Petri Net as a Manufacturing System Scheduling Tool

Dr. Tauseef Aized, Professor and Chair
Department of Mechanical, Mechatronics and Manufacturing Engineering, KSK Campus, University of Engineering and Technology, Lahore, Pakistan

1. Introduction

Scheduling problems arise when different types of jobs are processed by shared resources according to their technological precedence conditions. There is a need to determine the optimal input sequence of jobs and resource usage for a given job mix. Production scheduling problems are very complex and have been proved to be NP-hard problems. Different approaches to production planning and scheduling have been adopted which are as follows:

1. **Heuristic dispatching rules.** Good rules are obtained based on experience. These rules work but often not at the optimum level. They are also developed based on the system simulation models. But simulation models are often too specific to particular situations and hence the results cannot be very well generalized.
2. **Mathematical programming methods:** These have been extensively studied and can produce good results for specific systems. The mathematical models have to ignore many practical conditions in order to solve these models efficiently. These practical conditions such as material handling capacity, complex resource sharing and routing, and sophisticated discrete-event dynamics are very difficult to be mathematically and concisely described. The optimality will not hold if any parameters or structures change during an operational stage.
3. **Computational intelligence based approaches**: These include knowledge based systems, neural networks, and genetic algorithms. Knowledge-based systems have difficulty in acquiring the efficient rules and knowledge and the results cannot be guaranteed the best.
4. **Other methods:** Other approaches such as algebraic models and control theoretic methods are difficult to offer efficient solution methodologies. The methods based on CPM/PERT and queuing networks provide efficient solution methodologies but cannot describe shared resources, synchronization, and lot sizes easily.

Time-driven systems such as living organisms, ecological systems and world population have long been modeled and analyzed through difference/ differential equations. Since such equations have become a universal modeling framework for time-driven systems, analytical and numerical techniques have been developed to solve the equations in order to understand, control and optimize system behavior. Man-made technological environments such as computer, transportation and telecommunication networks or manufacturing and logistics systems represent systems whose behavior is governed by events occurring asynchronously over time. Events may be controlled (e.g.; release of a new job in a manufacturing facility) or uncontrolled (e. g; arrival of a customer request at the same manufacturing facility). Such systems are usually encountered whenever a set of tasks is to be performed by a set of resources requiring coordination of events, resource contention management and performance monitoring and optimization. Event-driven systems are of increasing importance in today's world because they are growing in number, size and sophistication. It is therefore imperative to have systematic design methodologies in order to achieve desirable performance and to avoid catastrophic failures. These systems may be asynchronous and sequential, exhibiting many characteristics: concurrency, conflict, mutual exclusion and non-determinism. These characteristics are difficult to describe using traditional control theory which deals with systems of continuous or synchronous discrete variables modeled by differential or difference equations. In addition, inappropriate control of the occurrence of events may lead to system deadlock, capacity overflows or may otherwise degrade system performance. These systems typically referred to as *discrete event dynamical systems (DEDS).*The following are the characteristics embedded in DEDS.

- ***Event-driven*: ** A discrete event system is characterized by a discrete state space where changes in state are triggered by event occurrences. Precedence is a key relation between events, that is, any event may be dependent on the occurrence of other events.
- ***Asynchronous*:** The asynchronous characteristic of discrete event systems is one of the most important properties by which they differ from traditional systems described by differential or difference equations. In time discretization of sampled systems, each change or step is synchronized by a global clock. In continuous systems, parameters vary continuously with time. However, in discrete event systems the events often occur asynchronously.
- ***Sequential Relation*:** Given a set of events, there may exist some sequential relationships among them. There is a sequential relation between two events if one event can occur only after the occurrence of the other.
- ***Concurrency*:** It means that there are no sequential relationships among the concerned events. For instance, two events are concurrent if either event may occur before the other.
- ***Conflict*:** It may occur when two or more processes require a common resource at the same time.
- ***Mutual exclusion:*** when conflict occurs, the events become mutually exclusive in the sense that they can not occur at the same time, whereas after one is complete, the other can occur.
- **Non-determinism:** Two kinds of non-determinism may occur. The first kind results from uncertain events' occurrence. For instance, if there is a conflict between two events, either of two events can occur randomly. The second kind of non-determinism results from small changes in process parameters, e.g.; processing time of an operation differ from time to time due to randomness , hence it can not be predicted accurately when an event will occur.
- ***System deadlock:*** A state may reach when none of the processes can continue. This can happen with the sharing of two resources between two processes and is usually the result of system design.

In order to capture the above properties, several mechanisms have been proposed and developed for modeling such systems. These are state machines, Petri nets, communicating sequential processes and finitely recursive processes. In order to conduct performance analysis of these kinds of systems, methods such as perturbation analysis, queuing network theory and Markov processes have been formulated and applied .An event-driven system can be abstracted as a state machine in which the states change when events occur. The finite state machine or automaton models results when the total number of states in a system is finite. However, when they are used to model discrete event system in a straight forward manner, the exponential increase in the number of states makes it very difficult to implement discrete event systems. Graphical representation is almost impossible and thus graphical visualization can not be easily realized .Some other models have also been developed for modeling and control of discrete event systems, e.g., supervisory control theory and finitely recursive processes. In supervisory control, the theory is elegant and is independent of the models used for applications. In most applications, each discrete event process is assumed to be modeled by an automaton or a state machine and its behavior is completely described by the language generated by the automaton. Many interesting theoretical results have been reported on controllability, observability and modular synthesis. However, the applicability to real-world distributed systems may be limited by the use of state machine representation. This approach encounters the state space explosion problem. Therefore, when a state machine is used to describe a complicated system, the design problem can easily become unmanageable. In addition, specifying the desirable language for a system is not easy. Finitely recursive processes (FRP) are mainly based on Hoare's communicating sequential processes. In the FRP formulation, given a set of events, a process is defined as a triple which consists of three components: a set of traces which the process can execute, an event function and a termination function. One of the important feature is that each process can be described as a set of recursive equations which implies that the description of a system can be implemented using equation forms. However, many problems remain open, e.g., the use of such equations to design supervisory controllers for real world systems.

2. Petri net as a DEDS modelling and scheduling mechanism

Petri net, as a graphical tool, provide a unified method for design of discrete event systems from hierarchical systems description to physical realizations. Compared with other models discussed above, they have the following advantages.

- Ease of modeling discrete event system characteristics: concurrency, asynchronous and synchronous features, conflicts, mutual exclusion, precedence relation, non-determinism and system deadlock,
- Ability to generate supervisory control code directly from the graphical Petri net representation,
- Ability to check the system for undesirable properties such as deadlock and instability,
- Performance analysis without simulation is possible for many systems. Production rates, resource utilization, reliability and performability can be evaluated.
- Discrete event simulation that can be driven from the model.
- Status information that allow for real-time monitoring
- Usefulness of scheduling because the Petri net model contains the system precedence relations as well as constraints on discrete event performance.

As a single representation tool, Petri net can aid in modelling, analysis, validation, verification, simulation, scheduling and performance evaluation at design stage. Once the system shows desirable behaviour, the net can be converted into control and monitor operations at run time. Therefore, Petri nets can be regarded as a powerful mathematical and graphical tool for design of various discrete events systems.

3. Petri net as a manufacturing system modelling mechanism

Modern manufacturing systems are highly parallel and distributed. They need to be analyzed from qualitative and quantitative points of view. Qualitative analysis looks for properties like the absence of deadlocks, the absence of overflows, or the presence of certain mutual exclusions in the use of shared resources. Its ultimate goal is to prove the correctness of the modeled system. Quantitative analysis looks for performance properties like throughput, responsiveness properties e.g., average completion times or utilization properties like utilization rates. Petri nets allow the construction of models amenable both for correctness and efficiency analysis. It can be considered as a graph theoretic tool especially suited to model and analyze Discrete Event Dynamical Systems (DEDS) which exhibit parallel evolutions and whose behavior are characterized by synchronization and sharing phenomena. Their suitability for modeling this type of system has led to their application in a wide range of fields. Examples of such DEDS are communication networks, computer systems and manufacturing systems. Petri nets have proven to be very useful in modeling, simulation and control of manufacturing systems. They provide very useful models for the following reasons:

- Petri nets capture the precedence relations and structural interactions of stochastic, concurrent and asynchronous events. In addition, their graphical nature helps to visualize such complex systems.
- Conflicts and buffer sizes can be modeled easily and efficiently.
- Deadlock in a system can be detected.
- Petri net models represent a hierarchical modeling tool with a well-developed mathematical and practical foundation.
- Various extensions of Petri nets allow for both qualitative and quantitative analyses of resource utilization, effect of failures and throughput rate.
- Petri net models give a structured framework for carrying out a systematic analysis of complex systems.
- Petri net models can also be used to implement real-time control for a flexible manufacturing system.

In this chapter, Petri net method is applied to model a semi-conductor manufacturing system, which is typically called multiple cluster tool system.

4. Multiple cluster tool system description

In a simple cluster tool, a material lot is loaded into a load-lock, pumped down to vacuum, and routed through a sequence of one or more modules in the cluster. After a wafer is completed, it is returned to the load-lock and after the completion of a lot, it is vented to atmosphere. In case loading/unloading time constitutes a significant component of total processing time, an improvement can be achieved by doubling the load-locks known as double load-lock cluster tool which can have throughput twice that of a single load-lock tool if material processing time is comparable with the sum of loading and unloading time. If the processing time is significantly greater than the loading/ unloading time, then the performance advantage of a double load-lock tool is rather insignificant. Processing

chambers normally consist of two or more processing modules. For instance, chemical vapour deposition (CVD) cluster tools usually have six or eight modules which are process modules with an aligner and a cooler module. Also, one or more of the process modules may behave as bottleneck with the maximum service demand because of longest processing time. An obvious approach for improving throughput of such a system is to duplicate the critical chamber and use both chambers alternatively to increase the throughput of the tool. A Cluster tool may be a single/ double or a multiple -blade tool. A single blade tool is one in which a robotic transporter can carry only one wafer at a time whereas a double blade cluster tool can carry two wafers at the same time. For cluster tools with many chambers, it may be beneficial to use several robots, each of which services a group of chambers. A cluster tool is operated by control software called a cluster tool controller which manages the job data, communicates with the process modules and wafer handling robots and coordinates their activities.
The considered multiple cluster tool system in this paper has four cluster tools. Each cluster has four processing modules for material processing and two load-locks one of which is for incoming material and the other is for outgoing material. Each cluster tool has a double-blade tool. The angle between two arms of the robotic transporter, that is, double blade-tool is always 180 degrees. The material between successive cluster tools is transported using automated guided vehicles as shown in Figure 1.

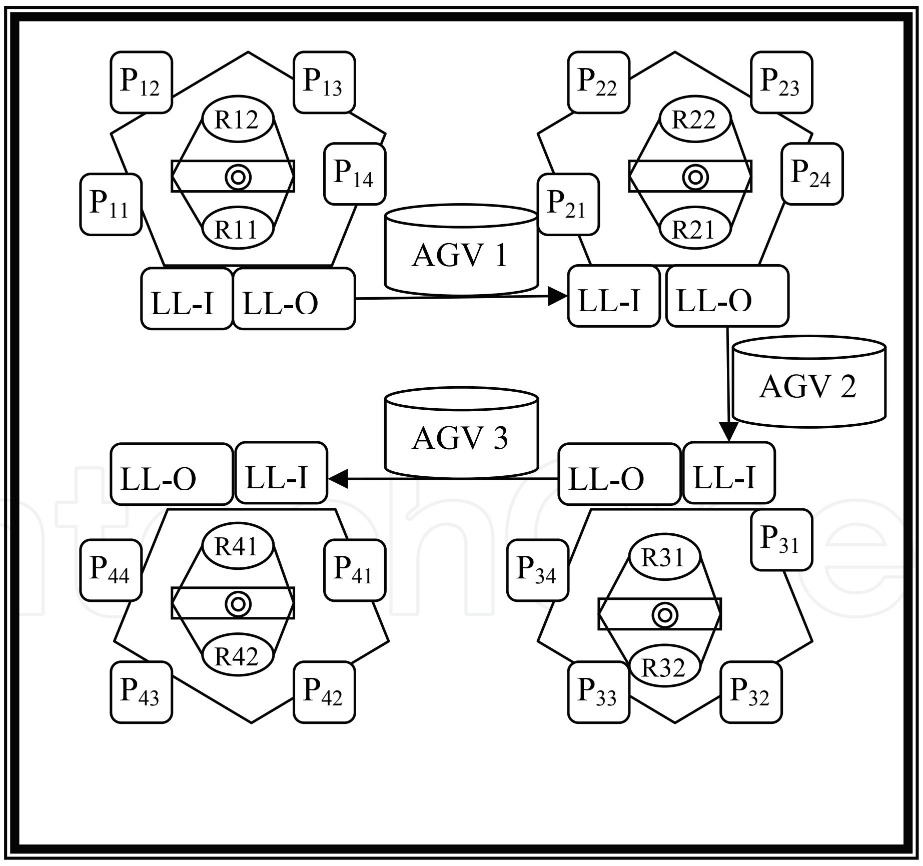
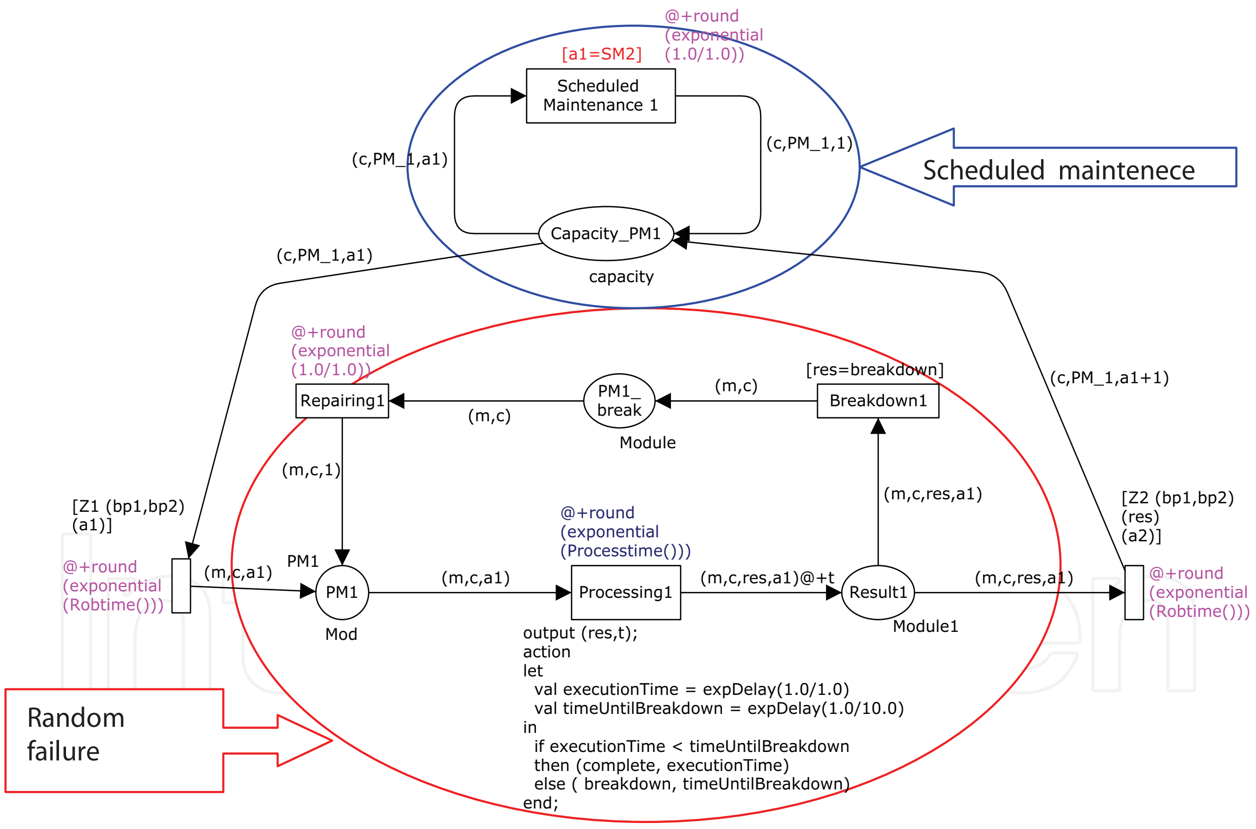


Fig. 1. The multiple cluster tool system.
Legend: P_{ij} – processing module 'j' of ith cluster tool, R_{ik}- robot arm 'k' for ith cluster tool, LL-I --load-lock for incoming material, LL-O -- load-lock for outgoing material, AGV-automated guided vehicle

5. Coloured Petri net

A semi-conductor manufacturing system is a discrete event dynamical (DEDS) system, which is asynchronous, parallel, and event driven in its nature. A DEDS can be characterized by events and conditions, which can be described by Petri net method easily. In a semi-conductor manufacturing system, events are occurring in a parallel way that can be modelled compactly by coloured Petri net method. A Petri net consists of places, transitions and directed arcs represented by circles, rectangular bars and arrows, respectively. Arcs run between places and transitions. Places may contain any number of tokens. A distribution of tokens over the places of a net is called a marking. Transitions act on input tokens by a process known as firing. A transition can fire if it is enabled, i.e., there are tokens in every input place. When a transition fires, it consumes the tokens from its input places, perform some processing task and places a specified number of tokens into each of its output places. The conditions of a DEDS are described by places, events are described by transitions, relations between events and conditions are described by arcs and holding of conditions are described by tokens in places. The occurrences of events are described by firing of transitions which remove tokens from input places and add tokens to output places and the behaviour of a system is described by firing of transitions and movements of tokens. Places, transitions and tokens must be assigned a meaning for proper interpretation of a model. In manufacturing systems, normally places represent resources like machines, materials etc. and the existence of one or more tokens in a place represents the availability of a particular resource, while no token indicates that the resource is unavailable. A transition firing represents an activity or process execution, for instance, a machining process. Places and transitions together represent conditions and precedence relationships in a system's operation.
Sorensen and Janssens (2004) presented a Petri net model of a continuous flow transfer line with unreliable machines. This study proposed a Petri net in which a place represents the state of a machine or of a buffer. For each machine, four places are added indicating that the machine is up, down, blocked or starved. The proposed scheme is not suitable for practical manufacturing systems which have a great many conditions and events and modelling through Sorensen and Janssens (2004) method will generate a too complex and intractable models. Gharbi and Loualalen (2006) provided a detailed analysis of finite-source retrial systems with multiple servers subject to random breakdowns and repairs using generalized stochastic Petri net models. Cao et al. (2007) presented a queuing generalized stochastic coloured timed Petri net (QGSCTPN) based approach for modelling of semiconductor wafer fabrication. In the proposed QGSCTPN, Cao et al. (2007) introduced two kinds of places and five kinds of transitions and presented a small minfab model re-entrant line with three machine groups. The study emphasized to further explore methods to model large semiconductor manufacturing systems with practical constraints like random failures. Zhou and Venkatesh (1999) presented augmented timed Petri net for modelling and analysing manufacturing systems with breakdowns with the help of deactivation places, transitions and arcs but resource breakdowns can be handled compactly and more effectively through non-hierarchical and hierarchical coloured Petri net , defined in (Jensen, 1992), and is explained in the following paragraph. Additionally, in contrast to the papers mentioned where resource breakdowns have been modelled either before or after a processing activity, this study proposes a modelling approach which can model not only before or after a processing operation but also a resource breakdown activity can take place when a resource is carrying out its operation. This approach is more practical as resources are subject to breakdowns while they are performing their operations.

In this study, the random failures of all material processing modules are modelled in the same way and is elaborated by an example shown in Figure 2. This modelling feature is carried out through a transition 'Processing1'. There is a code segment attached to this transition which is executed each time the transition occurs (fires). This code is used to declare two *values*, that is, value *exec time* (execution time) and value *time break* (time until breakdown) as exponential distribution functions. As this code is executed, the exponential functions generate values for execution time and time until breakdown based on mean execution or processing time of the module. These two values are compared in such a way that if execution time is less than time until breakdown then the result is success; otherwise it is failure. The success means the operation has been executed successfully and the failure means that the module has been breakdown during execution. The arc variables correspond to the relevant places in Figure 2. The scheduled maintenance is modelled in such a way that process modules are allowed to perform a fix number of processing operations after which the modules become unavailable and are repaired/ maintained and then again modules can perform processing operations for a specific number of times. This process-repair-process cycle repeats itself indefinitely. Figure 3 shows the snapshot of the CPN model taken from CPN Tools whereas Table 1 describes places and transitions used in the model.

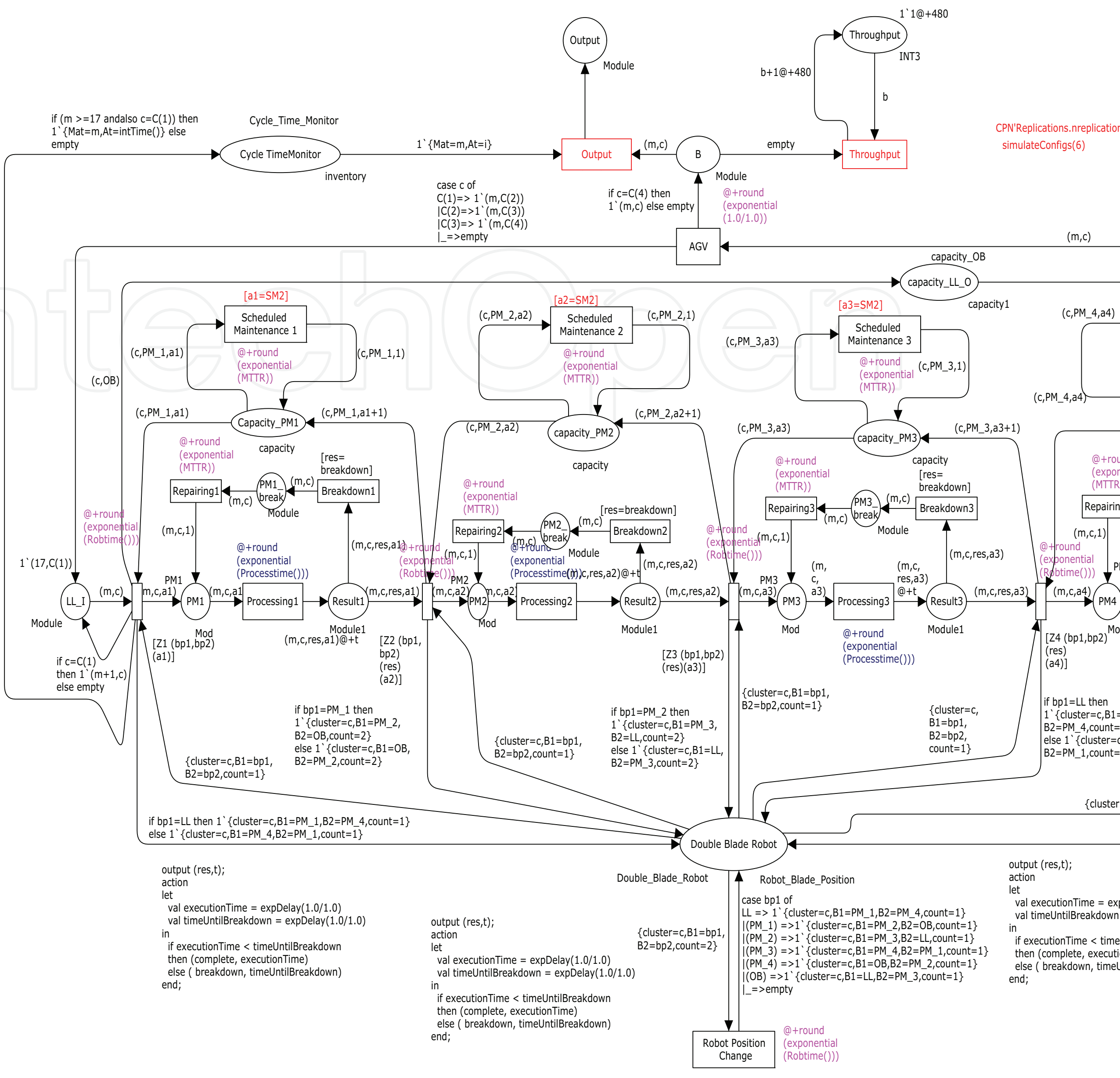


```
output (res,t);
action
let
  val executionTime = expDelay(1.0/1.0)
  val timeUntilBreakdown = expDelay(1.0/10.0)
in
  if executionTime < timeUntilBreakdown
  then (complete, executionTime)
  else ( breakdown, timeUntilBreakdown)
end;
```

Fig. 2. The random failure and scheduled maintenance using CPN Tools.

6. Model development

The model is developed using CPN Tools which is a CPN-based program developed on the basis of CPN ML language. The CPN ML language is derived from Standard ML which is a general purpose functional language. All material loading/unloading, processing, repair


Cycle_Time_Monitor
Cycle TimeMonitor
inventory
if (m >=17 andalso c=C(1)) then 1`{Mat=m,At=intTime()} else empty
1`{Mat=m,At=i}
Output
Module
Output
(m,c)
B
Module
empty
Throughput
1`1@+480
INT3
b+1@+480
b
Throughput
CPN'Replications.nreplication
simulateConfigs(6)
case c of C(1)=> 1`(m,C(2)) |C(2)=>1`(m,C(3)) |C(3)=> 1`(m,C(4)) |_=>empty
if c=C(4) then 1`(m,c) else empty
@+round (exponential (1.0/1.0))
AGV
(m,c)
capacity_OB
capacity_LL_O
capacity1
(c,OB)
[a1=SM2]
Scheduled Maintenance 1
(c,PM_1,a1)
(c,PM_1,1)
@+round (exponential (MTTR))
(c,PM_1,a1)
Capacity_PM1
(c,PM_1,a1+1)
capacity
@+round (exponential (MTTR))
[res= breakdown]
Repairing1
(m,c)
PM1_ break
(m,c)
Breakdown1
Module
(m,c,1)
@+round (exponential (Robtime()))
@+round (exponential (Processtime()))
(m,c,res,a1)
1`(17,C(1))
LL_I
(m,c)
Module
m,c,a1)
PM1
PM1
Mod
(m,c,a1
Processing1
Result1
Module1
(m,c,res,a1)@+t
(m,c,res,a1)
[Z1 (bp1,bp2) (a1)]
if c=C(1) then 1`(m+1,c) else empty
[a2=SM2]
Scheduled Maintenance 2
(c,PM_2,a2)
(c,PM_2,1)
@+round (exponential (MTTR))
(c,PM_2,a2)
capacity_PM2
(c,PM_2,a2+1)
capacity
@+round (exponential (MTTR))
[res=breakdown]
Repairing2
PM2_ break
(m,c)
Breakdown2
Module
(m,c)
(m,c,1)
@+round (exponential (Robtime()))
@+round (exponential (Processtime()))
(m,c,res,a2)@+t
(m,c,res,a2)
(m,c,a2)
PM2
PM2
Mod
m,c,a2
Processing2
Result2
Module1
(m,c,res,a2)
[Z2 (bp1, bp2) (res) (a2)]
[Z3 (bp1,bp2) (res)(a3)]
if bp1=PM_1 then 1`{cluster=c,B1=PM_2, B2=OB,count=2} else 1`{cluster=c,B1=OB, B2=PM_2,count=2}
{cluster=c,B1=bp1, B2=bp2,count=1}
{cluster=c,B1=bp1, B2=bp2,count=1}
if bp1=PM_2 then 1`{cluster=c,B1=PM_3, B2=LL,count=2} else 1`{cluster=c,B1=LL, B2=PM_3,count=2}
if bp1=LL then 1`{cluster=c,B1=PM_1,B2=PM_4,count=1} else 1`{cluster=c,B1=PM_4,B2=PM_1,count=1}
[a3=SM2]
Scheduled Maintenance 3
(c,PM_3,a3)
(c,PM_3,1)
@+round (exponential (MTTR))
(c,PM_3,a3)
capacity_PM3
(c,PM_3,a3+1)
capacity
@+round (exponential (MTTR))
[res= breakdown]
Repairing3
(m,c)
PM3_ break
(m,c)
Breakdown3
Module
(m,c,1)
@+round (exponential (Robtime()))
(m,c,res,a3)
(m,c,a3)
PM3
(m, c, a3)
PM3
Mod
Processing3
(m,c, res,a3) @+t
Result3
Module1
@+round (exponential (Processtime()))
(m,c,res,a3)
{cluster=c,B1=bp1, B2=bp2,count=1}
{cluster=c, B1=bp1, B2=bp2, count=1}
(c,PM_4,a4)
(c,PM_4,a4)
@+rou (expor (MTTR
Repairin
(m,c,1)
@+round (exponential (Robtime()))
(m,c,a4)
PM4
Mo
[Z4 (bp1,bp2) (res) (a4)]
if bp1=LL then 1`{cluster=c,B1= B2=PM_4,count= else 1`{cluster=c B2=PM_1,count=
{cluster
Double Blade Robot
Double_Blade_Robot
Robot_Blade_Position
{cluster=c,B1=bp1, B2=bp2,count=2}
@+round (exponential (Robtime()))
Robot Position Change

```
case bp1 of
LL => 1`{cluster=c,B1=PM_1,B2=PM_4,count=1}
|(PM_1) =>1`{cluster=c,B1=PM_2,B2=OB,count=1}
|(PM_2) =>1`{cluster=c,B1=PM_3,B2=LL,count=1}
|(PM_3) =>1`{cluster=c,B1=PM_4,B2=PM_1,count=1}
|(PM_4) =>1`{cluster=c,B1=OB,B2=PM_2,count=1}
|(OB) =>1`{cluster=c,B1=LL,B2=PM_3,count=1}
|_=>empty
```

```
output (res,t);
action
let
  val executionTime = expDelay(1.0/1.0)
  val timeUntilBreakdown = expDelay(1.0/1.0)
in
  if executionTime < timeUntilBreakdown
  then (complete, executionTime)
  else ( breakdown, timeUntilBreakdown)
end;
```

```
output (res,t);
action
let
  val executionTime = expDelay(1.0/1.0)
  val timeUntilBreakdown = expDelay(1.0/1.0)
in
  if executionTime < timeUntilBreakdown
  then (complete, executionTime)
  else ( breakdown, timeUntilBreakdown)
end;
```

```
output (res,t);
action
let
  val executionTime = exp
  val timeUntilBreakdown
in
  if executionTime < time
  then (complete, executi
  else ( breakdown, timeU
end;
```

Fig. 3. The multiple cluster tool system model developed using CPN Tools.

Places	Description	Transitions	Description
LL_I	Load-lock for incoming material	Processing 1,2,3,4	Processing activity in progress at module 1,2,3,4
LL_O	Load-lock for outgoing material	Breakdown1,2,3,4	Process modules breakdowns
PM1,2,3,4	Processing modules 1,2,3 and 4	Repairing1,2,3,4	Process module 1,2,3,4 under repair
Result1,2,3,4	Result of processing modules 1,2,3,4 respectively (Material processing successful or module breakdown during processing)	Scheduled Maintenance1,2,3,4	Process module 1,2,3,4 under scheduled maintenance
PM1,2,3,4_Break	Processing module 1,2,3,4 in breakdown condition	AGV	Material transportation between successive cluster tools
Capacity_PM1,2,3,4	Capacity specification for Processing module 1,2,3,4	Robot Position Change	Change of position of double blade robot tool
Capacity_LL_O	Capacity specification for load-lock for outgoing material		
Blade_Position_Robot	Orientation specification of double blade tool		
Cycle Time Monitor	Monitoring cycle time		

Table1. Places and transitions description

and transportation operations have been modelled using exponential distribution functions. The following simulation assumptions have been used in developing the model:

- The raw material is always available.
- All material handling robot times are the same.
- All module processing times are the same.
- All module repair times are the same.
- The automated guided vehicles for material transportation between clusters are always available.

These assumptions can be easily relaxed in coloured Petri net modelling, if required. Before collecting the resulting data, it is important to detect the warm-up period to access the steady state behaviour of the system. This study uses four stage SPC approach (Robinson, 2002) to find out the steady state results. The warning Limit (WL) and action Limit (AL) are calculated as under:

$$WL = \hat{\mu} \pm 1.96\hat{\sigma} / \sqrt{n}\ ,\ AL = \hat{\mu} \pm 3.09\hat{\sigma} / \sqrt{n}$$

Where $\hat{\mu}$ is mean value, $\hat{\sigma}$ is standard deviation and n is number of replications. As the input factors vary for individual simulation runs in this study, hence the warm-up period is also varying depending on the particular input factor settings. In general, ten independent replications have been carried out for each input factor settings and the warm-up period has been determined. This period has been excluded while collecting data from the simulation runs and the length of the steady state period has been determined according to the recommendations given in (Robinson, 2002) and hence simulation results are repeatable.

7. Results and discussion

There are three process execution input factors in this model, which are mean material handling robot loading/unloading time, MLT (minutes), mean module processing time, MPT (minutes) and mean time to repair related to processing module, MTTR (minutes) and two input factors represent resource breakdowns which are scheduled maintenance, SM (number of jobs) and mean time to failure, MTF (minutes). The impact of these factors on throughput (mean number of products per day) and cycle Time (mean number of minutes) is studied. The simulation results are given in Table 2.

In order to avoid the effects of different values, the same values are considered for respective ten levels of input factors. The throughput is decreasing as mean loading/ unloading time (MLT) is increasing because an increase in MLT requires materials to stay in the system for a longer time which causes an increase in cycle time and a corresponding decrease in throughput. The throughput is also decreasing with an increase of mean processing time (MPT) because more processing time means an increase in cycle time which causes a decrease in throughput. Mean loading/ unloading time has more adverse effect than mean processing time because for each module processing operation, there is one loading and another unloading operation. But this only holds true if the values of both MLT and MPT are the same. In an event where MPT is far more than MLT, it can impact throughput more adversely than MLT. Figures 4 and 5 show the impact of MLT and MPT on throughput and cycle time respectively.

SM and MTF are the measures of scheduled maintenance and random failures respectively. The scheduled maintenance is modelled in such a way that each processing module is down after a specific number of processing operations and is repaired after which it is again up for further processing. Figures 6 and 7 show an increase in throughput with a corresponding decrease in cycle time when the value of SM increases because an increase in SM value indicates that the processing module is capable of more operations before it needs to be repaired. MTF which represents random failure of modules during process execution has a more significant impact on throughput and cycle time. MTF is the mean time computed from the moment when a module starts process execution to the moment when that module breaks down provided the process execution is in progress. Hence MTF is computed each time when a module starts execution of a process and it is compared with process execution time to decide a success or failure of a process. If a module is processing a material/part and it breaks down during process execution, the material/part has to be unloaded from the resource and has to be reloaded on it after repair. This causes time wastage and increases cycle time and hence causes a decrease in throughput. Thus the values of MTF must be kept higher than mean values of process executions like MLT and MPT in order to achieve higher throughput and lower cycle time. At value 1, the values of MTF and process execution times are equal and hence throughput is lower due to a greater probability of resource breakdown during process execution. As the value of MTF increases up to value 2, there is a significant gain in throughput and decrease in cycle time because of a lower probability of resource

		Throughput			Cycle Time		
		Mean	95% CI	St.Dev	Mean	95% CI	St Dev
Robot loading/unloading time (MLT)	1	28.05	0.11	0.15	837.18	195.78	273.71
	2	20.87	0.08	0.11	1027.28	113.47	158.63
	3	15.67	0.06	0.09	1208.48	153.98	215.26
	4	12.28	0.05	0.06	1570.60	997.06	175.30
	5	10.01	0.05	0.07	1614.59	169.19	236.53
	6	8.39	0.05	0.07	1641.79	149.16	208.53
	7	7.20	0.05	0.08	2011.48	179.98	251.61
	8	6.30	0.02	0.03	2227.57	166.52	232.79
	9	5.59	0.04	0.06	2256.77	117.27	163.95
	10	5.00	0.03	0.05	2276.05	193.72	270.82
Module processing time (MPT)	1	28.09	28.09	0.16	831.39	94.86	132.62
	2	21.32	0.09	0.13	893.37	174.07	243.35
	3	16.96	0.07	0.10	948.82	173.09	241.98
	4	13.94	0.08	0.11	997.06	242.91	339.58
	5	11.87	0.06	0.09	1290.65	222.99	311.74
	6	10.26	0.09	0.13	1497.57	298.12	416.78
	7	9.04	0.09	0.13	1571.07	270.30	377.88
	8	8.12	0.06	0.08	2059.10	379.50	530.54
	9	7.31	0.07	0.10	2109.55	339.60	474.75
	10	6.73	0.08	0.11	2199.64	237.52	332.05
Mean time to repair (MTTR)	1	27.92	0.08	0.11	873.00	111.12	155.34
	2	20.68	0.13	0.18	1019.70	141.58	197.92
	3	16.07	0.11	0.15	1240.12	252.58	353.11
	4	13.17	0.05	0.07	1277.15	158.13	221.07
	5	11.09	0.08	0.11	1473.04	228.37	319.26
	6	9.52	0.08	0.11	1726.99	262.69	367.23
	7	8.37	0.05	0.07	1818.93	146.24	204.44
	8	7.46	0.07	0.10	1957.89	305.50	427.09
	9	6.74	0.06	0.08	2089.78	239.70	335.10
	10	6.12	0.05	0.07	2265.71	393.50	550.11
Scheduled maintenance (SM)	1	28.081	0.11	0.15	802.21	155.72	217.69
	2	29.373	0.08	0.12	761.51	132.57	185.33
	3	30.109	0.14	0.20	701.59	69.79	97.57
	4	30.95	0.12	0.16	663.18	190.21	265.91
	5	31.82	0.14	0.19	619.23	131.63	184.02
	6	33.04	0.20	0.28	576.98	144.21	201.61
	7	33.56	0.14	0.19	524.36	142.07	198.62
	8	34.06	0.10	0.14	490.37	137.48	192.19
	9	34.26	0.15	0.20	466.82	144.05	201.38
	10	34.59	0.09	0.13	449.38	142.16	198.73
Mean time to failure (MTF)	1	28.09	0.12	0.17	811.88	124.32	173.80
	2	36.24	0.10	0.14	583.03	83.10	116.18
	3	39.06	0.12	0.17	513.19	78.09	109.17
	4	40.24	0.10	0.15	460.94	85.71	119.82
	5	40.99	0.13	0.18	416.65	55.59	77.71
	6	41.47	0.09	0.13	380.64	84.47	118.09
	7	41.82	0.06	0.09	343.11	79.31	110.88
	8	41.96	0.07	0.10	319.81	84.10	117.57
	9	42.20	0.11	0.16	309.41	84.56	118.22
	10	42.34	0.16	0.22	298.08	40.01	55.94

Table 2. The simulation results (CI: Confidence interval)

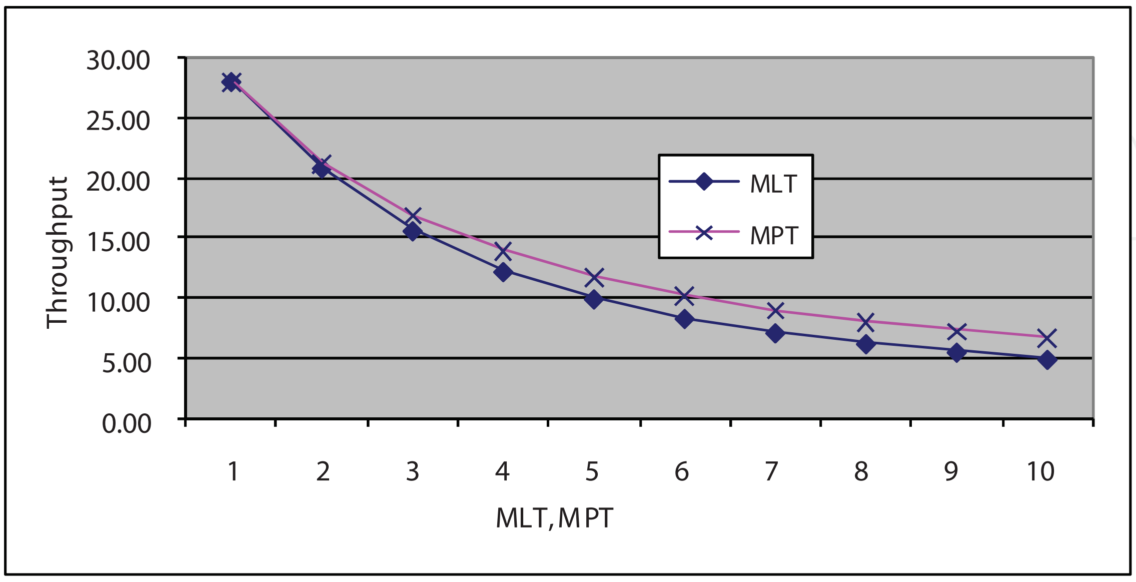


Fig. 4. The impact of robot loading/ unloading time and module processing time on throughput.

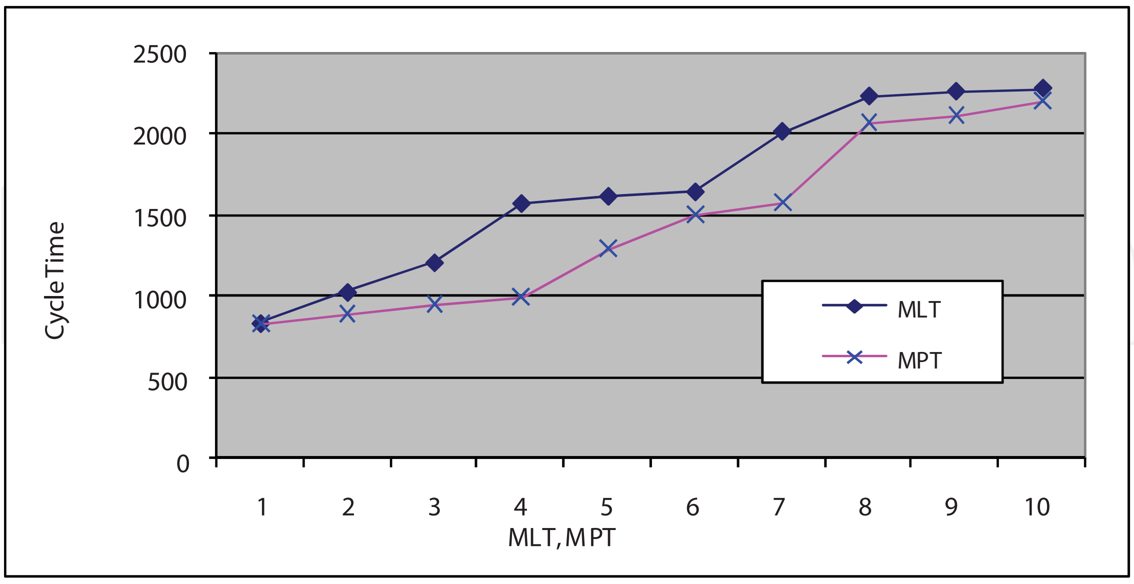


Fig. 5. The impact of robot loading/ unloading time and module processing time on Cycle time.

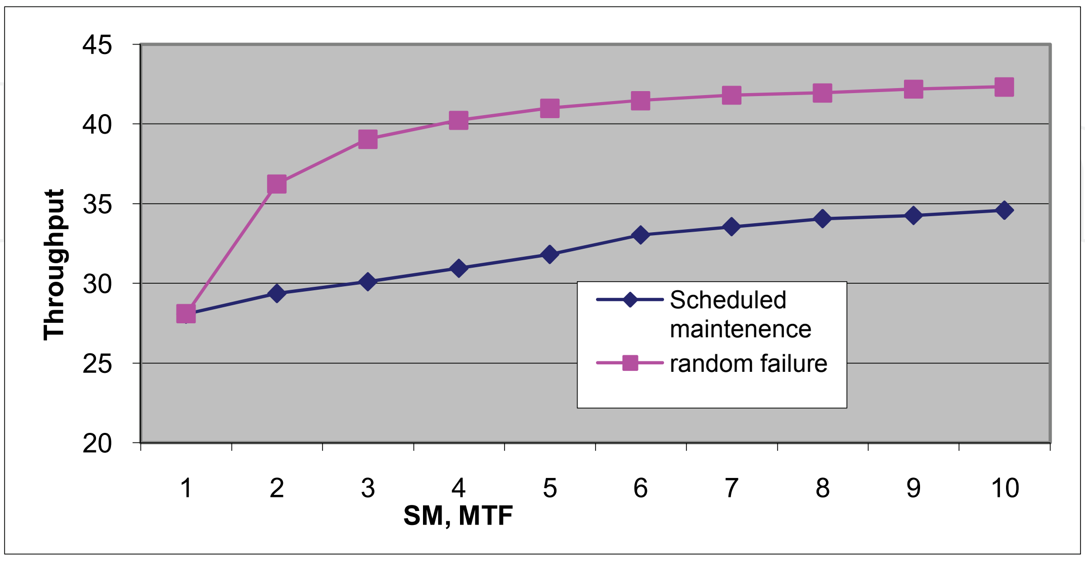


Fig. 6. The impact of scheduled maintenance and random failures on throughput.

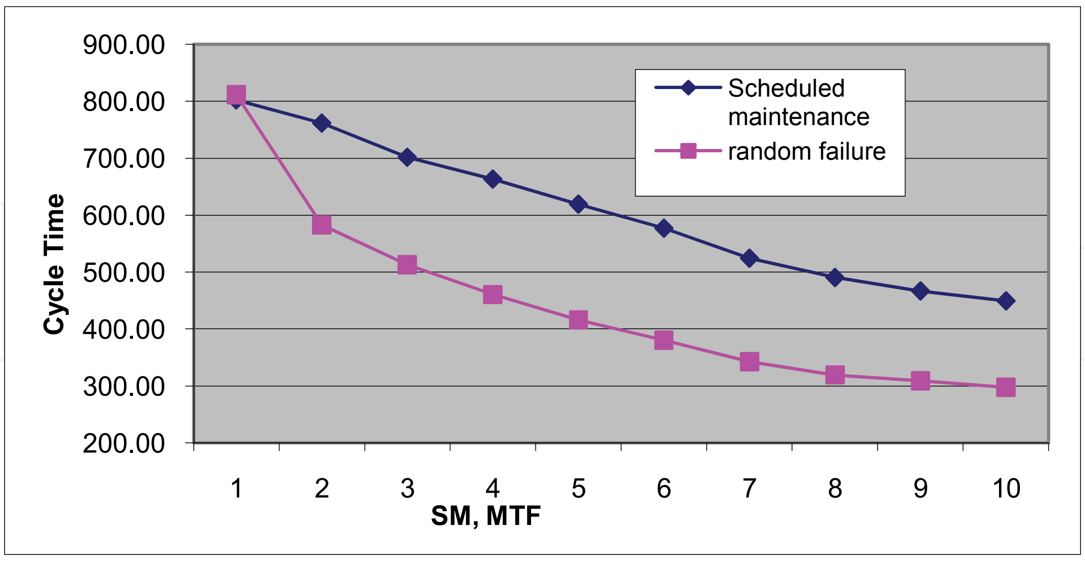


Fig. 7. The impact of scheduled maintenance/ random breakdowns on Cycle Time.

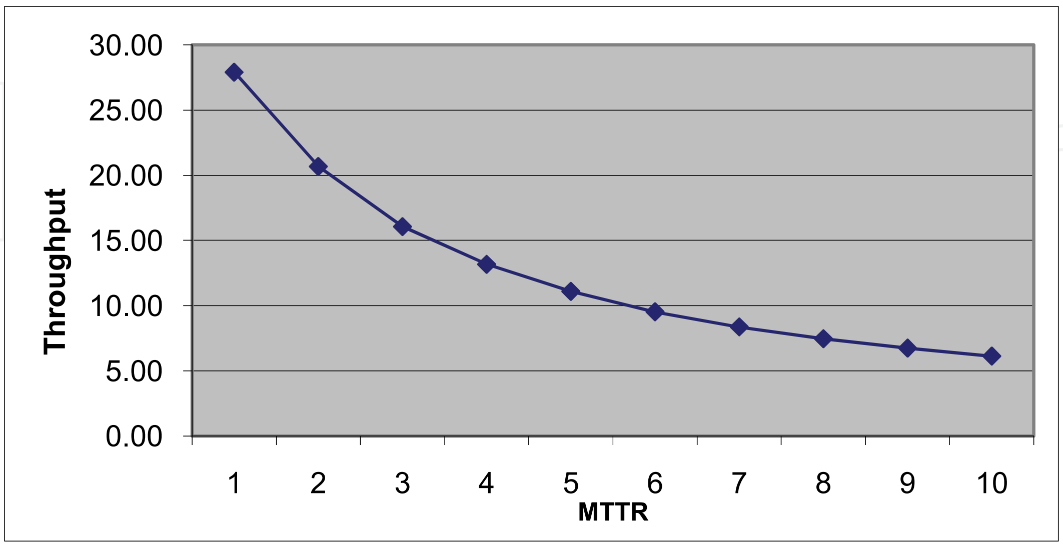


Fig. 8. The impact processing module mean time to repair on throughput.

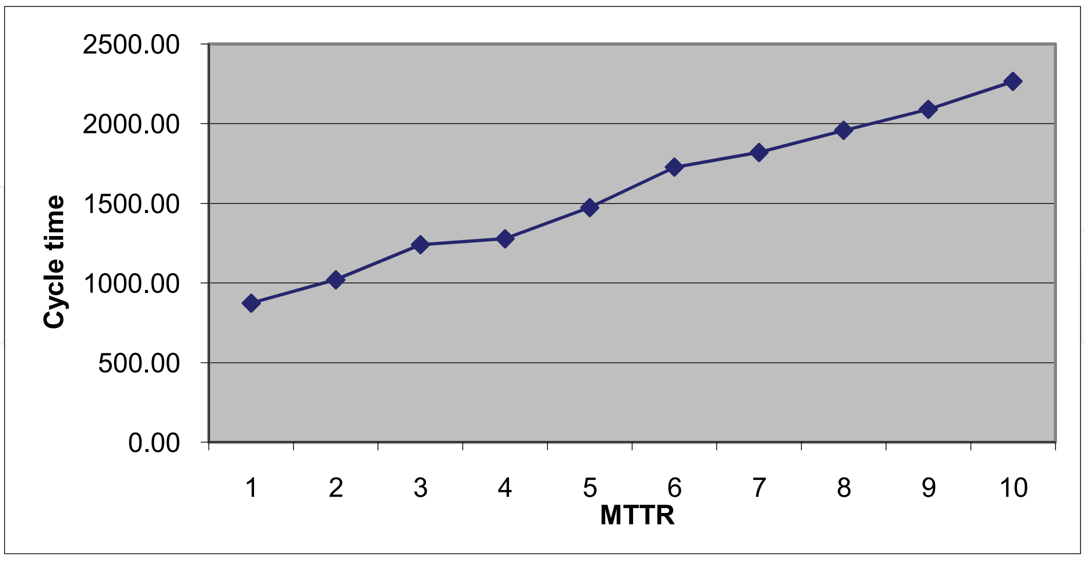


Fig. 9. The impact processing module mean time to repair on Cycle time.

breakdown during execution. This trend continues as MTF approaches higher values but the rate of increase in throughput and decrease in cycle time is reduced because after a certain value, value '7' in Figures 6 and 7, the probability of resource breakdown during execution is quite low. A resource with very high value of MTF, for instance 10, compared with its execution time means that it is highly reliable for the process. Such a resource is usually costly for the process and cannot contribute significantly, for instance compared with 7, 8 and 9 towards throughput and cycle time as is shown in Figures 6 and 7.
Figures 8 and 9 show the impact of processing module mean time to repair (MTTR) on throughput and cycle time. An increase in MTTR means more time is required to repair the processing module which decreases the working time to down time ratio of processing modules. The decrease in working to down times of processing modules forces the materials to stay a longer time in the system; thus increasing cycle time which deteriorates throughput of the system. In order to achieve higher throughput and lower cycle time values, the repair time of processing modules must be kept as low as possible.

8. Conclusion

Multiple cluster tool systems have emerged as an important semiconductor manufacturing system technology with the benefits of higher yield, shorter cycle time and tighter process control. This study has described a modelling technique using coloured Petri net to capture random failures of multiple cluster tool system and has shown that random failure is an important system limitation as far as throughput and cycle time are concerned. The random failures of cluster tools may be avoided by achieving a higher value of mean time between failures of process modules compared with process execution times. This multiple cluster tool system problem under discussion can be extended to incorporate the random failure modelling of internal robotic arms and AGV based transporters between clusters.

9. Acknowledgement

This chapter is partially based on the following publications.
1. Tauseef Aized "CPN and DoE based modeling and performance maximization of multiple product flexible manufacturing system with resource breakdowns". published by LAMBERT Academic Publishing, Germany,2010, ISBN: 978-3-8383-3888-0
2. Tauseef Aized "Modeling and analysis of multiple cluster tools system with random failures using colored Petri net" accepted, International Journal of Advanced Manufacturing Technology published by Springer ISSN: 0268-3768 (print version), ISSN: 1433-3015 (electronic version).
3. MengChu Zhou and Kurapati Venkatesh, "Modeling, Simulation and Control of Flexible manufacturing system: A Petri net approach". Published by World Scientific, 1999.

10. References

Sorensen, K. and Janssens, G.K. (2004). A Petri net model of a continuous flow transfer line with unreliable machines. European Journal of operational research, 152, pp. 248-262.

Gharbi, N., Loualalen, M. (2006). GSPN analysis of retrial systems with servers breakdowns and repairs. Applied mathematics and computation, 174, pp. 1151-1168.

Cao, Z.C., Qaio, F., Wu, Q. (2007). Queuing generalized stochastic colored timed Petri nets-based approach to modelling for semiconductor wafer fabrication. IEEE International Conf. on Control and Automation, Guangzhou. China, pp. 2834-2838.

MengChu Zhou and Kurapati Venkatesh. (1999). Modeling, simulation and control of flexible manufacturing systems. World Scientific Press.

Robinson, S. (2002). A statistical process control approach for estimating the warm-up period. Proceedings of 2002 winter simulation conference, pp. 439-445.

Jensen, K. (1992). Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 1, Springer-Verlag.