A general modular specification for distributed schedulers

Gerson G. H. Cavalheiro[†1], Yves Denneulin[2] and Jean-Louis Roch[1]

[1] Projet Apache[‡]
Laboratoire de Modélisation et Calcul, BP 53
F-38041 Grenoble Cedex 9, France
[2] Laboratoire d'Informatique Fondamentale de Lille
Cité Scientifique
F-59655 Villeneuve d'Ascq Cedex, France

Abstract. In parallel computing, performance is related both to algorithmic design choices at the application level and to the scheduling strategy. Concerning dynamic scheduling, general classifications have been proposed. They outline two fundamental units, related to control and information. In this paper, we propose a generic modular specification, based not on two but on four components. They and the interactions between them are precisely described. This specification has been used to implement various scheduling algorithms in two different parallel programming environments: PM^2 (ESPACE) and Athapascan (APACHE).

1 Introduction

From various general scheduling systems classifications for parallel architectures [2, 6], it is possible to extract two fundamental elements: a *control unit* to compute the distribution of work in a parallel machine and an *information unit* to evaluate the load of the machine. These units can implement different load balancing mechanisms, privileging either a (class of) machine architecture or an (class of) application structure.

A dynamic scheduling system (DSS for short) is particularly interesting in case of an unpredictable behavior at execution time. Irregular applications and networks of workstations (NOW) are typical cases where the execution behavior is unknown *a priori*. Usually a DSS for a tuple {*application,machine*} is specified by an entry in a scheduling classification, and is implemented as a black-box: it is difficult to tune it or reuse it for another application or machine without rewriting totally or partially its code. Some works in the literature discuss this problem and present possible solutions, like [4] where scheduling services are grouped in families and presented as functions prototypes.

In this paper we discuss the gap between the scheduling classifications and the implementation of DSSs. We present a modular structure for general scheduling

[†] CAPES–COFECUB Brazilian scholarship

[‡] Supported by CNRS, INPG, INRIA et UJF

systems. This general architecture, presented as a framework, aims at avoiding the introduction of additional constraints to implement the different classes of scheduling. This framework introduces two new functional units: a *work creation unit* and an *executor unit* and specifies an interface between the components.

2 General organization of a dynamic scheduling system

In this work a DSS is implemented at software level to control the execution of an application on a parallel architecture. It was conceived to be independent from both application and hardware and to support various classes of scheduling algorithms. The DSS runs on a parallel machine composed of several computing nodes, each one having a local memory and at least one real processor. There is also a support for communications among them.

The notion of job. Units of work are produced at the application level; from them, *jobs* are built and inserted in a graph of precedence (DAG), denoted G. G evolves dynamically; it represents at each instant of execution the current state of the application jobs. Each node in G represents a job; each edge, a precedence constraint between two jobs. So, a job is the basic element to be scheduled, it corresponds to a finite sequence of instructions, and has properties to allow its manipulation by the scheduler. At a given instant of time, a job is in one of the following states: *waiting*, the job has precedence constraints in G; *ready*, the job can be executed; *preempted*, the execution was interrupted; *running*, the job's instructions are executed; or *completed*, the precedence constraints in G imposed by the job are relaxed.

Correction of a scheduling algorithm. The scheduling manages the execution of jobs on the machine resources. For this, any DSS has to respect the following properties, that define its semantics:
(P_1) for the application: only jobs in the *ready* state can be put in the *running* state by the DSS;
(P_2) for the computing resources: if some resources of the machine (nodes) are idle while there are jobs in the *ready* state, at least one job will be put by the DSS in the *running* state after a finite delay. This delay, which may sometimes be bounded, defines the *liveness* of the DSS.
A DSS verifying those properties is said to be *correct*.

3 A modular architecture for a parallel scheduling

The DSS is implemented as a framework based on four modules (Fig. 1): a *Job Builder*, to construct and manage a graph of jobs; a *Scheduling Policy*, where the scheduling algorithm is implemented; a *Load Level*, to evaluate the load of the machine; and an *Executor*, which implements a pool of threads used to execute the jobs. Each module defines an interface to its services. An access to one of them represents a *read* or a *write* operation, both accomplished in a non-blocking fashion. A read operation is a request for data; it allows a module to get a data handled by another one. Write operations also provide resources for data exchange, but in the opposite direction: a module decides to send a data to another module (e.g. a data or status changes). As a reaction to the use of a service, an internal activity can be triggered and executed concurrently with the caller before the service ends. A short description of each module follows.

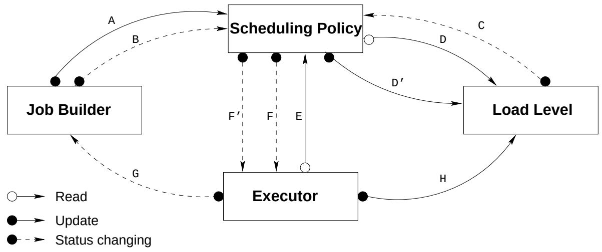

Fig. 1. Internal communication scheme between modules on the framework.

Job Builder unit. Its role is to build *jobs* from the tasks defined by the application, eventually giving access to their dependencies. So, it manages the evolutions of G. This module is attached to the application to receive and handle the tasks creation requests. The Job Builder unit must be informed when a job terminates to update the states of the jobs in G. In Fig. 1, arrow `G` shows the task Executor sending a job termination event to the Job Builder.

Scheduling Policy unit This is the core of the scheduling framework; it takes decisions about the placement of the jobs. In order to know how the application evolves, the Scheduling Policy receives from the Job Builder every change in G, e.g. when a job is created (arrow `A` in Fig. 1) or when its state changed (arrow `B`). To place the jobs to respect the load balancing scheme, the Scheduling Policy may consider the load on the parallel machine (arrow `D`). Notice that after taking the decisions, the load information must be updated (arrow `D'`). The Scheduling Policy must also react to two signals: an irregular load distribution detected by the Load Level (arrow `C`) and the requests of jobs to execute (arrow `E`) from the Executor. It can also create a new virtual processor (arrow `F`) or destruct one (arrow `F'`).

Executor unit. The Executor is implemented over a support that provides fine-grained activities; it furnishes virtual processors, called threads, to process the execution of the application jobs. Each thread executes an infinite loop of two actions: 1) send a request (arrow `E`) of a job to the Scheduling Policy; and 2) execute sequentially the job's instructions. The load information is updated after and before the execution of each job (arrow `H`).

Load Level unit. The Load Level unit collects the load informations in the computing system. These data are updated after the requests for write operations from the Scheduling Policy and Executor units. If an irregular distribution of the load in the machine is detected, this unit sends a message to the Scheduling Policy unit (arrow `C`). The definition of the load and the notion of irregular distribution depends on the Scheduling Policy unit.

4 Implemented environments

This general specification has been used to build Athapascan-GS and GTLB, both DSSs for two distinct parallel programming environments. Athapascan-GS [5] was implemented on top of Athapascan-0 [1] for the Athapascan-1 macro data-flow language (INRIA Apache Project, LMC laboratory) to support static and dynamic scheduling algorithms. Athapascan-GS can choose in the set of ready jobs the one that will be triggered in order to optimize a specific index of performance, like memory space, execution time or communication. The Generic Threads Load Balancer (GTLB) [3] defines a generic scheduler for highly irregular applications of tree search that belongs to the Branch&Bound family; it was implemented on the top of PM^2 (Espace project, LIFL laboratory) to support schedulers based on algorithms of mapping with migration. On both implementations, this design provides reusability opportunity in the development of specific scheduling algorithms.

5 Conclusion

In this paper, we have proposed a specification design to implement dynamic scheduling systems. This work completes classical classifications which analyze only two units of scheduling algorithms, dedicated respectively to the control and the load information. We claim that such a distinction is not precise enough; the major argument is that it does not take into account the relationships with the application (that produces tasks) and the execution (that consumes tasks). Two new modules are proposed: one dedicated to the work creation and another to the execution support, a protocol that specifies their interactions was also presented.

Two examples of DSSs implementing this general structure were presented: Athapascan-GS and GTLB, both supporting various scheduling algorithms.

References

1. J. Briat, I. Ginzburg, M. Pasin and B. Plateau. Athapascan Runtime: Efficiency for Irregular Problems. In *Proc. of the 3th Euro-Par Conference.* Passau, Aug. 1997.
2. T.L. Casavant and J.G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Trans. Soft. Eng.*. V. 14(2): 141-154, Fev. 1988.
3. Y. Denneulin. *Conception et ordonnancement des applications hautement irrégulières dans un contexte de parallélisme à grain fin.* PhD Thesis, Université de Lille, Jan. 1998.
4. C. Jacqmot. *Load Management in Distributed Computing Systems*: Towards Adaptative Strategies. DII, Université Catholique de Louvain, PhD Thesis, Louvain-la-Neuve, Jan. 1996.
5. J.-L. Roch et all. *Athapascan-1.* Apache Project, Grenoble, http://www-apache.imag.fr Oct. 1997.
6. M.H. Willebeek-LeMair and A.P. Reeves. Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE Trans. Par. and Dist. Syst.* V. 4(9): 979-993, Sept. 1993.