The Job Assignment Problem: A Study in Parallel and Distributed Machine Learning

Gerhard Weiß

Institut für Informatik, Technische Universität München D-80290 München, Germany
weissg@informatik.tu-muenchen.de

Abstract. This article describes a parallel and distributed machine learning approach to a basic variant of the job assignment problem. The approach is in the line of the multiagent learning paradigm as investigated in distributed artificial intelligence. The job assignment problem requires to solve the task of assigning a given set of jobs to a given set of executing nodes in such a way that the overall execution time is reduced, where the individual jobs may depend on each other and the individual nodes may differ from each other in their execution abilities. Experimental results are presented that illustrate this approach.

1 Introduction

The past years have witnessed a steadily growing interest in parallel and distributed information processing systems in artificial intelligence and computer science. This interest has led to new research and application activities in areas like parallel and distributed algorithms, concurrent programming, distributed database systems, and parallel and distributed hardware architectures. Three basic, interrelated reasons for this interest can be identified. First, the willingness and tendency in artificial intelligence and computer science to attack increasingly difficult problems and application domains which often require, for instance, to process very large amounts of data or data arising at different geographcial locations, and which are therefore often to difficult to be handled by more traditional, sequential and centralized systems. Second, the fact that these systems have the capacity to offer several useful properties like robustness, fault tolerance, scalability, and speed-up. Third, the fact that today the computer and network technology required for building such systems is available. A difficulty with parallel and distributed information processing systems is that they typically are rather complex and hard to specify in their dynamics and behavior. It is therefore broadly agreed that these systems should be able, at least to some extent, to self-improve their future performance, that is, to learn. Not surprisingly, today a broad range of work on learning in parallel and distributed information processing systems, or parallel and distributed machine learning for short, is available. Much of this work is centered around large-scale inductive learning (e.g., [1, 3, 7, 9]) and multiagent learning (e.g., [8, 10, 11]). What should be also mentioned here is the theory of team learning, which might be considered as an important contribution on the way toward a general theory

of parallel and distributed machine learning (e.g., [2, 5, 6]). The major property of this kind of learning is that the learning process itself is logically or geographically distributed over several components of the overall system and that these components conduct their learning activities in parallel. The field of parallel and distributed machine learning is of considerable importance, but also is rather young and still searching for its defining boundaries and shape. The work described in this article may be considered as an attempt to contribute to this search.

A problem being well suited for studying parallel and distributed learning is the job assignment problem (JAP). The basic variant of the JAP studied here requires to assign jobs to executing nodes such that the overall completion time is reduced, where there may be dependencies among the individual jobs and differencies in the execution abilities of the individual nodes. Obviously, the JAP inherently allows for parallelism and distributedness, simply because the jobs to be executed can be distributed over several nodes and because the nodes can execute different jobs in parallel. In addition to that, there are two further reasons why the JAP is an interesting and challenging subject of research not only from the point of view of machine learning. One reason is the complexity of this problem. The JAP is non-trivial and known to be a member of the class of NP-hard problems [4], and therefore in many cases it is even very difficult "to find a reasonable solution in a reasonable time". The other reason is that this problem is omnipresent in and highly relevant to many industrial application domains like product manufacturing and workflow organization. This is because the JAP constitutes the core of most scheduling tasks, and the effectiveness and efficiency of whole companies and organizations is therefore often considerably affected by the way in which they solve this problem in its concrete appearance.

The parallel and distributed learning approach to the JAP as it is introduced in this article follows the multiagent learning paradigm known from the field of distributed artificial intelligence. According to this approach, the nodes are considered as active entities or "agents" that in some sense can be said to be autonomous and intelligent, and the jobs are considered as passive entities or "resources" that in some way are used or handled by the agents. The individual agents are restricted in their abilities and, hence, have to interact somehow in order to improve their use and handling of the resources with respect to some predefined criteria. Learning in such a scenario can be interpreted as a search through the space of possible interaction schemes. Starting out from the concept of the estimates of the jobs' influence on the overall time required for completing all jobs, the described approach aims at appropriately adjusting these estimates by a parallel and distributed reinforcement learning scheme that only requires low-level communication and coordination among the individual nodes. This low-level characteristic makes this approach different from most other available multiagent learning approaches.

The article is structured as follows. The job assignment problem and three concrete instantiations of it are described in section 2. The learning approach is presented in detail in section 3. Experimental learning results for the three

Table 1:

job	<	time		
		N_1	N_2	N_3
1	2	40	80	100
2	8, 9	30	110	120
3	9	60	20	70
4	9	100	30	100
5	6	10	10	50
6	9	20	20	20
7	9, 10	70	50	20
8	–	40	20	80
9	–	30	90	80
10	–	60	50	20

Table 2:

job	<	time		
		N_1	N_2	N_3
1	2	10	50	50
2	3	10	40	70
3	4	10	100	50
4	5	10	60	90
5	6	10	100	50
6	–	10	50	80
7	–	40	80	80
8	–	100	120	60
9	–	60	30	70
10	–	20	90	80
11	–	100	100	30
12	–	60	20	50
13	–	10	40	70

Table 3:

job	<	time			
		N_1	N_2	N_3	N_4
1	–	70	10	10	10
2	–	60	10	10	10
3	5, 6	100	120	20	120
4	5, 6	110	120	80	20
5	7	60	90	10	100
6	8	50	140	100	10
7	–	100	150	20	70
8	–	90	100	140	10
9	–	10	10	10	60
10	–	140	100	20	120
11	–	70	10	80	90

instantiations of the JAP are shown in section 4. A brief summary and an outlook on future work is offered in section 5.

2 The Job-Assignment Problem (JAP)

The JAP as it is considered within the frame of the work described here can be formally described as follows. Let $J = \{J_1, \ldots, J_n\}$ be a set of jobs and $N = \{N_1, ..., N_m\}$ be a set of nodes, where each job can be executed by at least one of the nodes ($n, m \in \mathcal{N}$). The individual jobs may be ordered by a dependency relation, $<$, where $J_k < J_l$ means that J_k has to be completed before the execution of J_l can be started. J_k is called a predecessor of J_l, and J_l is called a successor of J_k. The nodes may differ from each other in as far as the time required for completing a job may be different for different nodes. The problem to be solved is to find an assignment of the jobs to the nodes such that the overall time required for completing all jobs contained in J is minimal. Because this problem is NP-hard, usually it is reformulated such that it is just required to find an almost optimal solution in polynomial time. The learning approach described in this article follows this reformulation, and aims at producing satisfying (and not necessarily optimal) solutions in reasonable time.

The tables 1 to 3 show three instantiations of the JAP, subsequently referred to as $I1$, $I2$, and $I3$, respectively. Consider the table 1 (the others are to be read analogously). There are 10 jobs and 3 nodes, and there are dependencies among some of the jobs. For instance, job J_1 has to be completed before job J_2 can be started, and the execution of job J_9 requires the completion of the jobs J_2, J_3, J_4, J_6, and J_7. Each job can be executed by each node, but there are differences in the time required by the nodes for executing the jobs. For instance, the nodes N_1, N_2, and N_3 need 40, 80, and 100 units of time, respectively, for completing the job J_1. As this table also shows, a node may require different time intervals

for completing different jobs. For instance, the node N_1 needs 40, 30, and 60 units of time for completing the jobs J_1, J_2, and J_3, respectively.

3 The Learning Approach

The basic idea underlying the multiagent learning approach described here is that each job is associated with an estimate of the job's influence on the overall completion time, and that these estimates are improved in the course of learning. As it is described in more detail below, this improvement as well as the execution of the jobs is done by the involved nodes in a parallel and distributed way. A high estimate indicates a significant impact on the overall completion time, and a job being associated with a high estimate therefore is identified as "critical" and should be completed as soon as possible. Learning proceeds in episodes, where an episode consists of the time intervall required for completing all jobs. The basic working steps realized during an episode can be conceptually described as follows:

until all jobs are completed do
(1) The idle nodes choose among the executable jobs, and this choice is done dependent on the nodes' execution times and the job estimates.
(2) The nodes execute their chosen jobs.
(3) If a node completes a job, then it adjusts the estimate of this job.

When an episode t is finished, the next episode $t+1$ starts and learning continues on the basis of the adjusted job estimates that are available at the end of episode t. This is iterated for a predefined, maximum number of episodes. The best solution found during these episodes is offered as the solution of the overall learning process. (A solution found in an episode need not necessarily be as good as the solution found in the preceding episode. Due to its statistical nature this approach does not guarantee a monotonic improvement of the solutions found in the course of learning.) The approach is parallel and distributed in as far as both job execution *(2)* and estimate adjustment *(3)* is done by different agents. A synchronization of the agents' activities occurs in step *(1)*. This also shows the potential advantages of this kind of learning over centralized learning approaches: it is more robust (e.g., failure of an individual node does not damage the overall learning process); it is more flexible (e.g., new nodes can be easily integrated in an ongoing learning process); and it is faster (because of inherent task and result sharing).

Many concrete forms of this conceptual description are possible. It was not the goal of the described work to exhaustively investigate all these forms. Instead, the work aimed at an improved understanding of the potential benefits and limitations of parallel and distributed machine learning in general, and therefore a concretization has been chosen that realizes this type of learning in an intuitive and relatively simple way and at the same time enables a conclusive and efficient experimental investigation. In the actual implementation, step *(1)* realizes a rank-based assignment of the executable jobs. This means that the job being

associated with the highest estimate is assigned first, the job having the second highest estimate is assigned next, and so forth. Moreover, if the assignment of a job is ambitious in the sense that there are several idle nodes capable of executing this job, then the node offering the shortest (job-specific) execution time is selected with highest probability. Formally, if $N[J_k]$ denotes the set of all idle nodes capable of executing a job J_k (at some time during an episode) and $T[i,k]$ denotes the time required by $N_i \in N[J_k]$ for completing J_k, then the probability that N_i executes J_k can be described by

$$\frac{T[i,k]}{\sum_{N_j \in N[J_k]} T[j,k]} \quad . \tag{1}$$

The actual implementation of step *(3)* offers two slightly different schemes, subsequently referred to as $A1$ and $A2$, for the adjustment of the job estimates. In the following, let E_k^t denote the estimate of job J_k at the beginning of episode t, let C_k^t denote the completion time of J_k in episode t, and let $\overline{C_k^t} = \frac{1}{t}\sum_{\tau=1}^{t} C_k^\tau$ denote the average completion time of J_k in the episodes 1 to t. According to the adjustment scheme $A1$, the estimates are updated immediately after job completion. Whenever a node finished a job J_k during an episode t, it modifies E_k^t according to

$$E_k^{t+1} = E_k^t + \alpha(C_k^t - \overline{C_k^t}) \quad , \tag{2}$$

where α is a factor called learning rate. The resulting estimate E_k^{t+1} is used for ranking in step *(1)* of episode $t+1$. The later (earlier) a job is completed, the higher (lower) is its estimate at the beginning of the next episode and, hence, the higher (lower) is the probability of an earlier execution of this job. According to the adjustment scheme $A2$, the job estimates are updated at the end of each epiode. In contrast to $A1$, this scheme explicitly takes into consideration that there may be dependencies among the jobs. Consider the situation at the end of episode t. Let $Pred(J_l)$ denote the set of all predecessors of J_l, let $Succ(J_l)$ denote the set of all successors of J_l, and let C^t denote the overall completion time in the episode t. For each $J_k \in Pred(J_l)$, the node which executed J_l pays a certain amount P_{lk}^t to the node which executed J_k during this episode. This amount is given by

$$P_{lk}^t = \begin{cases} (C_l^t - C_l^{t-1})(C^t - C^{t-1}) + P_l^t & \text{if } Succ(J_l) \neq \emptyset \\ (C_l^t - C_l^{t-1})(C^t - C^{t-1}) & \text{otherwise} \end{cases} \quad , \tag{3}$$

where P_l^t is the sum of all payments that the node which executed J_l received from the nodes which executed the successors of J_l at the end of the episode t, this is,

$$P_l^t = \sum_{J_i \in Succ(J_l)} P_{il}^t \quad . \tag{4}$$

After the node which executed a job J_k received all payments for this job from the nodes which executed this job's successors, it adjusts the estimate of J_k according to

$$E_k^{t+1} = E_k^t + \alpha(C_k^t - C_k^{t-1})(C^t - C^{t-1}) + \beta P_k^t \quad , \tag{5}$$

where α and β are factors called learning rates. The mechanism underlying this update scheme is illustrated in table 4. For instance, as this table shows, if a job J_k is finished later in episode t than in episode $t-1$ (i.e., $C_k^t - C_k^{t-1} > 0$) and the overall completion time increased (i.e., $C^t - C^{t-1} > 0$), then the estimate of J_k tends to increase (because the product $(C_k^t - C_k^{t-1})(C^t - C^{t-1})$ is positive, as expressed by the +). As a result, in this case it is likely that the job J_k will be executed earlier and, provided that there is a causal relationship between the completion time of J_k and the overall completion time, that an improved assignment will be generated in the next episode $t+1$. The effect of the

TABLE 4:

influence on E_k^t		$C^t - C^{t-1}$	
		> 0	< 0
$C_k^t - C_k^{t-1}$	> 0	+	−
	< 0	−	+

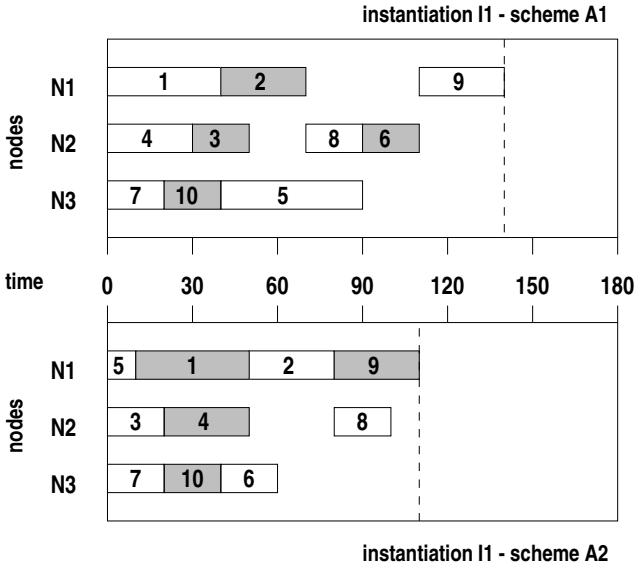
payments is that potential causal relationships between the completion time of the jobs and the overall completion time are propagated backwards through the job dependency network. All together, this adjustment scheme takes particularly care of "critical" dependency paths, that is, paths resulting in a late overall completion.

Learning according to this approach occurs in a parallel and distributed way. In particular, the estimates of different jobs may be adjusted concurrently, and all processors involved in job execution are also involved in the adjustment of the estimates. There are two major characteristics of this approach. First, it realizes a basic form of reinforcement learning. The only available learning feedback is the completion time of the individual jobs. This also means that there is no explict information available about how to gain an improved job assigment. Second, learning is just based on a low-level communication and coordination among the individual nodes. This also means that there is no time- or cost-consuming need for exchanging complex information or conducting complex negotiations in order to realize learning.

4 Experimental Results

The figures 1, 2, and 3 show for the JAP instantiations $I1$, $I2$, and $I3$, respectively, the best solutions learnt after 20 episodes by the adjustment schemes $A1$ and $A2$. In all experiments described here the learning rates α and β were set to one, and the parameters E_i^0, C_i^0, and C^0 where all initialized with zero. As figure 1 shows, the shortest overall completion times for $I1$ generated by $A1$ and $A2$ were 140 and 110, respectively; as figure 2 shows, the shortest overall completion times for $I2$ generated by $A1$ and $A2$ were 150 and 130, respectively; and as figure 3 shows, the shortest overall completion times for $I3$ generated by $A1$ and $A2$ were 130 and 160, respectively. As these figures also show, the scheme $A2$ produced slightly better results than the scheme $A1$ for the instantiations $I1$

FIGURE 1:



and $I3$. This observation has been also made in a number of other experiments not described here, and indicates that in general it is worth to explicitly take the dependencies among the jobs into consideration. This is not very surprising. The interesting point here is that good results could be also achieved if the dependencies are not explicitly taken into consideration - as the results for scheme A1 illustrate. This shows that the described learning approach is also of interest for domains in which the dependencies are unknown. In order to be able to evaluate these learning results, 20 random solutions for each of the three instantiations have been generated. The best random solutions found for $I1$, $I2$, and $I3$ had overall completion times of 230, 200, and 210, respectively. Obviously, both adjustment schemes clearly outperformed random search. Another measure of evaluation is given by the optimal overall completion times, which are 100 for $I1$, 130 for $I2$, and 100 for $I3$. It should be noted that the three instantiations under consideration were designed as test cases whose optimal solution are known – as mentioned above, the JAP is too complex to be optimally solved in general. A comparison with the optimal solutions shows that the optimum was found for $I2$ and was closely approached for $I1$ and $I3$. This qualitatively coincides with the comparative results gained in several other experiments. For reasons of completeness is should be mentioned that the optimal solution for $I1$ was found after 62 episodes (by $A1$) and for $I3$ after 41 episodes (by $A2$). It has to be stated, however, that the schemes $A1$ and $A2$ are not proven to always converge to the optimal solution, and therefore leave room for improvement.

FIGURE 2:

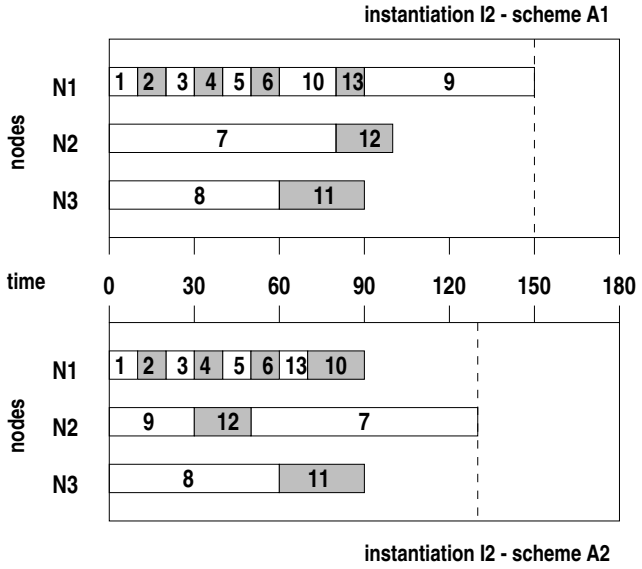
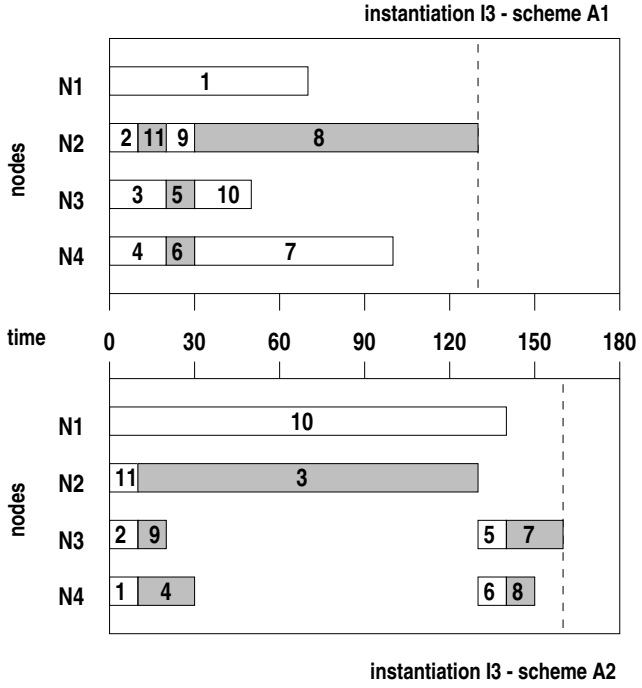


FIGURE 3:



5 Conclusion

The work described in this article applies parallel and distributed machine learning to a basic variant of the job-assignment problem. The learning approach

bases on the multiagent learning paradigm known from distributed artificial intelligence, and ascribes the jobs the passive role of resources and the nodes the active role of agents. The concept of the estimates of the jobs' influence on the overall completion time is introduced, and the job estimates are adjusted in a reinforcement learning style and without requiring intensive communication and coordination among the individual nodes. Three instantiations of the JAP are described and used for an experimental analyis. The available experiments show that very good learning results can be achieved in relatively short time intervals. Based on the experience and insights gained so far, the following two lines of continuing work are recommended:

- Further experiments with more complex JAP settings (e.g., larger number of nodes and/or jobs, variants of the JAP with varying numbers of nodes and/or jobs, and variants of the JAP with more sophisticated dependencies among the jobs and/or nodes). Ideally, such experiments will be conducted in context of concrete real-world applications like multi-processor scheduling in computer networks or industrial workflow optimization.
- Bottom-up extension of the learning approach towards distributed planning and lookahead mechanisms and explicit negotiation among the nodes.

These two lines should not be considered separately, but in close interaction. The results available so far indicate that it is worth to pursue these lines.

Parallel and distributed machine learning establishes a relatively young area of research and application, and there are many open issues that still have to be addressed in future work. Three of these issues considered as particularly essential are the following:

- applicability and limitations of traditional machine learning approaches in the context of parallel and distributed information processing systems;
- unique requirements for and principles of parallel and distributed machine learning;
- formal models of parallel and distributed machine learning.

The need for addressing these and related issues increases as parallelism and distributedness play an increasingly important role for computer-based information processing.

References

1. Chan, P.K., & Stolfo, S.J. (1995). A comparative evaluation of voting and meta-learning of partitioned data. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 90–98).
2. Daley, R.P., Pitt, L., Velauthapillai, M., Will, T. (1991). Relations between probabilistic and team one-shot learners. In *Proceedings of the Workshop on Computational Learning Theory* (pp. 228–239).
3. Davies, W., & Edwards, P. (1997). The communication of inductive inferences. In [10].

4. Garey, M.JR., & Johnson, D. (1979). *Computers and intractability*. New York: Freeman.
5. Jain, S., & Sharma, A. (1995). On aggregating teams of learning machines. *Theoretical Computer Science A*, 137(1), 85–108.
6. Pitt, L., & Smith, C. (1988). Probability and plurality for aggregations of learning machines. *Information and Computation*, 77, 77-92.
7. Provost, F.J., & Hennessy, D.N. (1995). Distributed machine learning: Scaling up with coarse grained parallelism. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology* (pp. 340–348).
8. Sen, S. (Ed.) (1996). *Adaptation, coevolution and learning in multiagent systems. Papers from the 1996 AAAI Symposium*. Technical Report SS-96-01. AAAI Press.
9. Sikora, R., & Shaw, M.J. (1991). *A distributed problem-solving approach to inductive learning*. Faculty Working Paper 91-0109. Department of Business Administration, University of Illinois at Urbana-Champaign.
10. Weiß, G. (Ed.) (1997). *Distributed artificial intelligence meets machine learning*. Lecture Notes in Artificial Intelligence, Vol. 1221. Springer-Verlag.
11. Weiß, G., & Sen, S. (Eds.) (1996). *Adaption and learning in multi-agent systems*. Lecture Notes in Artificial Intelligence, Vol. 1042. Springer-Verlag.

This article was processed using the LaTeX macro package with LLNCS style