Sustainable Computing: Informatics and Systems 4 (2014) 241–251

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

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

Reoptimization of the minimum total flow-time scheduling problem☆

Guy Baram, Tami Tamir*

School of Computer Science, The Interdisciplinary Center, Herzliya, Israel

ARTICLE INFO

Article history:
Received 28 February 2014
Received in revised form 2 July 2014
Accepted 11 August 2014

Keywords:
Algorithms
Scheduling
Minimum flow-time
Transition cost
Reoptimization

ABSTRACT

We consider *reoptimization* problems arising in dynamic scheduling environments, such as manufacturing systems and virtual machine managers. Due to changes in the environment (out-of-order or new resources, modified jobs' processing requirements, etc.), the schedule needs to be modified. That is, jobs might be migrated from their current machine to a different one. Migrations are associated with a cost – due to relocation overhead and machine set-up times. In some systems, a migration is also associated with job extension. The goal is to find a good modified schedule, with a low transition cost from the initial one. We consider the objective of minimizing the total flow-time.

We present optimal algorithms for the problem of achieving an optimal solution using the minimal possible transition cost. The algorithms and their running times depend on our assumptions on the instance and the allowed modifications. For the modification of machines' addition, we also present an optimal algorithm for achieving the best possible schedule using a given limited budget for the transition.

1. Introduction

This work studies a reoptimization variant of the classical scheduling problem of minimizing the total flow-time on identical machines (denoted in standard scheduling notation by $P||\sum C_j$ [16]). The minimum total flow-time problem can be solved efficiently by the simple greedy Shortest Processing Time algorithm (SPT) [30,9] that assigns the jobs in nondecreasing order by their length. This algorithm, as many other algorithms for combinatorial optimization problems, solves the problem from scratch, for a single arbitrary instance without having any constraints or preferences regarding the required solution – as long as it achieves the optimal objective value. However, many of the real-life scenarios motivating these problems involve systems that change dynamically over time. Thus, throughout the continuous operation of such a system, it is required to compute solutions for new problem instances, derived from previous instances.

Moreover, since the transition from one solution to another consumes energy (used for the physical migration of the job, for warm-up or set-up of the machines, or for activation of the new machines), a natural goal is to have the solution for the new instance *close* to the original one (under certain distance measure).

Solving a *reoptimization* problem involves two challenges:

1 *Computing* an optimal (or close to the optimal) solution for the new instance.
2 Efficiently *converting* the current solution to the new one.

Each of these challenges, even when considered alone, gives rise to many theoretical and practical questions. Obviously, combining the two challenges is an important goal, which shows up in many applications.

Applications: The reoptimization problem of minimizing the total flow-time arises naturally in manufacturing systems, where jobs might be migrated among production lines. Due to unexpected changes in the environment (out-of-order or new machines, timetables of task processing, etc.), the production schedule needs to be modified. Rescheduling tasks involves energy-loss due to relocation overhead and machine set-up times. In fact, our work is relevant to any dynamic scheduling environment, in which migrations of jobs are allowed though associated with an overhead caused due to the need to handle the modification and to absorb the migrating jobs in their new assignment. We describe below several less natural applications in cloud computing and networking.

With the proliferation of cloud computing, more and more applications are deployed in the data centers. Live migration is a common process in which a running virtual machine (VM) or application

☆ A preliminary version of this paper appears in the proceedings of the 1st Mediterranean Conference on Algorithms (MedAlg) December 2012, Ein-Gedi, Israel.

* Corresponding author. Tel.: +972 99602779; fax: +972 9 9568604.
E-mail addresses: guy.baram@gmail.com (G. Baram), tami@idc.ac.il (T. Tamir).

http://dx.doi.org/10.1016/j.suscom.2014.08.011
2210-5379/

moves between different physical machines without disconnecting the client or application [10]. Memory, storage, and network connectivity of the virtual machine are transferred from the original host machine to the destination. Such migrations involve a warm-up phase, and a memory-copy phase. In pre-copy memory migration, the Hypervisor typically copies all the memory pages from source to destination while the VM is still running on the source. Alternatively, in post-copy memory migration the VM is suspended, a minimal subset of the execution state of the VM (CPU state, registers and, optionally, non-pageable memory) is transferred to the target, and the VM is then resumed at the target. Live migration is performed in several VM managers such as Parallels Virtuozzo [25] and Xen [33]. Lot of attention, in both the industry and the academia is given recently to the problem of minimizing the overhead associated with migrations (see e.g., [10,19]). Using our notations, this refers to minimizing the transition costs and the job-extension penalties associated with rescheduling a job. Our work focuses in determining the best possible schedule given these costs. Sequential processing of jobs that might be migrated among several processors is performed also in several implementations of MapReduce (e.g., [5]). These implementations assume that different segments of MapReduce computations can be processed independently on remote computers [12].

Our reoptimization problem arises also in an RPC (Remote Procedure Call) service. In this environment, a cloud of servers can provide service to a limited number of simultaneous users [8]. If the number of requests is high, another virtual server could be temporarily rented, where the cost for using it is per user. The options are to put the RPC in a queue, thus causing latency in the service, or renting more virtual servers, enabling faster service and paying the additional servers' cost. In this application, the transition cost is not due to the migration itself, but due to the activation cost of the additional resources.

Some of our results will be extended to consider modifications that occur after the processing has begun, that is, at time $t > 0$. For this extension (see Section 3) we distinguish between environments in which the currently processed jobs can migrate and be restarted on different machines, and applications in which restarts are not allowed, and a currently processed job must complete its partial processing. The following application describes a system in which restarts are not allowed: In a semiconductor wafers production line, some of the coating methods involve purely physical processes such as high temperature vacuum evaporation (physical vapor deposition – PVD). During the process, a vacuum is created to enable the coating. Once the elements are in a vacuum environment, the process cannot be stopped as if the machine halts, it will be severely damaged [22]. Assume that at time $t > 0$ machines are added. Transferring jobs is costly – to capture the transition overhead and the changes required in programming the machines workplan. Also, the elements that are currently produced, that are already in vacuum state, must complete their production.

1.1. Problem statement and notation

An instance of our problem consists of a set J_0 of n_0 jobs and a set M_0 of m_0 identical machines. Denote by p_j the processing time of job j. A schedule S_0 of the initial instance is given. That is, for every job in J_0, it is specified on which machine it is assigned and on which time interval it is going to be processed. At any time, a machine can process at most one job and a job can be processed by at most one machine.

At time $t \geq 0$, a change in the system occurs. Possible changes include addition or removal of machines and/or jobs, as well as modification of processing time of jobs in J_0. Let J denote the modified set of jobs, and let $n = |J|$. Let M denote the modified set of machines, and let $m = |M|$. Our goal is to suggest a new schedule, S, for the modified instance, with good objective value and small transition cost form S_0. Assignment of a job to a different machine in S_0 and S is denoted *migration* and is associated with a cost. Formally, we are given a *price list* $\theta_{i,i'j}$, such that it costs $\theta_{i,i'j}$ to migrate job j from machine i to machine i'.

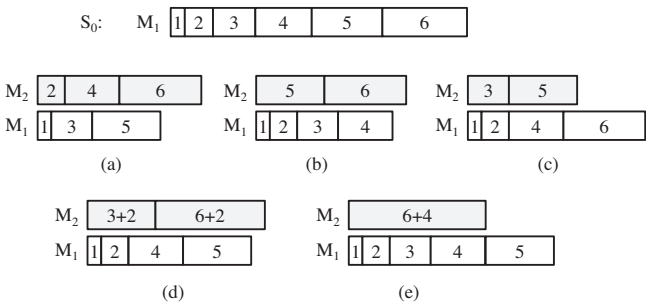

Fig. 1. (Top) An initial assignment, (a) an optimal reassignment achieved with transition cost 3, (b) a possible, and (c) an optimal reassignments achieved with limited budget $B = 2$, (d) an optimal reassignment assuming job-extension penalty $\delta = 2$, and (e) $\delta = 4$.

Moreover, in some systems job migrations are also associated with an extension of the job's processing time. In this extended model, in addition to the transition costs, we are given a *job-extension penalty list* $\delta_{i,i'j} \geq 0$, such that the processing time of job j is extended to $p_j + \delta_{i,i'j}$ when it is migrated from machine i to machine i'. We assume that all the involved parameters (processing times, transition costs and extension penalties) are integers. Some of our results assume unit transition costs, that is, for all j and $i \neq i'$, $\theta_{i,i'j} = 1$.[1]

For a given schedule, let C_j be the completion time of job j, that is, the time when the process of j completes. In this work we consider the problem of minimizing the sum of completion times, denote by $\sum C_j$ and also known as *total flow-time*. In the reoptimization problem, given S_0, J and M, the goal is to find a good schedule for J that is close to the initial schedule S_0.

We consider two problems:

1 Rescheduling to an optimal schedule using the minimal possible transition cost.
2 Given a budget B, find the best possible modified schedule that can be achieved without exceeding the budget B.

Example 1. Assume that six jobs of lengths 1, . . ., 6 are scheduled on a single machine in an optimal SPT order. The total flow time in this schedule is $\sum C_j = 56$. Assume that a second machine is added, and that all migrations have unit transition cost and no job-extension penalties. Fig. 1(a) presents an optimal modified schedule, for which the total flow-time is $\sum C_j = 34$. Migrating jobs appear in grey. The budget required to reach this schedule (or any other schedule with $\sum C_j = 34$) is 3. For a given budget, $B = 2$, it is possible to move, for example, to the modified schedules given in Fig. 1(b) and (c), having total flow-time 36 and 35, respectively. The schedule (c) is optimal for this budget.

Assume further that all migrations are associated with the same job-extension penalty, that is, for all i, i', j, we have $\delta_{i,i'j} = \delta$. An optimal solution for $\delta = 2$ achieving $\sum C_j = 41$ is given in Fig. 1(d). The transition cost to this schedule is 2, however, even with unlimited budget it is impossible to produce a schedule with lower total flow-time. For $\delta = 4$, only one job is migrated in the optimal solution (that achieves $\sum C_j = 45$), given in Fig. 1(e). The optimal algorithm

[1] Note that the constant 1 can be replaced by any other constant.