M. L. Pinedo: Scheduling: Theory, Algorithms, and Systems, Fourth Edition (Springer, 2012).