Two linear programming formulations for scheduling multi-user periodic arrival tasks

Emmanuel Medernach, Claire Hanen, Philippe Lacomme, Eric Sanlaville

Two formulations for a grid cyclic scheduling problem

Claire Hanen [*]   Philippe Lacomme [†]   Emmanuel Medernach [‡]

Eric Sanlaville (speaker) [§]

Grid cyclic scheduling : the EGEE case

Grid computing is now a major way to tackle large, possibly distributed computations. The european `EGEE` project [3] is a nice example of such systems. The `EGEE` grid is composed of many sites all around Europe and abroad. Each site disposes of several hundreds of computer units. Its first aim is to provide enough computation and storage resources for scientists, and especially for the physicists that will exploit the data of the LHC. Each experiment with the collider will provide an enormeous amount of data. It is also used for other applications, for instance concerning biology or health. The major issue when managing such enormous systems is the load balancing, while keeping consistency. `EGEE` is currently managed in a centralized way. This management raises complicated problems that are not within the scope of this presentation, which focuses on the management of one site. Each site is a parallel system. We will suppose here this system is a large cluster (that is the case for the site of the LPC, a physics laboratory in Clermont Ferrand, France. See [2]).

The cluster receives a large number of tasks. These tasks are usually independent. They come from different users, and have different characteristics (length, periodicity, memory requirements,...). During these months before the use of the LHC as an operational system, many simulations are done to verify if the grid, and the individual clusters, can bear the load. Typically, a LHC experiment will generate a flow of tasks (50 per second) with a large variety of durations (some with a few seconds of duration, some with a few hours). However, other types of tasks will have to be dealt with in the same time.

A cyclic scheduling model for the grid clusters

In this presentation, it is considered a flow of periodic tasks on a cluster of m identical machines. The following hypotheses are done:

- $< i, k >$ denotes the kth occurrence of task T_i. The total number of occurrences is supposed to be infinite.

- each occurrence has a release date $r_{ik} = r_i + k \cdot \alpha_i$, where α_i is called the period of task T_i

[*]`claire.hanen@lip6.fr`. LIP6, University of Paris 6, France

[†]`placomme@isima.fr`. LIMOS, University of Clermont-Ferrand, France

[‡]`medernac@clermont.in2p3.fr`. LIMOS and LPC, University of Clermont-Ferrand, France

[§]`eric.sanlaville@univ-lehavre.fr`. LITIS, University of Le Havre, France, and LIMOS, UMR CNRS 6158

- p_i is the processing time of task T_i. It is supposed to be known by the scheduler. The tasks are non preemptive. All durations are integer.

- Each task i is sent by a given user u.

The users may belong to different groups. Each user sends a set of tasks (sometimes called a bag of jobs). One characteristic of grid management (especially true for EGEE which is used by a large number of different users) is that the users are very sensitive to the equity between them. This implies that the cluster manager must schedule the tasks so that the cluster holds the load *and* the users are served with equity. We shall consider both requirements in turn.

The first requirement means simply that during the periodic part of a schedule, on average exactly one occurrence of T_i is processed during α_i time units. Note that in many cyclic scheduling problems, the task arrivals are not specified and the objective is to minimize the *cycle time*. Informally, the cycle time is the maximum duration between the end of two successive occurrences of the same task (see [1] for precise definitions and an overwiew). Our objective is different and can be expressed in terms of feasibility. A schedule is periodic of period T if, from a time θ, when an occurrence $< i, k >$ begins at time t_{ik}, then another occurrence of T_i begins at time $t_{ik} + T$. Note that the second occurrence is not always $< i, k + 1 >$, and that it is not obliged to be executed on the same machine. θ denotes the start of the periodic part. Concerning the problem at hand, we are looking for a periodic schedule which period is a multiple of each α_i. In the following, T is fixed to their least common multiplier. The partial schedule executed in an interval of size T is called a *pattern*. Another issue adressed in this presentation is the value of θ, that is, the size of the transitory part.

Basically , each user wishes that his tasks are executed at the shortest possible time after their submissions (release dates). But usually, a user will need the results of all the tasks of a same bag. Hence a good way of assessing the performance of a schedule for a given user u is to compute its em presence time: the sum of the lengths of the intervals during which at least one task of u is present in the system. Note that the presence time might be infinite. Hence it is better to use instead the ratio between the presence time during an interval of duration T, and T itself. The ratio, denoted *presence time ratio*, has value 1 if the tasks of u are always present. This is the case if some tasks have large durations (larger than T). In order to obtain fair schedules, we considered several agregated criteria like the mean, or the maximum, presence time ratio between all users which have no task of duration larger than or equal to T.

Two linear formulations for the grid cyclic scheduling problem

Let us first remark that the feasibility problem is simple. Indeed, an adaptation of the Mc-Naughton algorithm computes a feasible pattern of length T if any exists. Unfortunately, its performances in terms of presence time are not predictable. Hence we consider linear formulations with binary variables. They differ mainly by the way they consider the transitory part.

In the first formulation, the number of time intervals of duration T is fixed to K. The goal is to compute a partial schedule on the interval $[0, K \cdot T]$. Constraints are added to force that during the last interval, a *complete* pattern is obtained. A complete pattern contains exactly the right processing amount of each task. Other constraints are added to force that, once the first occurrence of a task is scheduled (at time t), other occurrences are scheduled at $t + T$, $t + 2T$,...

The figure shows the result obtained for a problem with two users and three machines. The first user has 3 tasks, the second user has 2 tasks. In this example all tasks have same period 12, hence $T = 12$, and a release date of 0 for the first user, of 9 for the second user. The linear program has been written for $K = 6$. However, it can be observed that a complete pattern appears at time $\theta = 24$. The pattern is then repeted at each T-interval. Taking $K = 3$ would have been sufficient.

Several remarks can be done from this example. A task is not assigned to the same machine on all intervals. On the contrary, the assignments of a task are obtained by circular permutation. Such a schedule is called *circular*. But other schedules might exist for which the machine assignments are obtained from a permutation that is not circular. This is an important point. Let us also consider the presence time for each user. It is easy to see that for user 1, there exists one task whose first occurrence finishes after T (the task of duration 9 finishes at time 29). This user is always present. For the second user, all occurrences of the second task finish 10 time units after their release dates, hence the presence time for the second user is 10 (presence time ratio equal to 5/6).
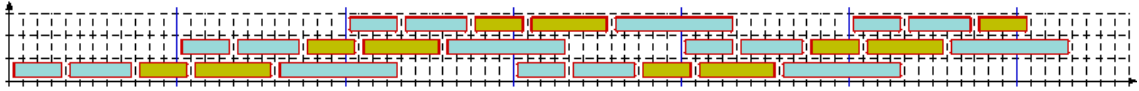

Figure 1: A cyclic schedule with its transitory part.

For practical purposes, the first formulation leads to a very large linear program that can not be solved efficiently for medium size instances.

The second formulation is based on the search for circular patterns. Indeed, a circular pattern of size T can be obtained by solving a 1-machine problem on a time interval of size $m \times T$. Then the m segments of size T might be attributed to each machine. It is possible to build a linear program with binary variables that uses this idea and minimizes one of our performance criteria. This, together with the fact that only one pattern is built, not the transitory part, explains that the linear program is much smaller. Furthermore, it is possible to compute from it the size of the transitory part necessary to obtain the pattern.

The main difficulty is that circular patterns may not be dominant. This major issue will be discussed during the presentation.

References

[1] C. Hanen and A. Munier (1995). *A Study of the Cyclic Scheduling Problem on Parallel Processors*. Discrete Applied Mathematics 57(2-3): 167-192.

[2] E. Medernach (2005). *Workload Analysis of a Cluster in a Grid Environment*. JSSPP 2005: 36-61.

[3] Enabling Grids for E-sciencE (EGEE project) http://www.eu-egee.org/