Systemic Solutions to Deadlock in FMS

Xu gang, Wu zhi Ming

Abstract—**In order to solve deadlock in FMS, an integrated design method for FMS is presented. This method is based on deadlock free scheduling, deadlock avoidance algorithm and deadlock detection and recovery. Deadlock is resolved from views of scheduling and controlling, which is a complete resolution to deadlock. In scheduling, deadlock free scheduling is applied. In the on line control, deadlock avoidance algorithm is used in the controller, which is an improvement and modification of the Banker's algorithm. This algorithm is verified with model checking tools: Spin. The advantage of this work lies in the complete solution to deadlock for FMS. This solution can control deadlock while preserving the system performance.**

I. Introduction

Manufacturers must adapt to the changes in the production environment as well as in the market in order to achieve and maintain global competitiveness. Flexible manufacturing systems (FMS) when designed and operated effectively, can be of assistance to manufacturers according to a defined production and process plans, which specify the activities and resources as well as sequence related conditions, such as precedence relations and synchronization. Alternative resources and routing for some of the jobs in FMS may be specified previously. This results in an increasing flexibility in job scheduling. Some resource allocation may lead to a deadlock situation, in particular when the resources in a system are limited. A deadlock is a state where a set of parts is in "circular waiting", i.e., each part in the deadlock set waits for a resource held by another part in the same set. Deadlock problems can cause unnecessary cost (e.g. long down-time and low use of some critical and expensive resources), particularly important to be solved in Flexible Manufacture Systems. Therefore, to develop efficient algorithms to improve and optimize the system performances while preventing deadlock situations becomes a basic requirement in running an FMS[13]. There are three strategies to deal with deadlock [3][7]: deadlock prevention, deadlock avoidance, deadlock detection and recovery.

There are many research papers devoted to deadlock problem. In [1] a deadlock avoidance algorithm is proposed for a class of Petri net models formed for flow shop manufacturing where a set of sequential processes are executed without alternating the order of using resources in each case. The algorithm controls the input flow of new tokens in a local area, ensuring that token evolutions in a system are always possible. Ezpeleta et al in [3] has proposed a policy for resource allocation based on the addition of new places to the net imposing restrictions that prevent the presence of unmarked siphons that may directly generate deadlocks[2]. The crucial point of the procedure is the complexity that involves in computing the set ofsiphons of the Petri net model. For a general class of Petri net models, in [7] both deadlock prevention and avoidance control policies are proposed. The first part is based on the net reachability graph, while the second part is based on a look-ahead procedure that searches for deadlock situations by simulating the evolution process of a system for a preestablished number of steps. Due to the fact that the avoidance policy does not assure that deadlocks are not reachable in future, they propose to combine this policy with a deadlock recovery system. In [5] Wu and Zhou point out that, if an Automated Manufacturing System (AMS) operates at the deadlock boundary, i.e., under the maximally permissive control policy, it will not be deadlocked but a blocking may occur more likely. They present an AMS that works near but not at the deadlock boundary in order to gain the highest productivity. For the first time he presents such a policy: Liveness-policy. Without being too conservative, it can effectively reduce or even eliminate the blocking possibility that exists under a maximally permissive control policy. In [8], Xu and Wu give an algorithm that can find the optimal deadlock-free scheduling without considering the buffer. An algorithm for deadlock-free scheduling with buffer allocation is presented in [12]. In [10] [11], deadlock prevention algorithm and deadlock avoidance algorithm are presented.

This paper is based on several work finished by the authors. They are organized effectively to resolve deadlock in FMS. This paper is organized as follows: the problem is presented in Section 2, the structure of the solution is introduced in Section3, the scheduler is given in Section 4, the system database in 5, controller in 6 and a conclusion in Section 7.

This work is supported by the National Natural and Science Foundation (Grant No. 60074011, 70071017).

Xu gang is a Ph.D from the Automation department of Shanghai Jiaotong University, Shanghai, China. Zip code 200030. (e-mail: steel_xu@yahoo.com).

Wu zhiming, is a professor with Shanghai Jiaotong University. He is now with the Department of Automation, Shanghai Jiaotong University (e-mail: ziminwu@sjtu.edu.cn).

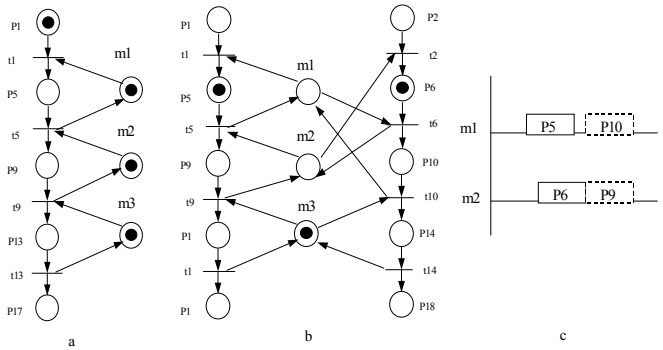

Fig.1 Petri net model of manufacturing system and its Gannt chart

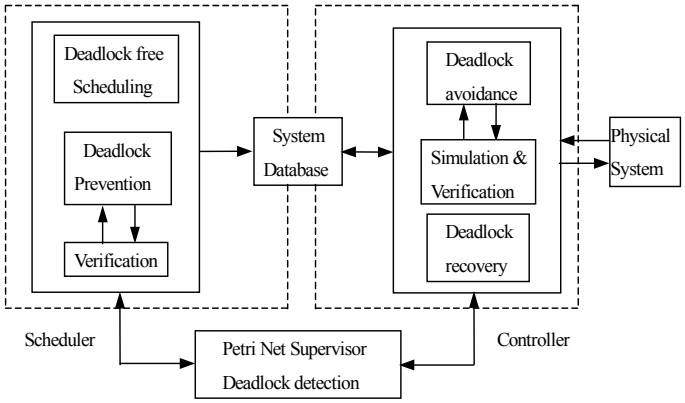

Fig.2. System Architecture

II. MODEL WITH PETRI NET

In this paper, a system is modeled with Systems of Simple Sequential Processes with Resources (S^3PR) Petri Net. A bottom-up method is used to model the whole system. Firstly, classify the whole system into several sub-systems, construct a sub-model for each sub-system; then merge the sub-models into a whole model. The places in the whole Petri net are separated into operation places and resource places. Tokens in an operation place represent that the operation is being executed, no token place represents that the operation is not. Token in the resource place represents that a resource is idle, no token in the resource place represents that the resource is being occupied. The sub-model is shown in Fig1.a.

Deadlock is a kind of system status, in which a set of parts enter into a waiting loop, each part in the set waits the resource occupied by another part in the set. Just like what the Fig1.b has demonstrated, the tokens in P5 and P6 are waiting for resources of M2 and M1 which are just occupied by the others. Thus the system can't evolve. The Gantt chart of the scheduling corresponding to the Fig.1.b is shown in Fig.1.c. The structure in Fig.1 can be extended to more machines and jobs with different routing procedures to describe a FMS.

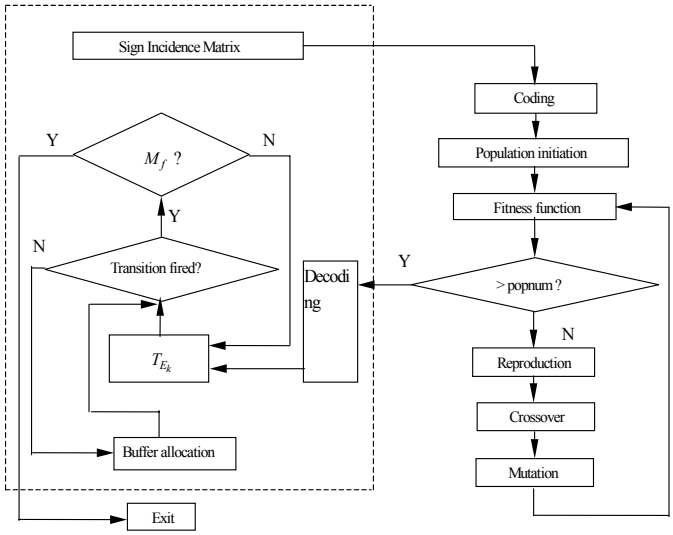

Fig.3. Algorithm

III. STRUCTURE OF THE DEADLOCK SOLUTION

Deadlock is considered in the scheduling and control parts. The whole control structure is shown in Fig.2. Jobs' information is put into Scheduler, an initial schedule will be generated. Then it will be send to database. AGV scheduling is not included into Scheduler, it is considered in the controller. This is the reason why the deadlock avoidance exists in controller. The controller gets the schedule from database and uses it to drive the plant. The system running information will be mapped to the Petri net model. The Petri net supervisor is in charge of supervising the system running. Because deadlock avoidance can't guarantee that deadlock can't occur, deadlock detection and deadlock recovery are needed. If deadlock occurs, the supervisor can detect and send command to the controller to execute the deadlock recovery. The deadlock prevention and deadlock avoidance algorithm can be simulated and verified by model checking tools: Spin.

IV. DEADLOCK FREE SCHEDULER

In Scheduler, deadlock free scheduling algorithm is applied to prevent from a deadlock while the system performance is guaranteed. Based on the FMS Petri Net model, this algorithm takes advantage of GA effective search to conduct scheduling. Such scheduling is deadlock-free, which is guaranteed by the embedded Petri Net checking procedure and reachability analysis.

Through the scheduler, the scheduling result including the buffer allocation is taken into account. This scheduling can solve three problems:

1. the deadlock free scheduling with no buffer can be gotten[8];
2. the deadlock free scheduling with buffers can be gotten[12];

3. the optimal deadlock free scheduling with infinite-capacity buffers can be gotten[12].

With such scheduler, some useful scheduling can be got as reference for users. Users can select from the scheduling pool according to their need.

Deadlock-free scheduling algorithm

The initial (or a considered current state) information of jobs can be expressed in the timed Petri net via the state of token distribution. From that, a set of initial schedules can be generated by using GA. Each schedule may correspond to a chromosome code string. After a serial of genetic operations and the decoding of chromosome, the transition firing set to the timed Petri net can be obtained.

According to this transition sequence, token player is used to check whether the deadlock status would occur in the system. Moreover, the system performance under implementation of different schedules can be evaluated.

To make the reachability graph develop normally, there are several ways to avoid the deadlock occurring in the evolution of token player.

(1) Delete the tested un-satisfying schedule and choose a new one from the chromosome code-pool as a new candidate [8].
(2) Add a new buffer to contain the processed part (which has not finished all of its operations) temporally and resolve the conflict of resource (buffer space) competition.
(3) Adjust the parts' input time and sojourn time (staying in different kinds of buffer) to avoid resource competition and deadlock.

All these three methods can be programmed as the sub-modules including in the main of GA scheduling program. They can be called, switched or branched in accordance with system constraints or user's commands.

Since the buffer allocation can be easily included into the resource set for scheduling if necessary, therefore, deadlock can be avoided and the final marking becomes reachable definitely. Suppose during the whole job-scheduling process, the number of allocated buffer can reach to an upper number *Bm* to avoid the deadlock/conflict and achieve a short makespan in the system, *Bm* becomes an important index for the execution of production batch.

If the shop keeps a limited number of buffers that is less than the maximum buffer capacity *Bm* needed by implementing the above ideal schedule, then as the embedded execution of task scheduling reaches some definite step, a job that finishes processing on one machine (not finish all its operations yet) is not able to move to an empty buffer/machine to release that machine resource. In that case, a temporary blocking or a deadlock appears in the shop. To deal with such situation (caused by limited buffer), the following procedures can be taken:

(i) Stop inputting new job from load station to reduce space occupation of work in process (WIP) in the shop.
(ii) Wait for other jobs to finish their processings and release a buffer to resolve the blocking state of the shop. As soon as the blocking is resolved, new job can be input again.

If all the jobs occupying the respective machines have finished their processing and no buffer has been released, the tested schedule has fallen into deadlock and can be deleted. One may choose a new one from the chromosome pool and try again.

Buffer will be allocated to the corresponding job when there is deadlock or conflict during the searching procedure of Petri net. The detail of the algorithm is shown in Figure 3.

1) Deadlock prevention

The initial scheduling result can be verified with deadlock prevention policy [10] in model checking tool.

V. THE SYSTEM DATABASE

The initial scheduling generated by scheduler is written into database, and stored in the database for the controller use. Controller will get the scheduling information according to the its algorithm. The system database is relational database.

VI. CONTROLLER

Banker's algorithm is applied in the controller. Such controller can prevent the system from entering into deadlock. The main function of the controller is as follows [11]:

1. Controlling the system running to avoid deadlock.
2. Recovering the system if deadlock occurs.
3. Simulating the behavior of the controller.
4. Verifying the properties of the controller.

Deadlock avoidance algorithm[11]

It is an improvement and modification of the Banker's algorithm. The outline of the algorithm is shown as following:

Every operation of a job is taken as a process. The relationship between job and resource is changed to the relationship between operation and resource. Check every active process to see if *Available resource + Allocation resource* $\geq$ *operations resource Claim*. Dynamic change of the *operation resource Claim* is taken into account. If an operation of a job finishes, it will release the resource that it occupies, and go on to the job's next operation. The operation is taken as a process here. The process can finish firstly, and releases the resource it occupies. Thus, the resource can be used by another process. That is, completing the operations of a job can release resources orderly. So, the resource releases don't need to wait until the job finishes. In this case, the *operation resource Claim* is designed to change with the job progressing. The definition of *Allocation resource* and *Available resource* are not changed. The *operation resource Claim* varies with the information flowing in channels.

The route selection of Jobs is involved in the modified

algorithm. When one route is not satisfied with the condition of the Banker's algorithm, another route is selected. If this route is still not satisfied with the condition of the Banker's algorithm, then the next one, or waiting.

Deadlock detection algorithm:

The empty siphon is directly related to deadlock in Petri net. The Petri net supervisor is responsible for checking empty siphon in the system Petri net[3]. If the supervisor detect empty siphon (deadlock), it will send command to the controller, then the controller will execute the deadlock recovery algorithm. The command includes the jobs information that fall into deadlock.

Deadlock recovery algorithm:

According to the command sent from supervisor, the controller will select one or more victims from the Jobs that enter into deadlock, and put it into buffer. Then, the system will go on running.

Simulation and Verification for the FMS Controller

The simulation is carried out on SPIN. SPIN is a tool for analyzing models expressed in the modeling language PROMELA [9]. PROMELA was chosen for the study because of its focus on the interaction between processes. It is loosely based on Hoare's language of CSP and on Dijkstra's guarded command. Given a PROMELA model, SPIN can perform simulations or exhaustive verifications of the system state space, during which it checks for the absence of deadlocks and for un-executable code. It can also verify linear time temporal constraints. Exhaustive verification can show conclusively whether a model contains errors. The model simulation tool provided in SPIN allows users to interactively simulate execution of PROMELA models. This tool is invaluable in the initial development and refinement of the model for the controller. The controller's verification frame is shown in Fig.4. The simulation result of a FMS is shown as Fig.5.

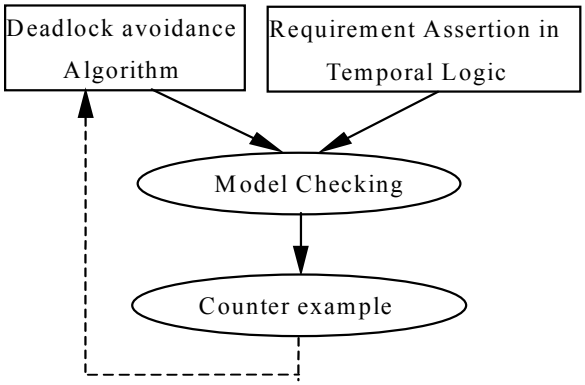

Fig.4.Controller verification frame

In the Spin Message Sequence Chart (Fig.5), there are eight vertical lines that represent the eight processes in the system (listing from the left to the right): Job1, Job2, buffer, AGV, M1, M2, M3, and init process (which is used to initialize the M1, M2, M3 process). The arrows connecting the processes are channels. Boxes correspond to steps in the simulation. Communication between Job1, Job2, buffer, AGV, M1, M2, M3 is executed through channels. Job1 process sends J1Rinfo (Job1 routing information) through AGV process to M1 process (corresponding to process M:4 in the graph; M2 process to M:5 process; M3 process to M:6 process), M1 process finishes its operation, sends the J1Rinfo to AGV process, then to M2 process. Finally, M2 process to AGV, then to M3 process. Thus, all operations of Job1 are finished and M3 Process sends the final J1Rinfo through AGV to Job1 process. The same to the progressing of J2Rinfo. If the requested resource is busy, the system will send the JRinfo to buffer process. The graphical output only highlights the interaction between processes.

VII. CONCLUSION

A complete deadlock resolution is presented in this paper. The deadlock is resolved from views of scheduling and control. In this FMS structure, the control algorithm can be simulated and verified. The solution of deadlock is considered all sided. This is also the distinct characteristic of this work. This work has been adopted by the project of the virtual workshop prototype design in Shanghai Jiaotong Univerisity.

REFERENCES

[1] Banaszak, Z.A.; Krogh, B.H. Dec. 1990. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. IEEE Transactions on Robotics and Automation, Volume: 6 Issue: 6, 724 –734

[2] Barkaoui, K.; Ben Abdallah, I. 1995. A deadlock prevention method for a class of FMS. Systems, Man and Cybernetics, 1995. IEEE International Conference on Intelligent Systems for the 21st Century, Volume: 5, 4119 -4124 vol.5

[3] Ezpeleta, J.; Colom, J.M.; Martinez, J. April 1995. A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Transactions on Robotics and Automation, Volume: 11 Issue: 2, 173 –184

[4] Gen, M.; Tsujimura, Y.; Kubota, E. 1994. Solving job-shop scheduling problems by genetic algorithm. Systems, Man, and Cybernetics, 1994. IEEE International Conference on Humans, Information and Technology, Volume: 2, 1994, 1577 –1582, 2

[5] Naiqi Wu ; MengChu Zhou. Oct 2001. Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems. IEEE Transactions on Robotics and Automation, Volume: 17 No.5, Page(s): 658 –669

[6] Song, Y.; Hughes, J.G. 1999. A genetic algorithm with a machine order-based representation scheme for a class of job shop scheduling problem. Proceedings of the

American Control Conference, 1999. Volume: 2 , 1999. Page(s): 895 -899 vol.2

[7] Viswanadham, N.; Narahari, Y.; Johnson, T.L. 1990. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. IEEE Transactions on Robotics and Automation, Volume: 6 Issue: 6, Dec. 1990 Page(s): 713 –723.

[8] Xu Gang; Zhiming Wu. 2002. Deadlock-free scheduling method using petri net model analysis and GA search. Proceedings of the 2002 International Conference on Control Applications, 2002. Volume: 2 , 2002. Page(s): 1153 –1158.

[9] Holzmann, G.J. 1997. The model checker SPIN. IEEE Transactions on Software Engineering, Volume: 23 Issue: 5 , May 1997 279–295

[10] Xu Gang; Zhiming Wu. 2003. Deadlock Prevention for Flexible Manufacturing System. Proceedings of the 2003 American Control Conference.

[11] Xu Gang; Zhiming Wu. 2003. The Application and Verification of Banker's Algorithm for Deadlock Avoidance in Flexible Manufacturing System with Spin. Proceedings of the 2003 IEEE International Conference on Robotics and Automation.(ICRA 2003), 2003,2165-2170.

[12] Xu Gang; Zhiming Wu. 2003. DEADLOCK-FREE SCHEDULING STRATEGY FOR AUTOMATED PRODUCTION CELL. Proceedings of the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2, 20-24, 850-855.

[13] Zhou,M.C, Deadlock Avoidance Methods for a Distributed Robotic System: Petri Net Modeling and Analysis". Journal of Robotic Systems,12(3),177-187.

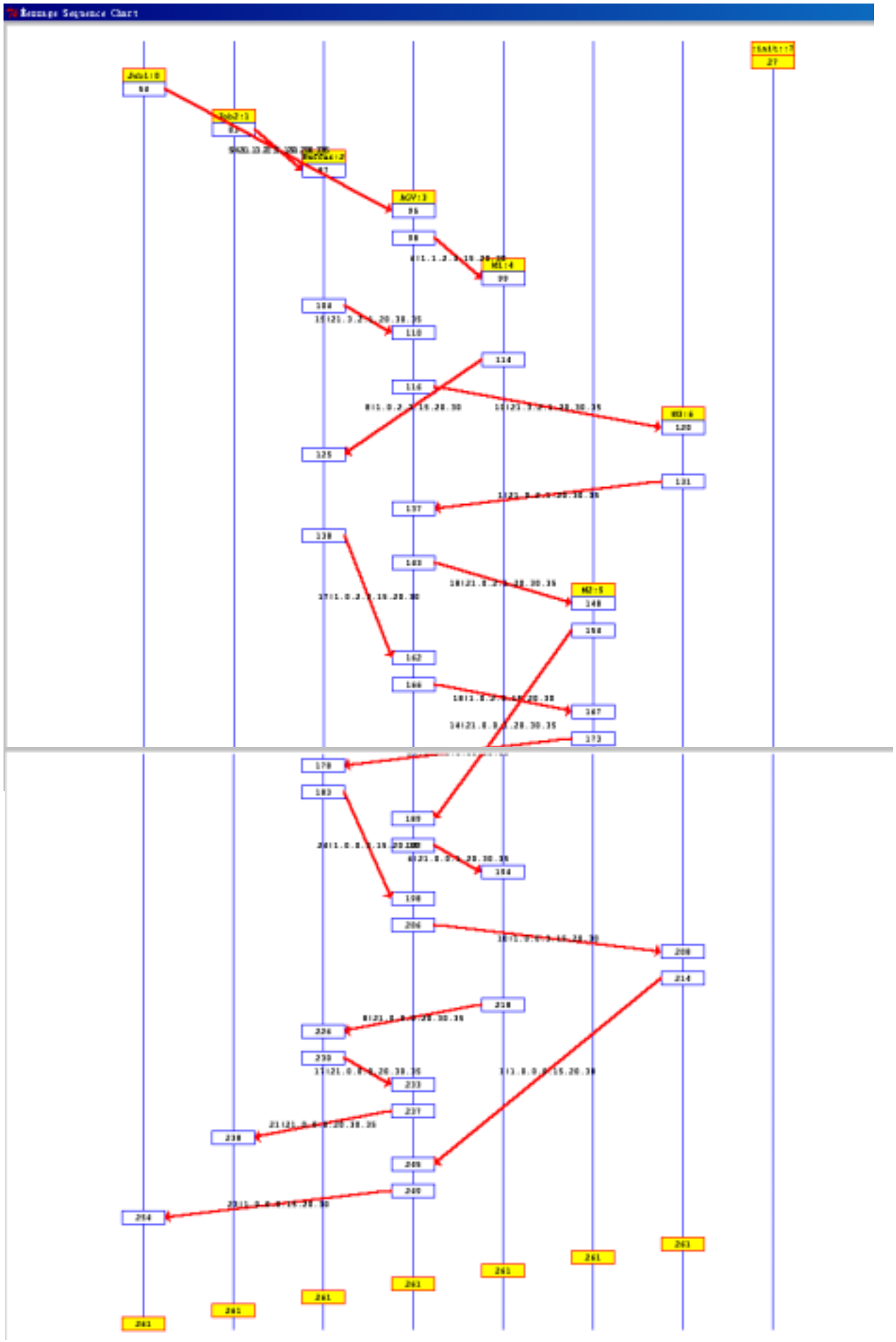

Fig.5. Simulation result of the FMS