

Scheduling Tasks in Real-Time Systems Using Evolutionary Strategies

Garrison W. Greenwood and Christian Lang
Dept. of Elec. & Computer Engineering
Western Michigan University
Kalamazoo, MI 49008

Steve Hurley
Dept. of Computing Mathematics
University of Wales at Cardiff
Cardiff, CF2 4YN, U.K.

Abstract *Finding feasible schedules for tasks running in hard, real-time distributed computing systems is generally NP-hard. This paper describes a heuristic algorithm using evolutionary strategies. Our results indicate the evolutionary strategies can find feasible schedules (assuming they exist) in very short periods of time.*

1 Introduction

Scheduling in real-time distributed systems requires that the tasks be assigned to computing resources and the order of execution be stipulated such that a) no precedence constraints are violated, and b) all temporal constraints are satisfied. A schedule is called *feasible* if both of these conditions are met. Finding feasible schedules in distributed systems is generally NP-hard which means one must resort to using heuristic techniques.

In this paper we present a scheduling algorithm for tasks in hard real-time distributed systems using *evolutionary strategies* (ES). Our previous work indicates that ES are capable of finding low schedule length task allocations in distributed systems in far less time than that required by other heuristic scheduling algorithms [1].

2 Problem Description

The distributed systems of interest consists of a number of homogeneous nodes that communicate over an interconnection network via message passing. We assume that communications are reliable and have predictable latencies.

Each task T_i is executed at regular intervals of period PD_i where any subtask of T_i executes once each period. Tasks have a defined precedence. (This can be represented as a digraph where vertices represent tasks and a directed arc from a task points to its successor.) Each task has a known execution time and deadline for completion. No task or subtask can be preempted once it has begun execution. In support of fault tol-

erance, certain critical tasks will be replicated. Each replicated task must be assigned to a unique node. Finally, the size of all messages transferred between a task and its successor is known.

3 Description of the ES

ES are based upon the principles of adaptive selection found in the natural world. Each generation (iteration of the ES algorithm) takes a population of individuals (potential solutions) and modifies the genetic material (problem parameters) to produce new offspring. Both the parents and the offspring are evaluated but only the highest fit individuals (better solutions) survive over multiple generations. *This means the ES is simultaneously investigating several regions of the search space which greatly decreases the amount of time required to locate a feasible schedule (assuming one exists).* ES have been successfully used to solve various types of optimization problems. The reader is referred to Michalewicz for an excellent discussion of the ES technique [2].

The particular genetic encoding for an individual is referred to as the *genotype*. New genotypes are created by special operations (e.g., mutation) which modifies the genetic material. Decoding this genetic material gives the observed characteristics of the individual which is referred to as the *phenotype*. In our case, the genotype consists of P integer lists which reflects the tasks allocated to the P processors in the distributed system. The left-to-right order of tasks in a list indicates the order of execution. The phenotype is the resulting schedule based upon this task allocation and execution ordering. The ES terminates after a fixed number of generations (Γ) have been produced and evaluated or earlier if a feasible schedule is found. The ES is implemented as follows:

1. Conduct a breadth-first search of the task graph numbering the vertices in the order visited.
2. Create an initial population of μ individuals by selecting vertices from the task graph in numerical order

and randomly assigning them to processors in the distributed system.

3. For each individual, generate one offspring by applying a mutation operator (described below). This creates a population with a total of 2μ individuals.

4. Evaluate all individuals to determine their fitness by computing the schedule length based upon the associated task assignments and execution order. Fitness increases as the number of tasks that meet their deadline increases.

5. Select the μ fittest individuals for survival. Discard the other individuals.

6. Proceed to step 3 unless a feasible solution has been found or Γ generations have been evaluated.

New genotypes are created with mutation operators that are applied with a specific probability. The two operators we used are denoted by M_1 and M_2 . M_1 swaps two adjacent tasks assigned to the same processor (affecting the execution order) while M_2 exchanges threads of execution between processors (affecting the load balance).

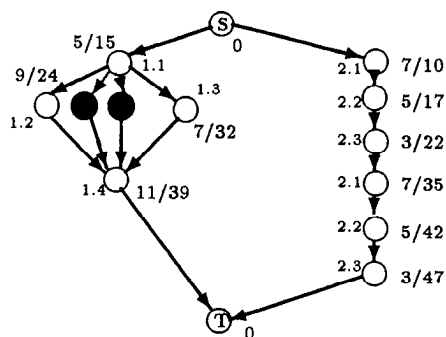


Figure 1: A Task Graph

Let $C(T_i)$ and $DL(T_i)$ represent the completion time and deadline, respectively, of task T_i . Clearly a schedule will only be feasible if $C(T_i) \leq DL(T_i) \forall i$. The fitness of genotype i with N tasks to schedule is defined as

$$G_f^i = \begin{cases} \sum_{k=0}^N DL(T_k) - C(T_k) & \text{if deadline not met} \\ 0 & \text{if deadline met} \end{cases}$$

Notice that the fitness of a genotype is negative if the schedule is infeasible and exactly 0 if it is feasible.

3 Results

To evaluate our scheduling technique, we chose the real-time scheduling problem defined by Ramamritham [3]. Two periodic tasks are to be scheduled in a 3-processor distributed system. Periodic Task 1

consists of 4 subtasks with a period of 50 time units. Periodic Task 2 consists of 3 subtasks with a period of 25 time units. Thus in a 50 time period, the subtasks of Periodic Task 1 (Periodic Task 2) will execute once (twice). The task graph for this problem is shown in figure 1. The notation X.Y means Periodic task X, subtask Y. A/B next to a node refers to the execution time and deadline, respectively. Subtask 1.2 is to be duplicated 2 more times to support fault tolerant requirements. (These replications are the dark nodes in figure 1.) These 3 subtasks must be assigned to a different processors in the distributed system and all must have the same execution times and deadlines. Task 1.4 "votes" on the correct result from these subtasks.

The mutation operators M_1 and M_2 were applied with probability p_1 and $p_2 = 1 - p_1$, respectively. We evaluated the performance of these operators for various values of p_1 and p_2 to see the affect on the convergence rate of the ES. A population size of $\mu=100$ was used for all tests and the average performance taken over 100 runs was recorded. For $p_2 > 0.7$, a feasible schedule was not found in around 25% of the runs. For $p_2 < 0.4$, a feasible schedule was found over 98% of the time. In support of fault tolerance, all replicated tasks must be assigned to distinct nodes. We conjecture that if p_2 is too large, meeting this fault tolerant requirement is difficult to achieve since tasks are frequently moved between processors. Based on these results, we suggest if the number of replicated tasks is large (with respect to the number of distinct nodes), $p_1 \approx p_2$ will give reasonable convergence rates.

The running time of the ES on a SPARC-IPC workstation took approximately four seconds to find this feasible schedule. Feasible schedules for this problem were found typically in less than 40 generations. This shows that the ES is a viable technique for quickly finding schedules in real-time, distributed systems.

References

- [1] G. Greenwood, A. Gupta, and K. McSweeney, "Scheduling Tasks in Multiprocessor Systems Using Evolutionary Strategies", *Proc. of 1st IEEE Conf. on Evolutionary Computation*, pp. 345-349, June 1994
- [2] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, 2nd ed., Springer Verlag, 1994
- [3] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks", Technical Report 90-01, Univ. of Massachusetts, Jan 1989