
Production Systems and Information Engineering
Volume 5 (2009), pp. 3-18.

ADAPTIVE APPROACHES TO DISTRIBUTED RESOURCE ALLOCATION

Balázs Csanád Csáji
Computer and Automation Research Institute,
Hungarian Academy of Sciences; and
Department of Mathematical Engineering,
Catholic University of Louvain, Belgium
balazs.csaji@sztaki.hu

László Monostori
Computer and Automation Research Institute,
Hungarian Academy of Sciences; and
Faculty of Mechanical Engineering,
Budapest University of Technology and Economics
laszlo.monostori@sztaki.hu

[Received March 2009 and accepted May 2009]

Abstract. The problem of allocating scarce, reusable resources over time to interconnected tasks in uncertain and changing environments, in order to optimize a performance measure, arises in many real-world domains. The paper examines several recent distributed optimization approaches to this problem and compares their properties, such as the guarantees of finding (near-)optimal solutions, their robustness against disturbances or against imprecise, uncertain models, with a special emphasis on adaptive capabilities. The paper also presents a reinforcement learning based distributed resource control system and argues that this method represent one of the most promising approaches to handling resource allocation problems in the presence of uncertainties.

Keywords: resource allocation, adaptive algorithms, distributed optimization, stochastic processes, reinforcement learning

1. Introduction

Efficient allocation of reusable resources over time is an important problem in many real-world applications, such as manufacturing production control (e.g. production scheduling), fleet management (e.g. freight transportation), personnel management, scheduling of computer programs (e.g. in massively parallel GRID systems), managing a construction project or controlling a cellular mobile network. In general, they can be described as optimization problems

which include the assignment of a finite set of scarce reusable resources to interconnected tasks that have temporal extensions.

The resource allocation related combinatorial optimization problems, such as the job-shop scheduling problem or the traveling salesman problem, are known to be strongly NP-hard, moreover, they do not have any good polynomial time approximation algorithm, either [1]. These problems have a huge literature, e.g. [2], however, most classical approaches concentrate on static and deterministic variants and their scaling properties are often poor. In contrast, real-world problems are usually very large, the environment is uncertain and can even change dynamically. Therefore, complexity and uncertainty seriously limit the applicability of classical solution methods.

In the past decades a considerable amount of research has been done to enhance decision-making, such as resource allocation, and several new paradigms have appeared that handled the problem in large-scale, dynamic and uncertain environments. Distributed decision-making is often favorable [3], not only because it can speed up the computation, but also because it can result in more robust and flexible solutions. For example, if we take a multi-agent based point of view combined with a heterarchical architecture, it can present several advantages [4], such as self-configuration, scalability, fault tolerance, massive parallelism, reduced complexity, increased flexibility, reduced cost and potentially emergent behavior [5].

The structure of the paper is as follows. First, a general *Resource Allocation Problem* (RAP) is specified. Then, a few widespread distributed resource allocation approaches are considered, and their key properties are investigated, with a special emphasis on their adaptive capabilities. Finally, a *Reinforcement Learning* (RL) based distributed RA system is presented and its properties are demonstrated by experimental results. RL-based resource allocation is argued to be one of the most promising approaches among the systems presented.

2. Resource Allocation Framework

First, a deterministic resource allocation problem is considered: an instance of the problem can be characterized by an 8-tuple $\langle \mathcal{R}, \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{C}, d, e, i \rangle$. In detail the problem consists of a set of reusable *resources* $\mathcal{R}$ together with $\mathcal{S}$ that corresponds to the set of possible *resource states*. A set of allowed *operations* $\mathcal{O}$ is also given with a subset $\mathcal{T} \subseteq \mathcal{O}$, which denotes the *target operations* or *tasks*. $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{O}$ are supposed to be finite and they are pairwise disjoint. There can be *precedence constraints* between the tasks, which are represented by a partial ordering $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$. The *durations* of the operations depending on the state of the executing resource are defined by a *partial* function $d : \mathcal{S} \times \mathcal{O} \to \mathbb{N}$,

where $\mathbb{N}$ is the set of natural numbers, thus, we have a discrete-time model. Every operation can *affect* the state of the executing resource as well, that is described by $e : \mathcal{S} \times \mathcal{O} \to \mathcal{S}$, which is also a partial function. It is assumed that $dom(d) = dom(e)$, where $dom(\cdot)$ denotes the domain set of a function. Finally, the *initial states* of the available resources are given by $i : \mathcal{R} \to \mathcal{S}$.

The state of a resource can contain all relevant information about it, for example, its type and current setup (scheduling problems), its location and load (logistic problems) or its condition (maintenance and repair problems). Similarly, an operation can affect the state in many ways, e.g., it can change the setup of the resource, its location or its condition. The system must allocate each task (target operation) to a resource, however, there may be cases when first the state of a resource must be modified in order to be capable of executing a certain task (e.g. a transporter may first need to travel to its loading/source point, a machine may require repair or setup). In these cases non-task operations can be applied. They can modify the states of the resources without directly serving a demand (executing a task). It may be the case that during the resource allocation process a *non-task* operation is applied several times, but other *non-task* operations are completely avoided (for example because of their high cost). Nevertheless, finally all *tasks* must be completed.

A *solution* for a deterministic RAP is a partial function, the *resource allocator function*, $\varrho : \mathcal{R} \times \mathbb{N} \to \mathcal{O}$ that assigns the *starting times* of the operations to the resources. Note that the operations are supposed to be *non-preemptive* (they must not be interrupted).

A solution to a RAP is called *feasible* if the following properties are satisfied:

1. Each task is rendered to exactly one resource and start time:
$$\forall v \in \mathcal{T} : \exists\,! \langle r, t\rangle \in dom(\varrho) : v = \varrho(r, t)$$
2. All resources execute at most one operation at a time:
$$\neg\, \exists u, v \in \mathcal{O} : u = \varrho(r, t_1) \,\wedge\, v = \varrho(r, t_2) \,\wedge\, t_1 \leq t_2 < t_1 + d(s(r, t_1), u)$$
3. The precedence constraints of the tasks are kept:
$$\forall \langle u, v\rangle \in \mathcal{C} : [u = \varrho(r_1, t_1) \,\wedge\, v = \varrho(r_2, t_2)] \Rightarrow [t_1 + d(s(r_1, t_1), u) \leq t_2]$$
4. Every operation-to-resource assignment is valid:
$$\forall \langle r, t\rangle \in dom(\varrho) : \langle s(r, t), \varrho(r, t)\rangle \in dom(d)$$

where $s : \mathcal{R} \times \mathbb{N} \to \mathcal{S}$ describes the states of the resources at given time points,

$$s(r,t) = \begin{cases} i(r) & \text{if } t = 0 \\ s(r, t-1) & \text{if } \langle r, t\rangle \notin dom(\varrho) \\ e(s(r, t-1), \varrho(r, t)) & \text{otherwise.} \end{cases}$$

A RAP is called *correctly specified* if there exists at least one feasible solution. In what follows it is assumed that the problems are correctly specified. The set of all feasible solutions is denoted by $\mathbb{S}$. There is a performance (or cost) associated with each solution defined by a *performance measure* $\kappa : \mathbb{S} \to \mathbb{R}$ that often depends on the task completion times only. Typical performance measures that appear in practice include: maximum completion time or mean flow time. The aim of resource allocation is to compute a feasible solution with maximal performance (or minimal cost).

Note that the performance measure can assign penalties for violating *release* and *due* dates or even reflect the *priorities* of the tasks.

So far our model has been deterministic, now we turn to stochastic RAPs. The stochastic variant of the described general class of RAPs can be defined by randomizing functions d, e and i. Consequently, the operation durations become random, $d : \mathcal{S} \times \mathcal{O} \to \Delta(\mathbb{N})$, where $\Delta(\mathbb{N})$ is the space of probability distributions over $\mathbb{N}$. The effect of the operations is also uncertain, $e : \mathcal{S} \times \mathcal{O} \to \Delta(\mathcal{S})$ and the initial states of the resources can be stochastic as well, $i : \mathcal{R} \to \Delta(\mathcal{S})$. Note that the elements in the domain sets of functions d, e and i are probability distributions, we denote the corresponding random variables by D, E and I, respectively. We use the notation $X \sim f$ to indicate that random variable X has probability distribution f. Thus, $D(s, o) \sim d(s, o)$, $E(s, o) \sim e(s, o)$ and $I(r) \sim i(r)$ for all $s \in \mathcal{S}$, $o \in \mathcal{O}$ and $r \in \mathcal{R}$.

In stochastic RAPs the performance of a solution is also a random variable. Therefore, in order to compare the performance of different solutions we have to compare random variables. There are many ways in which this comparison can be made. For example, we can say that a random variable has stochastic dominance over another random variable 'almost surely', 'in likelihood ratio sense', 'stochastically', 'in the increasing convex sense' or 'in expectation'. In different applications various types of comparisons can be suitable, however, probably the most natural one is based upon the expected values of the random variables. The paper applies this kind of comparison.

Now, we classify the basic types of resource allocation techniques. In deterministic RAPs, there is no real difference between open- and closed-loop control. Thus, we can safely restrict ourself to open-loop methods. If the solution is aimed at generating the resource allocation off-line in advance, then it is called *predictive*. Thus, predictive solutions perform open-loop control and assume a deterministic environment. In stochastic resource allocation there are some data (e.g. the actual durations) that will only be available during the execution of the plan. According to the usage of this information, we identify two basic types of solution techniques. An open-loop solution that can deal with the uncertainties of the environment is called *proactive*. A proactive solution

allocates the operations to resources and defines the orders of the operations, but, because the durations are uncertain, it does not determine precise starting times. This kind of technique can be applied when only the durations of the operations are stochastic, but, the states of the resources are known in full (e.g. stochastic job-shop scheduling).

Finally, in the stochastic case a closed-loop solution to a RAP is called *reactive*. A reactive solution is allowed to make the decisions on-line, as the resource allocation process actually evolves and more information becomes available. Naturally, a reactive solution is not a simple ϱ function, but instead a resource allocation *policy* (a mapping from states to actions) which controls the process. Predictive and proactive RA has been investigated extensively over the past decades. The paper focuses on reactive resource allocation solutions only.

3. Distributed Resource Allocation

In this section a few widespread distributed resource allocation approaches will be considered and their key properties, such as the guarantees of finding an optimal (or a near optimal) solution, their robustness against different disturbances, such as breakdowns, or against imprecise, uncertain models, will be investigated, with a special emphasis on their adaptive capabilities.

A multi-agent system is a special distributed system with localized decision-making and, usually, localized storage. An agent is basically a self-directed (mostly software) entity with its own value system and a means to communicate with other such objects [4]. For a general survey on the application of multi-agent systems in manufacturing, see [6].

3.1. The PROSA Architecture

A basic agent-based architecture for manufacturing systems is PROSA [7]. The general idea underlying this approach is to consider both the resources (for example, machines and transporters) and the jobs (sets of interconnected tasks) as active entities. The standard architecture of the PROSA approach (see Figure 1) consists of three types of basic agents: order agents (internal logistics), product agents (process plans), and resource agents (resource handling). However, the PROSA architecture in itself is only a general framework and it does not offer any direct resource allocation solutions.

PROSA is a starting point for the design and development of multi-agent manufacturing control. Resource agents correspond to physical parts (production resources in the system, such as factories, shops, machines, furnaces, conveyors, pipelines, material storages, personnel, etc.), and contain an information processing part that controls the resource. Product agents hold the process

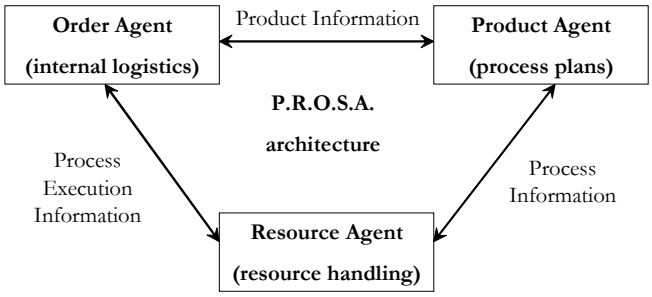

Figure 1. The PROSA reference architecture

and product knowledge to ensure the correct making of the product. They act as information server to other agents. Order agents represent a task or a job (an ordered set of tasks) in the manufacturing system. They are responsible for performing the assigned work correctly, effectively and on time.

3.2. Swarm Optimization

A great number of distributed optimization techniques were inspired by various biological systems [8], such as bird flocks, wolf packs, fish schools, termite hills or ant colonies. These approaches show up strongly robust and parallel behavior.

The ant colony optimization algorithm [9] is, in general, a randomized algorithm to solve *Shortest Path (SP)* problems in graphs. It can be shown that RAs can be formalized as special SP problems.

The PROSA architecture can also be extended by ant-colony-type optimization methods [10], in that case a new type of agent is introduced, called an ant. Agents of this type are mobile and they gather and distribute information in the manufacturing system. Their main assumption is that the agents are much faster than the ironware that they control, and that makes the system capable of prediction. Agents are faster and therefore can emulate the system's behaviour several times before the actual decision is taken.

The resource allocation in this system is made by local decisions. Each order agent sends out ants (mobile agents), which are moving downstream in a virtual manner. They gather information about the possible schedules from the resource agents and then return to the order agent with the information. The order agent chooses a schedule and then it sends ants to book the necessary resources. After that the order agent regularly sends booking ants to re-book the previously found best schedule, because if the booking is not refreshed, it

evaporates (like the pheromone in the analogy of food-foraging ants) after a while. From time to time the order agent sends ants to survey the possible new (and better) schedules. If they find a better solution, the order agent sends ants to book the resources that are needed for the new schedule and the old booking information will simply evaporate.

Swarm optimization methods are very robust, they can naturally adapt to environmental changes, since the agents continuously explore the current situation and the obsolete data simply evaporates if not refreshed regularly. However, these techniques often have the disadvantage that finding an optimal or even a relatively good solution cannot be easily guaranteed theoretically. For example, the ant-colony-based extension of PROSA faces almost exclusively the routing problem in resource allocation (how the tasks that belong to the same job should be processes through the machines) and it mostly ignores sequencing problems (the efficient ordering of the tasks that belong to different jobs). Therefore, the ant-colony-based extension of PROSA is very likely to be strongly sub-optimal, despite its very nice adaptive capabilities.

3.3. Negotiation-Based Approaches

There are multi-agent systems which use some kinds of negotiation or market-based mechanism in order to achieve efficient resource allocation [11]. In this case, the tasks or the jobs are associated with order agents, while the resources are controlled by resource agents, similarly to the PROSA architecture.

Market-based resource allocation is a recursive, iterative process with announce-bid-award cycles. During RA the tasks are announced to the agents that control the resources, and they can bid for the available jobs. A typical market-based system would work as follows: if a new job arrives at the system, a new order agent is created and associated with that job. An order agent or a group of cooperating order agents announces a sequence of operations and the resource agents can bid for that sequence. Only resource agents being able to do at least the first operation of that job are allowed to bid. Before an agent bids, it gathers information about the possible costs of making that sequence. If the sequence contains only one operation, the agent has all the information it needs, however, if the sequence contains other operations as well, which probably cannot be processed by the machine of the agent, it starts to search for subcontractors. It becomes a partial order agent and announces the remaining part of the sequence. The other resource agents, which can do the next operation, may bid for the remaining operation sequence. Consequently, a recursive announce-bid process begins. In the end, when all the possible costs of that (partial) job are known, the agent bids. If it made the highest bid (in a given time-frame), the agent (and its subcontractors) get the job.

A disadvantage is that during this mechanism, the jobs or tasks are, usually, announced one by one, which can lead to myopic behavior and, therefore, guaranteeing an optimal or even an approximately good solution is often very hard. Regarding adaptive behavior, market-based RA is often less robust than swarm-optimization methods, since, e.g. if a resource breaks down it is very likely that a large part of the negotiation process has to be restarted.

3.4. Problem Decomposition

The idea of divide-and-conquer is often applied in order to decrease computational complexity in combinatorial optimization problems. The main idea is to decompose the problem and solve the resulting sub-problems independently. In most cases calculating the sub-solutions can be done in a distributed way [12].

These approaches can be effectively applied in many cases, however, defining a decomposition which guarantees both efficient speedup together with the property that combining the optimal solutions of the sub-problems results in a global optimal solution is very demanding. Therefore, when we apply decomposition, we usually have to give up optimality and be satisfied with fast but sometimes far-from-optimal solutions. Moreover, it is hard to make these systems robust against disturbances. Tracking environmental changes can be often accomplished by the complete recalculation of the whole solution only.

3.5. Distributed Constraint-Satisfaction

Resource allocation problems (at least their deterministic variants) can be often formulated as constraint-satisfaction problems [13]. In this case, they aim at solving the problem formulated as follows:

$$\begin{aligned} \text{optimize} \quad & f(x_1, x_2, \ldots, x_n), \\ \text{subject to} \quad & g_j(x_1, x_2, \ldots, x_n) \leq b_j, \end{aligned}$$

where $x_i \in \Omega_i$, $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. Functions f and g_j are real-valued and so are $b_j \in \mathbb{R}$. Most resource allocation problems, for example, resource constrained project scheduling, can be even formulated as a linear programming problem, which formulation can be written as

$$\begin{aligned} \text{optimize} \quad & c^T x, \\ \text{subject to} \quad & Ax \leq b, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $x, c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and c^T denotes the transpose of c. Then, distributed variants of constrained optimization approaches can be used to

compute a solution. In that case, a close-to-optimal solution is often guaranteed, however, the computation time is usually large. The main problems with these approaches are that they cannot take uncertainties into account and, moreover, they are not robust against noises and disturbances.

4. Machine Learning and Resource Control

Machine learning techniques represent a promising new way to deal with resource allocation problems in complex, uncertain and changing environments. These problems can be often formulated as Markov decision processes and they can be solved by *Reinforcement Learning* (RL) algorithms [14, 15, 16, 17].

Now, we propose an RL-based adaptive sampler to compute an approximately optimal resource control policy in a distributed way. The sampling is done by iteratively simulating the resource control process. After each trial the policy is refined through recursive updates on the value function using the actual result of the simulation. Thus, from an abstract point of view, the optimization is accomplished through adaptive sampling. In order to achieve this, the RAP must be reformulated as a controlled Markov process.

4.1. Markov Decision Processes

Sequential decision making under uncertainty is often modelled by MDPs. This section contains the basic definitions and some preliminaries. By a (finite, discrete-time, stationary, fully observable) *Markov Decision Process* (MDP) we mean a stochastic system that can be characterized by an 8-tuple $\langle \mathbb{X}, \mathbb{T}, \mathbb{A}, \mathcal{A}, p, g, \alpha, \beta \rangle$, where the components are: $\mathbb{X}$ is a finite set of discrete *states*, $\mathbb{T} \subseteq \mathbb{X}$ is a set of *terminal states*, $\mathbb{A}$ is a finite set of control *actions*. $\mathcal{A} : \mathbb{X} \to \mathcal{P}(\mathbb{A})$ is the *availability function* that renders each state a set of actions available in that state where $\mathcal{P}$ denotes the power set. The *transition function* is given by $p : \mathbb{X} \times \mathbb{A} \to \Delta(\mathbb{X})$ where $\Delta(\mathbb{X})$ is the space of probability distributions over $\mathbb{X}$. Let us denote by $p(y|x, a)$ the probability of arrival at state y after executing action $a \in \mathcal{A}(x)$ in state x. The *immediate cost function* is defined by $g : \mathbb{X} \times \mathbb{A} \times \mathbb{X} \to \mathbb{R}$, where $g(x, a, y)$ is the cost of arrival at state y after taking action $a \in \mathcal{A}(x)$ in state x. We consider discounted MDPs and the *discount rate* is denoted by $\alpha \in [0, 1)$. Finally, $\beta \in \Delta(\mathbb{X})$ determines the *initial probability distribution* of the states in the stochastic system.

A (stationary, randomized, Markov) control *policy* is a function from states to probability distributions over actions, $\pi : \mathbb{X} \to \Delta(\mathbb{A})$. The initial probability distribution β, the transition probabilities p together with a control policy π completely determine the progress of the system in a stochastic sense, namely, it defines a homogeneous Markov chain on $\mathbb{X}$.

The *cost-to-go* function of a control policy is $Q^\pi : \mathbb{X} \times \mathbb{A} \to \mathbb{R}$, where $Q^\pi(x, a)$ gives the expected cumulative [discounted] costs when the system is in state x, it takes control action a and it follows policy π thereafter

$$Q^\pi(x, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \alpha^t G_t^\pi \;\middle|\; X_0 = x, A_0 = a \right], \tag{4.1}$$

where $G_t^\pi = g(X_t, A_t^\pi, X_{t+1})$, A_t^π is selected according to control policy π and the next state, X_{t+1}, has $p(X_t, A_t^\pi)$ probability distribution.

A policy $\pi_1 \leq \pi_2$ if and only if $\forall x \in \mathbb{X}, \forall a \in \mathbb{A} : Q^{\pi_1}(x, a) \leq Q^{\pi_2}(x, a)$. A policy is called *optimal* if it is better than or equal to all other control policies. The objective in MDPs is to compute a near-optimal policy.

There always exits at least one optimal (even stationary and deterministic) control policy. Although there may be many optimal policies, they all share the same unique optimal action-value function, denoted by Q^*. This function must satisfy a (Hamilton-Jacoby-) *Bellman type optimality equation* [18]:

$$Q^*(x, a) = \mathbb{E}\Big[g(x, a, Y) + \alpha \min_{B \in \mathcal{A}(Y)} Q^*(Y, B) \Big], \tag{4.2}$$

where Y is a random variable with $p(x, a)$ distribution.

From an action-value function it is straightforward to get a policy, for example, by selecting in each state in a greedy way an action producing minimal costs.

4.2. Adaptive Sampling

General RAPs with stochastic durations can be formulated as MDPs, as shown in [17]. Then, the challenge of finding a good policy can be accomplished by approximate Q-learning. In that case, the possible occurrences of the resource control process are iteratively simulated, starting from the initial stage of the resources. Each trial produces a sample trajectory that can be described as a sequence of state-action pairs. After each trial, the approximated values of the visited pairs are updated by the Q-learning rule.

The one-step Q-learning rule is $Q_{t+1} = TQ_t$, where

$$(TQ_t)(x, a) = (1 - \gamma_t(x, a))\, Q_t(x, a) + \gamma_t(x, a)(KQ_t)(x, a) \tag{4.3}$$

$$(KQ_t)(x, a) = g(x, a, Y) - Q_t(x, a) + \alpha \min_{b \in \mathcal{A}(Y)} Q_t(Y, b),$$

where Y and $g(x, a, Y)$ are random variables generated from the pair (x, a) by simulation, that is, according to probability distribution $p(x, a)$; the coefficients $\gamma_t(x, a)$ are called the *learning rate* and $\gamma_t(x, a) \neq 0$ only if (x, a) was visited during trial t. It is well-known [18] that if for all x and a:

$\sum_{t=1}^{\infty} \gamma_t(x,a) = \infty$ and $\sum_{t=1}^{\infty} \gamma_t^2(x,a) < \infty$, the Q-learning algorithm will converge with probability one to the optimal value function in the case of lookup table representation. Because the problem is acyclic, it is advised to apply *prioritized sweeping*, and perform the backups in an order opposite to which they appeared in during simulation, starting from a terminal state.

To balance between *exploration* and *exploitation*, and so to ensure the convergence of Q-learning, we can use the standard Boltzmann formula [18].

4.3. Cost-to-Go Approximation

In systems with large state spaces, the action-value function is usually approximated by a (typically parametric) function. Let us denote the space of action-value functions over $\mathbb{X} \times \mathbb{A}$ by $\mathbb{Q}(\mathbb{X} \times \mathbb{A})$. The method of fitted Q-learning arises when after each trial the action-value function is projected onto a suitable function space $\mathcal{F}$ with a possible error $\epsilon > 0$. The update rule becomes $Q_{t+1} = \Phi\, T\, Q_t$, where Φ denotes a projection operator to function space $\mathcal{F}$. In [17] support vector regression is suggested to effectively maintain the cost-to-go function. The value estimation then takes the form as follows

$$\tilde{Q}(x,a) = \sum_{i=1}^{l} (w_i^* - w_i) K(y_i, y) + b\,, \tag{4.4}$$

where K is an inner product kernel, $y = \phi(x,a)$ represents some peculiar features of x and a, w_i, w_i^* are the weights of the regression and b is a bias. As a kernel the usual choice is a Gaussian type function $K(y_1, y_2) = \exp(-\left\|y_1 - y_2\right\|^2 / \sigma^2)$ where $\sigma > 0$ is a user-defined parameter.

Partitioning the search space by decomposing the problem and applying limited-lookahead rollout algorithms in the initial stage can also speed up the computation of a near-optimal cost-to-go function considerably [17].

4.4. Distributed Sampling

In this section we investigate how the sampling presented can be distributed among several processors even if the value function is local to each processor.

If a common (global) storage is available to the processors, then it is straightforward to parallelize the sampling-based approximate cost-to-go function computation: each processor can search independently by making trials, however, they all share (read and write) the same global cost-to-go function. They update the value function estimations asynchronously.

A more complex situation arises when the memory is completely local to the processors, which is realistic if they are physically separated, e.g., in a GRID.

A way of dividing the computation of a good policy among several processors is possible when there is only one 'global' value function, however, it is stored in a distributed way. Each processor stores a part of the value function and it asks for estimations which it requires but does not have from the others. The applicability of this approach lies in the fact that the underlying MDP is acyclic and, thus, it can be effectively partitioned, for example, by starting the trials of each processor from a different starting state.

If the processors have their own completely local value functions, they can have widely different estimates on the optimal state-action values. In order to effectively compute a global value function, the processors should count how many times they updated the estimates of the different pairs. Finally, the values of the global Q-function can be combined from the individual estimates by a Boltzmann formula.

5. Experimental Results

In order to investigate our RL-based distributed resource control approach, numerical simulations were initiated and carried out.

First, the proposed approach was tested on Hurink's benchmark dataset [19]. It contains *Flexible Job-Shop (FJS)* scheduling problems with 6–30 jobs (30–225 tasks) and 5–15 resources. The performance measure is make-span, thus, the total completion time has to be minimized. These problems are 'hard', which means, e.g. that standard dispatching rules or heuristics perform poorly on them. This dataset consists of four subsets, each subset containing about 60 problems. The subsets (sdata, edata, rdata, vdata) differ in the ratio of resource interchangeability, shown in the 'flexib' column in Table 1. The other columns show the average error (avg err) and the standard deviation (std dev) after carrying out N iterations. The execution of 10 000 simulated trials (after on the average the system has achieved a solution with less than 5 % error) takes only a few seconds on an ordinary computer of today.

benchmark		**1000 iterations**		**5000 iterations**		**10 000 iterations**	
dataset	**flexib**	**avg err**	**std dev**	**avg err**	**std dev**	**avg err**	**std dev**
sdata	1.0	8.54 %	5.02 %	5.69 %	4.61 %	3.57 %	4.43 %
edata	1.2	12.37 %	8.26 %	8.03 %	6.12 %	5.26 %	4.92 %
rdata	2.0	16.14 %	7.98 %	11.41 %	7.37 %	7.14 %	5.38 %
vdata	5.0	10.18 %	5.91 %	7.73 %	4.73 %	3.49 %	3.56 %
average	2.3	11.81 %	6.79 %	8.21 %	5.70 %	4.86 %	4.57 %

Table 1. Performance (average error and deviation) on benchmark datasets

We initiated experiments on a simulated factory by modelling the structure of a real plant producing customized mass-products. We used randomly generated orders (jobs) with random due dates. The tasks and the process-plans of the jobs, however, covered real products. In this plant the machines require product-type dependent setup times, and another specialty of the plant is that, at some previously given time points, preemptions are allowed. The performance measure applied was to minimize the number of late jobs and an additional secondary performance measure was to minimize the total cumulative lateness, which can be applied to comparing two situations having the same number of late jobs. In Table 2 the convergence speed (average error and standard deviation) relative to the number of resources and tasks is demonstrated. The workload of the system was approximately 90 %. The results show that the suggested resource control algorithm can perform efficiently on large-scale problems, e.g. with 100 resources and 10 000 tasks.

configuration		**1000 iterations**		**5000 iterations**		**10 000 iterations**	
machs	**tasks**	**avg err**	**std dev**	**avg err**	**std dev**	**avg err**	**std dev**
6	30	4.01 %	2.24 %	3.03 %	1.92 %	2.12 %	1.85 %
16	140	4.26 %	2.32 %	3.28 %	2.12 %	2.45 %	1.98 %
25	280	7.05 %	2.55 %	4.14 %	2.16 %	3.61 %	2.06 %
30	560	7.56 %	3.56 %	5.96 %	2.47 %	4.57 %	2.12 %
50	2000	8.69 %	7.11 %	7.24 %	5.08 %	6.04 %	4.53 %
100	10000	15.07 %	11.89 %	10.31%	7.97 %	9.11 %	7.58 %

Table 2. Performance relative to the number of machines and tasks

We also investigated the parallelization of the method, namely, the speedup of the system relative to the number of processors. The average number of iterations was studied until the system could reach a solution with less than 5 % error on Hurink's dataset. We treated the average speed of a single processor as a unit (cf. with the data in Table 1). In Figure 2 the horizontal axis represents the number of processors applied, while the vertical axis shows the relative speedup achieved. We applied two kinds of parallelization: in the first case (dark gray bars), each processor could access a global value function. It means that all of the processors could read and write the same global action-value function, but otherwise, they searched independently. In that case the speedup was almost linear. In the second case (light gray bars), each processor had its own, completely local action-value function and, after the search was finished, these individual functions were combined. The experiments show that the computation of the RL-based resource control can be effectively distributed

even if there is not a commonly accessible action-value function available and each processor works locally with its own estimates.

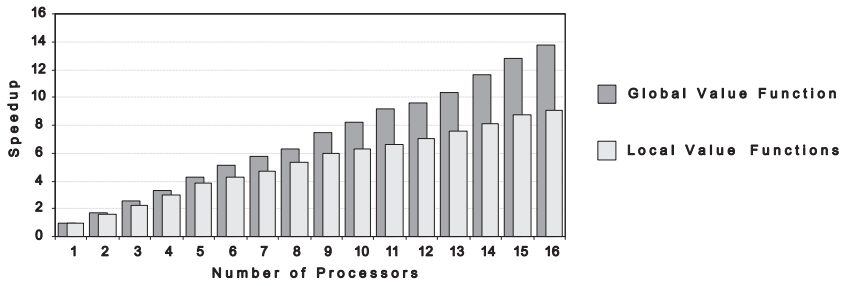

Figure 2. Distributed sampling: speedup relative to the number of processors

6. Concluding Remarks

Efficient allocation of scarce, reusable resources over time in uncertain and dynamic environments is an important problem that arises in many real world domains, such as production control. The paper examined some distributed RA approaches and presented an RL-based adaptive solution as well. The effectiveness of the latter approach was demonstrated by results of numerical simulation experiments on both benchmark and industry-related data.

There are several advantages why RL-based solutions are preferable to other kinds of distributed approaches described above. These favorable features are:

1. RL methods are *robust*, they essentially handle the problem under the presence of uncertainties, since they apply the theory of MDPs.
2. They can quickly *adapt* to unexpected changes in the environmental dynamics, such as breakdowns. This property can be explained by the Lipschitz type dependence of the optimal value function on the transition-probabilities and the immediate-cost function [20].
3. There are theoretical *guarantees* of finding optimal (or approximately optimal) solutions, at least asymptotically, in the limit.
4. Moreover, the actual convergence *speed* is usually high, especially in the case of applying distributed sampling or problem decomposition.
5. Additionally, the resulting distributed RL-based resource allocation *scales well* with the size of the problem. It can effectively handle large-scale problems without dramatic retrogression in the performance.
6. Finally, the proposed method constitutes an *any-time* solution, since the sampling can be stopped after any number of iterations.

Consequently, RL approaches have great potentials in dealing with real-world RAPs, since they can handle large-scale problems even in dynamically changing and uncertain environments. They seem to be one of the most promising approaches for distributed resource allocation in real-world domains.

Acknowledgements

The research presented was partially supported by the Hungarian Scientific Research Fund (OTKA) through the project "Production Structures as Complex Adaptive Systems". The paper also presented research results of the Belgian Programme on Interuniversity Attraction Poles, initiated by the Belgian Federal Science Policy Office, and a grant Action de Recherche Concertée (ARC) of the Communauté Française de Belgique. The scientific responsibility rests with its authors. Balázs Csanád Csáji acknowledges the postdoctoral fellowship of the Université catholique de Louvain. The authors express their thanks to Tamás Kis for his contribution to the tests on industrial data.

REFERENCES

[1] Williamson, D. P., A., H. L., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevastjanov, S. V., and Shmoys, D. B.: Short shop schedules. *Operations Research*, **45**, (1997), 288–294.

[2] Pinedo, M.: *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, 2002.

[3] Perkins, J. R., Humes, C., and Kumar, P. R.: Distributed scheduling of flexible manufacturing systems: Stability and performance. *IEEE Transactions on Robotics and Automation*, **10**, (1994), 133–141.

[4] Baker, A. D.: A survey of factory control algorithms that can be implemented in a multi-agent heterarchy: Dispatching, scheduling, and pull. *Journal of Manufacturing Systems*, **17**, (1998), 297–320.

[5] Ueda, K., Márkus, A., Monostori, L., Kals, H. J. J., and Arai, T.: Emergent Synthesis Methodologies for Manufacturing. *Annals of the CIRP – Manufacturing Technology*, **50**, (2001), 535–551.

[6] Monostori, L., Váncza, J., and Kumara, S. R. T.: Agent-based systems for manufacturing. *Annals of the CIRP*, **55**(2), (2006), 697–720.

[7] Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., and Peeters, P.: Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, **37**, (1998), 255–274.

[8] Kennedy, J. and Eberhart, R. C.: Particle swarm optimization. *IEEE International Conference on Neural Networks*, **4**, (1995), 1942–1948.

[9] Moyson, F. and Manderick, B.: The collective behaviour of ants: an example of self-organization in massive parallelism. In *Proceedings of AAAI Spring Symposium on Parallel Models of Intelligence*, Stanford, California, 1988.

[10] Hadeli, Valckenaers, P., Kollingbaum, M., and Van Brussel, H.: Multi-agent coordination and control using stigmergy. *Computers in Industry*, **53**, (2004), 75–96.

[11] Márkus, A., Kis, T., Váncza, J., and Monostori, L.: A market approach to holonic manufacturing. *Annals of the CIRP*, **45**, (1996), 433–436.

[12] Wu, T., Ye, N., and Zhang, D.: Comparison of distributed methods for resource allocation. *International Journal of Production Research*, **43**, (2005), 515–536.

[13] Modi, P. J., Hyuckchul, J., Tambe, M., Shen, W., and Kulkarni, S.: Dynamic distributed resource allocation: Distributed constraint satisfaction approach. In *Pre-proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages*, 2001, pp. 181–193.

[14] Zhang, W. and Dietterich, T.: A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995, pp. 1114–1120.

[15] Ueda, K., Hatono, I., Fujii, N., and Vaario, J.: Reinforcement learning approaches to biological manufacturing systems. *Annals of the CIRP – Manufacturing Technology*, **49**, (2000), 343–346.

[16] Aydin, M. E. and Öztemel, E.: Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, **33**, (2000), 169–178.

[17] Csáji, B. C. and Monostori, L.: Adaptive stochastic resource control: A machine learning approach. *Journal of Artificial Intelligence Research (JAIR)*, **32**, (2008), 453–486.

[18] Bertsekas, D. P.: *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edn., 2001.

[19] Hurink, E., Jurisch, B., and Thole, M.: Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations Research Spectrum*, **15**, (1994), 205–215.

[20] Csáji, B. C. and Monostori, L.: Value function based reinforcement learning in changing Markovian environments. *Journal of Machine Learning Research (JMLR)*, **9**, (2008), 1679–1709.