1

Introduction

This chapter introduces the definitions of resource allocation systems (RASs) and their deadlock resolution and system scheduling. Different approaches to control and schedule RASs based on Petri nets are briefly reviewed. In addition, both the pros and cons of these approaches are introduced to show the development of deadlock resolution and system scheduling methods for RASs based on Petri nets.

1.1 Resource Allocation Systems

Resource scarcity is a common scenario in many discrete event dynamic systems (DEDSs). In such systems, available resources (such as machines, robots, drives, and programs) have to be shared among concurrently running processes (such as parts, vehicles, and data) and they must compete in order to be granted their allocation and achieve some system objectives such as maximizing makespan and minimizing tardiness. These systems are named resource allocation systems (RASs). This chapter briefly introduces different deadlock resolution and scheduling methodologies based on Petri nets, which are in the view of resource allocation and have successfully been applied to many RASs, such as flexible manufacturing systems (FMSs) (Li *et al.*, 2008*b*), project management systems (Kumar & Ganesh, 1998), and multithread software engineering systems (López-Grao & Colom, 2012).

According to Reveliotis *et al.* (1997), an RAS usually contains a set of resource types $R_i, i = 1, \ldots, |R|$, and a set of jobs $P_j, j = 1, \ldots, |P|$. Each resource type R_i is further characterized by its capacity that is a finite positive integer indicating at most how many units of R_i are in the system. A job P_j often contains a set of tasks or job stages $P_{jk}, k = 1, \ldots, |P_j|$, representing the operations to be processed to finish the job. The operational assumptions on job routing structures and resource requirements give rise to complex behavioral patterns, leading to different control and scheduling approaches. In terms of the resource requirements for jobs, RASs

Supervisory Control and Scheduling of Resource Allocation Systems: Reachability Graph Perspective,
First Edition. Bo Huang and MengChu Zhou.

can be categorized into the following four classes: (i) single-unit RASs in which each operation only requires one resource from a single resource type; (ii) single-type RASs in which each operation requires an arbitrary number of resources from the same resource type; (iii) conjunctive (AND) RASs in which each operation requires an arbitrary number of resources from different resource types; and (iv) disjunctive/conjunctive (AND/OR) RASs in which an operation may be associated with a set of conjunctive requests for different resource types. Among them, the AND/OR RASs are the most generalized ones.

In an RAS, there are often a limited number of shared resources to be allocated. The competition for such shared resources by different operations may cause deadlocks where two or more operations are each indefinitely waiting for the other to release their acquired resources. Deadlocks can lead to unnecessary economic cost and even catastrophic results in practical systems. Hence, they are a highly undesirable situation.

For a deadlock in RASs to occur, four conditions must be satisfied (Coffman *et al.*, 1971):

1) Mutual exclusion: when operations claim exclusive control of the resources they need;
2) Hold and wait: when operations hold resources already allocated to them while waiting for additional resources.
3) No preemption: when a resource can only be voluntarily released by the operation holding it.
4) Circular wait: when a circular chain of operations exists, where each operation holds one or more resources that are being requested by the next operation in the chain.

If a deadlock occurs, the above four conditions are met. To handle deadlocks in RASs, most approaches work by preventing one of the four conditions from being satisfied. In general, there are four deadlock handling approaches: deadlock ignoring, deadlock detection and recovery, deadlock avoidance, and deadlock prevention.

Deadlock ignoring, which is also known as an ostrich algorithm, assumes that deadlocks are exceedingly rare in the system and the influence incurred by a deadlock is tolerable. In this case, deadlock ignoring is acceptable from a technical and economical point of view. For example, in UNIX, if the process table is full and processes still try to fork some subprocesses, a deadlock can occur. However, it happens rarely and the procedures to prevent it are cumbersome. Thus, it is often ignored.

Deadlock detection and recovery (Reveliotis, 2000; Wysk *et al.*, 1994) uses a monitor to detect and recover from deadlocks. When a deadlock occurs, the monitor

detects it and then actions are taken to either terminate operations in the deadlock or release some resources held by operations to recover the system from the deadlock. This approach requires the periodic detection of deadlocks, which uses a large amount of data and may become complex if several kinds of shared resources are considered. Its efficiency depends on the response time of the implemented algorithms for deadlock detection and recovery. Thus, it is only applied to systems where detection and recovery strategies are not expensive.

Deadlock avoidance (Ezpeleta & Recalde, 2004; Fanti *et al.*, 2006; Hsieh, 2004; Lawley, 2000; Luo *et al.*, 2019; Reveliotis, 2007; Wu & Zhou, 2005; Xing *et al.*, 2009) is a dynamic and online approach that works in a real-time manner and bases the decision on information about the resource allocation status. In deadlock avoidance, both deadlocks and bad states (which are the states that inevitably lead to deadlocks) are avoided by using look-ahead procedures. The approach usually has high resource utilization and throughput since it tries to permit more states in a system, but sometimes does not eliminate all deadlocks. If it cannot avoid all deadlocks, some recovery strategies are required (Wysk *et al.*, 1994).

Deadlock prevention (Abdallah *et al.*, 2002; Chen *et al.*, 2016; Ezpeleta *et al.*, 1995; Fanti & Zhou, 2004; Feng *et al.*, 2020; Li & Zhao, 2008; Luo *et al.*, 2009; Tricas *et al.*, 2000; Zhou & DiCesare, 1991) works by preventing at least one of the four conditions necessary for the occurrence of a deadlock at any point of the RAS dynamic. It is an off-line computational approach which controls the requirements for resources to ensure that deadlocks never occur. When compared with deadlock avoidance, this approach tends to be more conservative and reduces resource utilization and system productivity. However, it has the advantages of safety and stability. In addition, its controller implementation does not need the knowledge of system states, which leads to a simple control law.

For deadlock resolution of RASs, there are mainly three modeling tools: digraphs, automata, and Petri nets. Each tool has its advantages and has greatly improved the researchers' insight into deadlock phenomena of RASs. Digraphs or graph theory is a simple and an intuitive tool to describe the interactions between operations and resources in RASs. In a digraph, deadlocks are usually related to the circuits in the graph. Thus, the occurrence of deadlocks can be detected by computing all the circuits of the graph and can then be eliminated by some deadlock resolution policies. In this field, some representative research groups are led by Wysk *et al.* (1991, 1994) and Fanti *et al.* (1997, 2002). Supervisory control theory (SCT), originated by Ramadge and Wonham (1987), uses formal language and finite state automata to provide a formal and generic framework of modeling and control of DEDSs. SCT has a profound influence on the supervisory control of DEDSs by using other formalisms such as Petri nets. Many effective and computationally efficient deadlock control policies are developed based on automata. Reveliotis, Ferreira, and Lawley are the representative researchers in

this field (Lawley & Mittenthal, 1999; Lawley *et al.*, 1998; Reveliotis & Ferreira, 1996). Particularly, Reveliotis *et al.* (1997) propose a significant deadlock avoidance strategy with polynomial complexity for a class of RASs based on finite state automata, which is then extended based on Petri nets by Park and Reveliotis (2001).

On the other hand, to meet the needs of market development, many practical RASs have to quickly respond to changes in the market by integrated planning and scheduling, which allocate limited shared resources to different operations and determine the sequences of operations so that the constraints of the system are met and performance criteria are optimized. RASs usually contain a great number of choices of resources and processing routes to allow high productivity. Thus, it imposes a challenging problem, that is, how to correctly allocate shared resources to different operations and effectively schedule the operations to achieve the highest efficiency. Such a scheduling problem is NP-hard and its computational time to obtain an optimal schedule grows exponentially with the problem size (Cao *et al.*, 2019; Kolisch & Drexl, 1997). To handle the RAS scheduling problem, there are generally two kinds of scheduling methods: exact methods and heuristic methods.

Exact methods are useful in obtaining an optimal schedule for small-scale RASs. For example, linear programming is a useful method that integrates various decision variables into one model to get an optimal solution for an RAS scheduling problem (Burkard & Hatzl, 2005; Qiao *et al.*, 2017). It focuses on a single objective such as maximizing the profit and minimizing the cost. However, practical problems do not always have one objective to achieve. For this reason, goal programming that exploits multiobjective optimization becomes attractive for real-world RAS scheduling (Azaiez & Al Sharif, 2005; Moro & Ramos, 1999). In general, such mathematical programming methods can ensure the optimality of the obtained solutions, but they have some limitations over complex systems since it is difficult for them to describe the constraints in complex RASs, such as the RASs with shared resources and alternative routes. In addition, exact methods are usually applied to small systems due to their heavy computational burden.

When exact methods are too slow to obtain optimal schedules for RASs, heuristic algorithms can be used to obtain suboptimal schedules in a reasonable time frame. Many researchers have combined heuristic dispatching rules (Dominic *et al.*, 2004; Li *et al.*, 2013) such as first come first serve, shortest processing time, and least working remaining time, with the evolution of systems' models to handle the RAS scheduling. Dispatching rules are often easy to use in a practical system, but they have limited information about the entire system and fail to schedule complex models well. To address this problem, metaheuristics are usually used. The representative metaheuristic strategies for RAS scheduling in the literature are simulated annealing (Jozefowska *et al.*, 2001; Yuan *et al.*, 2019a), genetic algorithm (Hou *et al.*, 2017; Pezzella *et al.*, 2008), tabu search (Nonobe

& Ibaraki, 2002; Zuo *et al.*, 2019), ant colony optimization (Feng, *et al.*, 2019; Rajendran & Ziegler, 2004), particle swarm optimization (Kang *et al.*, 2016; Sha & Hsu, 2006), and metaheuristic hybridization (Chen & Shahandashti, 2009; Yuan *et al.*, 2019b). The metaheuristic methods can often provide good solutions in a reasonable period of time, but it is rarely possible to tell how close the obtained solutions are to the optimal solution. Therefore, RAS scheduling is a typical combinatorial optimization problem for which a desirable scheduling method should include both easy formulation of the systems and quick computation of optimal or suboptimal solutions.

1.2 Supervisory Control and Scheduling with Petri Nets

Petri nets (Murata, 1989), also known as place/transition nets, are a graphical and mathematical formalism which is widely used to model and analyze RASs such as manufacturing systems, computer and communication networks, and transportation systems, because Petri nets can naturally model the systems' structural properties including sequential execution, concurrency, synchronization, and choice, and effectively analyze the systems' behavioral properties such as boundedness and deadlock.

To deal with the deadlock problem in RASs, there are mainly two kinds of Petri-net-based methods: structural analysis (Ezpeleta *et al.*, 1995; Huang *et al.*, 2001; Li & Zhou, 2006, 2008; Wu & Zhou, 2010; Xing *et al.*, 2011; Ye *et al.*, 2018) and reachability graph analysis (Basile *et al.*, 2013; Chen *et al.*, 2011; Ghaffari *et al.*, 2003; Li *et al.*, 2020; Uzam & Zhou, 2006). The former allows one to derive a control policy via special Petri net structures, e.g., resource-transition circuits and siphons, and the resulting control law is usually computationally efficient. However, its applications are often confined to certain classes of Petri nets and its obtained liveness-enforcing supervisors are usually restricted since some permissive behavior may be excluded in the controlled systems. The latter utilizes the Petri net reachability graph that fully reflects the behavior of the underlying system. Although its computational burden is always heavy, it is usually more general, which means that it can be applied to more kinds of Petri nets and often allows designers to obtain deadlock-free supervisors with maximally or highly permissive behavior.

Therefore, reachability graph analysis is an important deadlock resolution technique. In Ghaffari *et al.* (2003), the theory of regions is used to derive an optimal supervisor if such a supervisor exists. To improve its computational efficiency, Uzam and Zhou (2006) develop an iterative approach to deadlock control. They divide a reachability graph into a live zone (LZ) and a deadlock

zone (DZ). A marking in LZ is called a legal marking that can reach the initial marking, while DZ contains deadlock markings, livelock markings, and bad markings that inevitably lead to deadlocks and livelocks. First-met bad markings (FBMs) are those in DZ that are immediately reachable from some markings in LZ. At each iteration, to prevent an FBM from being reached, a control place and its arcs are designed by constructing a place invariant according to a well-established invariant-based control method (Yamalidou *et al.*, 1996). This process is iteratively carried out until all FBMs are forbidden. This method is easy to use. However, it does not consider the optimality of the controlled net and the structural minimality of the obtained supervisor.

In Piroddi *et al.* (2008, 2009), reachability graph analysis is combined with siphon analysis to obtain structure-small supervisors with highly permissive behavior. To obtain optimal and structurally minimal supervisors, Chen and Li (2011) propose a deadlock prevention method based on an integer linear program. A vector covering method is used to reduce the number of markings to be considered in the supervisor synthesis. Then, a liveness-enforcing supervisor is obtained by formulating and solving an integer linear program that forbids all FBMs, permits all legal markings, and minimizes the number of added places if such a supervisor exists. The obtained supervisor is optimal and structurally minimal in terms of places in the supervisor. However, it suffers from the state explosion problem produced by an exponential increase of the number of reachable states as system size increases and its computational burden is heavy since it needs to generate the whole reachability graph and often solve a large integer linear program.

On the other hand, the reachability graph of a Petri net represents the evolution of the state space of the underlying system. Thus, Petri nets are also an ideal and popular tool for the scheduling of RASs (Huang *et al.*, 2014; Luo *et al.*, 2015; Lee & DiCesare, 1994; Moro *et al.*, 2002*b*).

When the Petri net model of an RAS is constructed, a search algorithm can run on its reachability graph to find a schedule from a given initial marking to a given goal marking. However, due to the state explosion problem, generating the entire reachability graph is often difficult even for a moderate-sized Petri net model. To deal with this problem, Lee and DiCesare (1994) combine the modeling of Petri nets with an intelligent A* search to find an optimal or suboptimal firing sequence of transitions within the Petri net's reachability graph, only needing to explore a partial reachability graph with a heuristic function. In addition, if the used heuristic function is admissible, the obtained schedule is guaranteed to be optimal. Some admissible heuristic functions for the A* algorithm have been proposed in (Lee & Lee, 2010; Mejía, 2002; Moro *et al.*, 2002*b*; Xiong & Zhou, 1998). However, a problem observed is that, although the generation of the whole reachability graph is avoided, the number of the explored markings still grows exponentially with the

problem size and/or initial markings, making this method only applicable to small systems. To speed up the A* search, researchers have developed several improvements on it such as improved heuristic functions (Elmekkawy & Elmaraghy, 2003; Jeng & Peng, 1999), A* algorithms with a controlled or limited backtracking strategy (Mejía, 2002; Moro *et al.*, 2002*a*; Xiong & Zhou, 1998), hybrid A* algorithms to relax the evaluation scope (Lee & Lee, 2010; Moro *et al.*, 2002*b*), the deadlock-free dynamic window search (Luo *et al.*, 2015), and the iterative deepening A* with backtracking (Baruwa *et al.*, 2015). However, they are either only to optimally schedule small-scale systems or to accelerate the search process without controlling the quality of obtained solutions.

1.3 Summary

Results' quality and computational efficiency are two important factors in the supervisory control and system scheduling of RASs. A lot of work has been done by many researchers and engineers in the deadlock handling and system scheduling of RASs to obtain optimal or suboptimal solutions with smaller computational burden. This monograph focuses on both the solutions' quality and computational speed in the deadlock prevention and system scheduling of RASs by using Petri nets and their reachability graphs. Most of the new methods for the supervisory control and system scheduling with reachability graphs are included in this monograph, such as the theory of regions (Ghaffari *et al.*, 2003), the symbolic representations of Petri nets (Chen *et al.*, 2011; Pastor *et al.*, 2001), and some of our recent research results (Huang *et al.*, 2012*a*, 2014, 2015*b*,c, 2017, 2018*a*, 2019).

1.4 Bibliographical Notes

The survey papers and books regarding the deadlock handling and system scheduling of RASs can be found in the literature. The paper (Fanti & Zhou, 2004) surveys a lot of deadlock control approaches in the literature. Li *et al.* (2008*b*, 2012) review and compare many Petri-net-based deadlock control policies. The scheduling methods based on Petri nets and their applications can be found in Tuncel and Bayhan (2007). Rios and Chaimowicz (2010) gives a survey and classifications of the A* algorithms for the pathfinding. A recent survey about deadlock control policies for automated manufacturing systems with unreliable resources is given in (Du *et al.*, 2020). For the books dealing with supervisory control and heuristic search, please refer to Chen and Li (2013), Edelkamp and Schroedl (2012), and Hrúz and Zhou (2007).