J. Parallel Distrib. Comput. 75 (2015) 101–106

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# Dynamic task scheduling using a directed neural network

Binodini Tripathy[a], Smita Dash[b], Sasmita Kumari Padhy[c,*]

[a] KIIT University, Odisha, India
[b] SOA University Bhubaneswar, Odisha, India
[c] National Institute of Technology, Patna, India

## HIGHLIGHTS

- Development of the learning method for ANN.
- Development of a method for optimization of RBFNN.
- Use of DSO in task scheduling.
- Use of DSO trained ANN in task scheduling.
- Use of DSO trained RBFNN in task scheduling.

## ARTICLE INFO

*Article history:*
Received 7 June 2014
Received in revised form
23 September 2014
Accepted 26 September 2014
Available online 8 October 2014

*Keywords:*
Task scheduling
Directed search optimization
Neural network

## ABSTRACT

This article is based on the problem of work flow scheduling in grid environment of multi-processors. We, in this paper, introduce three novel approaches for the task scheduling problem using recently proposed Directed Search Optimization (DSO). In the first attempt, task scheduling is framed as an optimization problem and solved by DSO. Next, this paper makes use of DSO as a training algorithm to train (a) a three layer Artificial Neural Network (ANN) and then (b) Radial Basis Function Neural Networks (RBFNN). These DSO trained networks are used for task scheduling and interestingly yield better performance than contemporary algorithms as evidenced by simulation results.

## 1. Introduction

A computational grid is a coordinated phenomenon to share resources. This solves the problem in organization that is virtual, dynamic and multi-institutional. This coordinates resources that are not controlled centrally. The grid uses some standard protocols those are open and of general-purpose. The grid, in fact, is an interfacing tool for delivering important service quality. This virtualizes the resource, provides on-demand, and shares the resource among the organizations. This is consisting of a device to share the power of the computer and at the same time to store the data on the Internet [10,17]. There is a requirement for the management of the grid through an efficient scheduling approach.

Multiprocessor scheduling is an NP-hard problem [14,8,2]. The scheduling problem for tasks either dependent or independent is a well-studied discipline in the literature. In this article, we study the problem in an environment that is heterogeneous. These dynamic scheduling approaches are applicable to any larger set of real-time applications. These applications can be executed deterministically. Some of the conventional approaches provide global optimum, however longer time for execution and limited application for real-world problems are the drawbacks [15], whereas some other conventional approaches used may be deterministic and fast, but however fall into local optima [6].

This led to the research studies for the application of meta-heuristics, because efficacy or application of these algorithms is not limited for a specific problem. Available approaches for multiprocessor scheduling are broadly classified into heuristics and meta heuristics categories. The tasks maintain a queue in heuristic approaches. This queue is a priority queue and the processor processes the tasks on a first come first served basis, while in meta-heuristic approaches, they solve a class of computing problems by hybridizing user-provided procedures. In fact, these are heuristics by themselves, but in an efficient manner. Accordingly, Genetic Algorithms (GA) [14,19], Ant Colony Optimization (ACO) [5], Particle Swarm Optimization (PSO) [22,4,1,13,18], etc., are used for a better solution of the problem. But the results are constrained by efficiency. Hence, this paper proposes DSO [23] for task scheduling.

* Corresponding author.
*E-mail address:* chavisiba@rediffmail.com (S.K. Padhy).

http://dx.doi.org/10.1016/j.jpdc.2014.09.015
0743-7315/© 2014 Elsevier Inc. All rights reserved.

The art of using the artificial neural network (ANN) for task scheduling has been gaining momentum since last three decades. ANN trained with Back Propagation (ANN–BP) once again falls short of providing exact solution to the problem. Hence, this paper proposes DSO [23] as a training algorithm for ANN to be used in task scheduling.

Using neural networks has the limitations of large complexity and also fails because of over-fitting, local optima. On the other hand, RBFNNs, with only one hidden layer, have the ability to find global optima. In addition to less computational complexity, simulations performed in the literature reveal that the RBFNN produces superior performance as compared to other existing ANN-based approaches. Hence the works on task scheduling using RBFNN became an established and an active area of academic research and development [16,7,21,12].

However, there still are some difficulties with building RBFNNs. Main problems with RBFNNs are in determining the number of RBFs, number of cluster centers, etc. The conventional approaches consume longer time in determining the parameters using trial-and-error methods. Selecting the free parameters for the RBFs is also an issue for RBFNN. To get rid of the above-mentioned problems like trial-and-error steps and that of local optimal, Barreto et al. [3] used GA and Feng [9] used PSO to decide the centers of hidden neurons, spread and bias parameters by minimizing the Mean Square Error (MSE) of the desired outputs and actual outputs. In this paper we use DSO [23] for the training of ANN and RBFNN equalizers.

The paper is organized as follows: Section 2 discusses the problem of task scheduling. Sections 3–5 respectively discuss proposed approaches, DSO, DSO-trained ANN, and DSO-trained RBFNN. Performance of proposed approaches is evaluated through simulations explained in Section 6. Finally conclusion of the paper is outlined in Section 7.

## 2. The problem

This article considers the problem of assignment of task in the following manner. In the multiprocessor scheduling problem, the schedule is normally reflected as a task graph (TG). A simple task graph, used throughout this article, with communication and computation costs is shown in Fig. 1 and devised in a way used in [20]. The task graph $T = (N, S)$ is a set $N$ of $n$ nodes and a set $S$ of $s$ sides. Here, the nodes correspond to tasks derived from the applied task, and the sides correspond to constrained conditions among the tasks that are dependent. To be more specific, each side $s_{i,j} \in S$ between task $n_i$ and $n_j$ meaning that the objective output of task $n_i$ can transmit to task $n_j$ to make the next task $n_j$ to start execution. With reference to Fig. 1, the task with no predecessor is called as the entry task and the task with no successor is called the exit task.

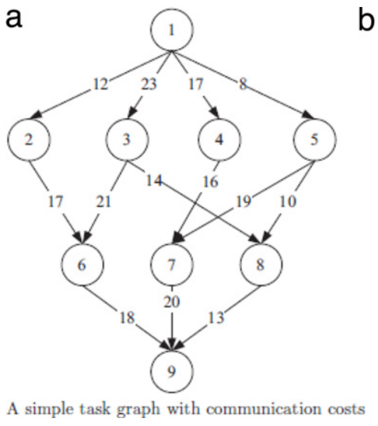
The weights $w_i$ attached to the tasks $n_i \in N$ are positive and termed as computation cost. However, the computing efficiency of the processors in heterogeneous environments is different from each other. The weights $w_{i,j}$ and $\hat{w}_i$ respectively are computation cost of the task $n_i$ on the processor $p_j$ and average computation cost. The computation cost of a task in this paper refers to the task execution time.

The nonnegative weight $c_{i,j}$ associated with side $s_{i,j} \in S$ corresponds to its communication cost between dependent tasks $n_i$ and $n_j$. If the tasks those are dependent are with different processors, then we need to compute the communication cost. Hence, actual communication cost is zero for the tasks with the same processor.

Consider a TG for $T$ tasks and with $P$ processors. For the task $n_i$, its earliest start time $T_{\text{start}}(n_i, p_j)$ on the processor $p_j$ is defined as:

$$T_{\text{start}}\left(n_i, p_j\right) = \max\left\{T_{\text{free}}\left(p_j\right), T_{\text{ready}}\left(n_i, p_j\right)\right\}. \quad (1)$$

a



b

| Task | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| 1 | 14 | 16 | 14 | 17 |
| 2 | 10 | 12 | 12 | 14 |
| 3 | 17 | 19 | 15 | 16 |
| 4 | 20 | 22 | 18 | 22 |
| 5 | 21 | 18 | 16 | 20 |
| 6 | 12 | 14 | 14 | 16 |
| 7 | 16 | 16 | 20 | 17 |
| 8 | 19 | 17 | 20 | 18 |
| 9 | 14 | 15 | 16 | 16 |

Computation costs

**Fig. 1.** An example of multiprocessor scheduling problem. (a) A simple task graph with communication costs and (b) computation costs [20].

Here, $T_{\text{free}}\left(p_j\right)$ is the time when processor $p_j$ is free for task $n_i$ to be executed. Most of the time, $T_{\text{free}}\left(p_j\right)$ refers to the completion of the last task by the processor $p_j$. In some cases task $n_i$ may be injected in a time when the processor $p_j$ is idle. In other words, when the length of an idle time gap is more than $w_i$, $T_{\text{free}}\left(p_j\right)$ may finish before the finish time of the last task. Here, $T_{\text{ready}}\left(n_i, p_j\right)$ represents the time of arrival of the entire data set at the processor $p_j$, and defined as:

$$T_{\text{ready}}\left(n_i, p_j\right) = \max_{n_k \in \text{pred}(n_i)}\left[T_{\text{finish}}\left(n_k\right) + c_{k,i}\right]. \quad (2)$$

Here, $T_{\text{finish}}(n_k)$ is actual finish time for the execution of the task $n_k$ and pred $(n_i)$ is the set of predecessors for the same task $n_i$.

In non-pre-emptive environments, $T_{\text{finish}}\left(n_i, p_j\right)$ is the fastest time to finish the task $n_i$ with processor $p_j$ as:

$$T_{\text{finish}}\left(n_i, p_j\right) = T_{\text{start}}\left(n_i, p_j\right) + w_{i,j}. \quad (3)$$

When task $n_i$ is fixed for scheduling with the processor $p_j$, the quickest start and finish time with the processor $p_j$ are also actual start and finish time for the task, respectively $\left(\text{i.e., } T_{\text{start}}(n_i) = T_{\text{start}}\left(n_i, p_j\right) \text{ and } T_{\text{finish}}(n_i) = T_{\text{finish}}\left(n_i, p_j\right)\right)$. Assuming that the starting time for the first task is time 0, the total length of schedule is termed as the *makespan* and is the actual largest finish time of the exit task, which is:

$$makespan = \max_i\left[T_{\text{finish}}\left(n_{\text{exit}}\right)\right]. \quad (4)$$

Hence, the objective of the problem is to process the task set with the processor set while minimizing the *makespan* having considered the constraints.

## 3. Task assignment using DSO [23]

This section first outlines basic definitions, parameter ranges and nomenclature used in Section 3.1, followed by algorithm steps of DSO as proposed by Dexuan Zou et al. [23] in Section 3.2.

### 3.1. Terminology

This sub-section outlines the terms those are better explained in Fig. 2.

- $j_g$: global best solution vector.
- $r$ is a random number in the region [0, 1]
- $x_i^j(k)$ and $x_i^j(k+1)$: value of $x_i^j$ in the $k$th iteration, and updated component (determined by $p_\alpha$) respectively.