Optimized Scheduling for the Masses

David J. Montana
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
dmontana@bbn.com

Abstract

Most applications for which optimized scheduling would provide significant benefit are still performed either manually or using simple dispatch rules. The main impediment is cost. Since software and algorithms must currently be developed for each application individually, the costs of development are high. Only applications with large amounts of money tied to the quality of the schedules can justify these costs. We aim to change this by making optimized scheduling a commodity item (like spreadsheet-based number crunching). Central to this effort is Vishnu, a scheduling software system that is

- reconfigurable - a problem representation framework allows the user to specify the problem to solve
- optimizing - an automated scheduler uses genetic search to find optimized schedules
- web-based - all interaction with the system is via a web server, and the display is a standard web browser
- open-source - all the software is freely available and platform-independent

With the help of the open-source community, we hope to develop Vishnu to the point where it is sophisticated enough to perform well on most scheduling problems yet easy enough to use that it gains wide acceptance.

1 Reconfigurable Scheduling

While there have been a variety of important real-world success stories for automatic optimized scheduling, these are isolated cases. The vast majority of scheduling applications are still performed either manually or using simple dispatch rules. A challenge for the coming years is to make optimized scheduling accessible to all the applications for which it has until this point been impractical due to costs and complexities. A key part of addressing this challenge is likely to be what we refer to as "reconfigurable" scheduling.

Optimizing schedulers have traditionally targeted a single problem or narrow class of problems. Changing a scheduler to handle a new problem or domain has required redesigning the scheduler and rewriting portions of its software. This is expensive and time-consuming and therefore has limited the applicability of this technology.

A reconfigurable scheduler can handle a wide range of different scheduling applications and domains without modification of its software. The user specifies the problem to solve as configuration data, which is sent as input to the scheduler. The scheduler returns the assignments as outputs. There are different degrees of reconfigurability depending on the expressiveness and flexibility of the problem representation framework and to what extent the scheduler can solve any problem represented in this framework. A truly reconfigurable scheduler should at least handle most classical scheduling problems.

Having a reconfigurable scheduler opens up the possibility of low-cost optimized scheduling, i.e. optimized scheduling as a commodity item. If the reconfigurable scheduler is sufficiently powerful and easy to use, then configuring it for a new problem should be quick and inexpensive, even if the problem is unlike any that it is currently configured to solve.

There have been a variety of reconfigurable schedulers implemented. Most of these use a custom computer language for representing the problem and a separate solver that can solve problems expressed in this language. Perhaps the most widely used such language is AMPL [Fourer *et al.*, 1993]. AMPL provides a way to express mathematical programming problems, including some scheduling problems. There exist multiple solvers for AMPL including one from ILOG. Another language is OPL (Optimization Program-

ming Language) [Hentenryck, 1999]. It is oriented towards constraint-based scheduling, and ILOG also has a solver for it. The Computer-Aided Constraint Programming Project has developed a language called EaCL (Easy Constraint Language) plus an associated solver, and these are also oriented towards constraint-based scheduling. The Starflip project has created a language for expressing scheduling problems with fuzzy constraints and a corresponding solver [Raggl and Slany, 1998].

We have developed a new reconfigurable scheduler called Vishnu that is distinguished by the simplicity of its problem representation framework (which uses a fixed-size set of formulas rather than a full-scale language), the efficiency of its solver (which uses a genetic algorithm along with smart caching of results of formulas), and its web-based architecture (which uses a standard browser as the user interface for viewing schedules and editing problems). Because Vishnu was just recently developed and is not yet known to the scheduling community, we next provide a relatively brief description of how it works. We then discuss how we are releasing Vishnu open source and make the case for why the scheduling community should adopt it as a standard and as a focus of a unified research and development effort.

2 The Vishnu Scheduling System

We start by describing a framework that provides a way to define scheduling problems to solve. We then examine a scheduler that can find optimized schedules for problems defined using the framework. Finally, we discuss the full web-based system, called Vishnu, that integrates the problem representation framework, the scheduler, a browser-based user interface, and an interface to external processes.

2.1 Problem Representation Framework

A problem representation has three components: the metadata, the data, and the scheduling semantics. We now give a brief description of each of these. For a detailed description, see [Montana, 2001b], or for a more comprehensive overview, see [Montana, 2001a].

The metadata defines problem-specific data formats. It specifies complex data structures, or objects, constructed from other simpler, primarily atomic, data types. The atomic types are string, number, datetime, boolean, and list. One of these data structures must be designated as representing tasks, and a different one as representing resources.

The data are instances of objects. The data for every problem must include tasks and resources. Data of other types are also permitted. The other data can include business rules, distance matrices, or anything needed to define the logic for how to schedule.

Constraint	Formula
Optimization Criterion	maxover (resources, “resource”, complete (resource)) - starttime
Capability	task.machine = resource.id
Task Duration	task.duration
Prerequisites	if (task.preceedingstep != “”, list (preceedingstep))
Task Unavailability	mapover (prerequisites, “task2”, interval (starttime, taskendtime (task2)))

Table 1: Constraints for Job-shop Scheduling Problem

The most interesting part of a problem representation is the scheduling semantics, which tell how to use the data to perform scheduling. The scheduling semantics are defined by a set of formulas involving the fields of the metadata. For each type of scheduling constraint, such as whether a certain resource can perform a certain task or when a particular task is available to be scheduled, there exists a hook for a user to specify a formula. For each hook/constraint, there is a set of variables (e.g., *task* and *resource*) whose values set the context. Each constraint also has a default formula so that the user only has to define formulas for those hooks that are relevant to the problem. The concept of scheduling semantics is best illustrated by examples.

Example 1: Job-shop scheduling - There are M machines and N manufacturing jobs to be completed. Each job has M steps, with each step corresponding to a different specified machine. There is a specified order in which the steps for a certain job must be performed, with one step not able to start until the previous step has ended. The objective is to minimize the end time of the last step completed.

The task object, step, and resource object, machine, are defined as:
- **step** - id (string), duration (number), machine (string), and preceedingstep (string)
- **machine** - id (string)

The constraints with non-default values are shown in Table 1.

Example 2: Vehicle Routing with Time Windows - There are M vehicles and N customers from whom to pick up cargo. Each vehicle has a limited capacity for cargo, and each piece of cargo contributes a different amount towards this capacity. There is a certain window of time in which each pickup must be initiated, and the pickups require a certain non-zero time. Each vehicle that is utilized starts at a central depot, makes a circuit of all its customers, and then returns to the depot. The objective is to minimize the

Constraint	Formula
Optimization Criterion	sumover (resources, ”resource”, preptime (resource)) + sumover (tasks, ”task”, if (hasvalue (resourcefor (task)), 0, 1000))
Delta Criterion	preptime (resource) - previousdelta (resource)
Task Duration	extra.servicetime
Setup Duration	dist (task.location, if (hasvalue (previous), previous.location, extra.depotlocation))
Wrapup Duration	if (hasvalue (next), 0, dist (task.location, extra.depotlocation))
Task Unavailability	list (interval (starttime, starttime + task.earliest), interval (starttime + task.latest + extra.servicetime, endtime))
Capacity Contributions	list (task.load)
Capacity Thresholds	list (resource.capacity)

Table 2: Constraints for Vehicle Routing Problem

total distance traveled by the vehicles.

The task, resource, and other object are respectively:

- **customer** - id (string), load (number), earliest (number), latest (number), and location (xycoord)
- **vehicle** - id (string) and capacity (number)
- **extradata** - servicetime (number) and depotlocation (xycoord)

The single object of type extradata is named *extra*. The constraints with non-default values are shown in Table 2.

2.2 Genetic Scheduler

Evolutionary algorithms are flexible enough to solve most scheduling problems. However, genetic schedulers traditionally have been targeted towards particular problems. For each problem, researchers have developed a problem-specific representation and problem-specific operators [Montana, 1998]. We have created a different type of genetic scheduler, capable of finding optimized schedules for any problem specified using our problem representation framework. The details of how our reconfigurable scheduler works are in [Montana, 2001b]; here we give just a brief overview.

The basic approach is that introduced by [Whitley *et al.*, 1989] and refined by [Syswerda, 1991]. An order-based genetic algorithm is wrapped around a greedy schedule builder. The genetic algorithm feeds different task orderings to the greedy scheduler. The greedy scheduler assigns tasks one at a time in the order provided (to the extent that this is possible), assuring that all hard constraints are met and selecting the best possible resource and time for each task. The formulas in the scheduling semantics allow the genetic algorithm and greedy sceduler to compute any problem-specific information that they need.

The performance of the reconfigurable scheduler is adequate when the problem is not too large or when getting a good, but not necessarily optimal, solution suffices. For a problem, such as the traveling salesman problem, where a custom heuristic already exists, our scheduler generally does not perform as well as the custom heuristic. As we discuss in 4, in the long run there is the potential for acheiving high performance without sacrificing generality by making the scheduler smarter about recognizing classes of problems and applying specialized algorithms.

2.3 Web-based Architecture

We have wrapped the reconfigurable scheduler and problem representation framework in a web-based system, which we call Vishnu. The architecture for this system is shown in Figure 1. The central database stores multiple scheduling problems, where a problem includes both the problem definition and any assignments made by the scheduler. The web server moderates all database accesses and communications between different components. The scheduler(s) is(are) just another client and can be replaced with any scheduler that obeys the defined interface specifications.

The browser-based user interface allows multiple users to interact with the scheduler and problem representations in different ways. For one, the user can graphically edit (or create) problem definitions. The user can also initiate and cancel scheduler runs. (The scheduler only executes on a problem when requested.) Using a browser, the user can view color-coded graphical representations of the schedules created by the scheduler. Eventually, we plan for Vishnu to allow the user to manually override the decisions of the scheduler.

An alternative way of interacting with the system is via an external process. There exist a set of well-defined data formats and URLs that allow an automated feed of data from an external process. For example, one could automatically feed changes from a central corporate database into a scheduler. Another type of external feed is from a Cougaar multiagent system [Montana *et al.*, 2000]. Cougaar is an open-source multiagent architecture into which we have tightly integrated Vishnu. The Cougaar-Vishnu bridge that we have written automatically performs most of the steps

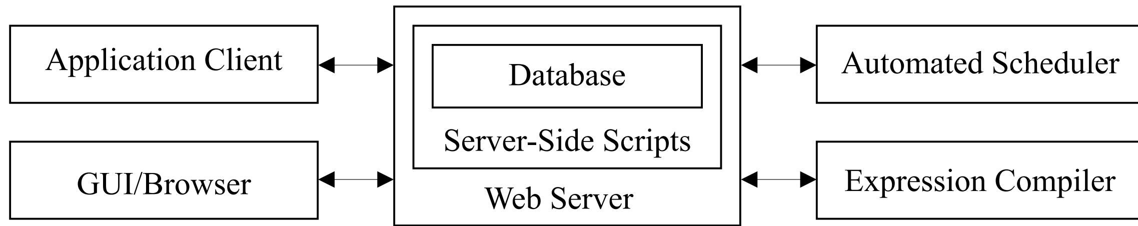

Figure 1: Vishnu system architecture

required to make a Vishnu-based Cougaar agent, including defining the metadata and data, feeding the problem to Vishnu, reading back the assignments from Vishnu, and updating the Cougaar database appropriately.

3 The Future: A Call for Support

We are releasing Vishnu open source, which means that we will allow free distribution of not only the executable code but also the source code. The power of the open source approach has been demonstrated most notably by the Linux operating system and the Apache web server, two pieces of software whose high quality and high acceptance stem from the distributed community of developers that support them. We hope that a critical mass of both the scheduling community and the open-source software community will provide similar support to Vishnu. In addition to a unification of effort for research and software development, such a joint undertaking would help in the development of standards for the scheduling community. We now discuss these benefits in more detail.

3.1 Standards for Scheduling

Definition of and adherence to a set of standards allows easy interchange of both data and software components. An example of this is the MP3 standard. Users can exchange audio files represented using the MP3 format knowing that any MP3-compliant player can handle these files. Some of the standards that Vishnu can help establish are for:

Scheduling problem definition - For Vishnu, we have developed a general representation for a scheduling problem, one that we hope will continue to grow and change. While we have better ways for showing a scheduling problem to a human, when we save such a problem to file we use XML. We have defined a DTD specifying an XML format that we sometimes refer to as SchXML (pronounced skex-em-el). It allows interchange of scheduling problems between machines, between researchers, and between systems (e.g., for a feed from a central corporate database). Anybody can read anybody else's problem and data with no possibility of misinterpretation.

Automated scheduler interface - In Vishnu, we have also developed some formats and protocols that allow the scheduler to grab problems from the web server and write back the assignments. While not yet formally defined in the same way as the problem representation, such standards would allow interchangability of schedulers. If somebody were to devise a scheduler that solves a particular class of problems best, anybody else could download the code and install it in their system. There would be no need to change the rest of the system in order to plug in the new scheduler. Such standards would also allow use of multiple schedulers, each suitable to a particular class of problems, with the ability to invoke the best scheduler for a particular problem.

Scheduling problem results - Another important standard is that for representing the set of assignments that have been made by a scheduler. With such a standard, users can exchange results. Displays for viewing and manipulating these results can then be written that will handle any data specified in the standard format.

3.2 Shared Efforts and Common Goals

Expanding on what we said above, open source software promotes the creation of a community. This is not just a community of users but also of developers since many of the users also contribute actively to improving the software. Instead of each scheduling system developer starting from scratch and building a custom system for each problem, there is the possibility of a combined effort continually building a better system. For example, to extend what one developer did, a second developer need just grab their code, plug it into his or her own Vishnu instance, make changes, and return the updated code to the central repository for others to use and extend/modify. Efforts are not duplicated and are instead geared towards mutual reinforcement.

4 Some Research Issues

The current version of Vishnu is just a starting point. We need to continue to make a variety of improvements before we can realize its promise. Of most interest to this audience are the still-open scheduling research issues that need to be

investigatated in the future. These include:

Automatic scheduler customization - The current automated scheduler implements a single algorithm that handles all cases. This provides the advantage of simplicity but has the major disadvantage of suboptimal performance. Algorithms tuned specifically to a limited class of problems can often outperform our current general approach for problems in this class. We could improve performance of the automated scheduler by including a variety of different scheduling algorithms, each targeted to a particular class of problems, plus some logic that automatically selects which scheduling algorithm to use for a given problem.

Problem representation improvements - There are still many concepts not yet capable of being represented in our problem representation framework that are necessary for describing certain scheduling problems. For example, our current handling of capacity is oversimplified and does not allow for concepts such as unloading and then reloading. We are also missing some concepts that would help with dynamic rescheduling, such as a soft constraint on schedule stability. We need to expand the representation capability in order to expand the applicability of Vishnu.

Human-scheduler interaction - Currently, Vishnu does not implement alerting, i.e. the automated scheduler notifying the human about potential problems with the schedule. Alerting would allow the human to make a decision about how to handle each problem, including the possibility of going outside of the constraints of the scheduling problem (e.g., renegotiating a delivery date or diverting resources). Humans also need the capability, not currently provided, to override the decisions of the automated scheduler. This applies particularly to those cases when not all information is captured in the problem definition.

5 Conclusion

Despite the large amount of research on optimized scheduling, applications of optimized scheduling are still relatively rare. This is because of the high cost of development of problem-specific scheduling systems. Reconfigurable scheduling provides the opportunity to greatly reduce the costs of configuring a scheduling system for a particular problem by eliminating the need for development of new software or algorithms. Vishnu is such a reconfigurable scheduling system and offers the promise of some day bringing optimized scheduling to the masses of computer users. We have released Vishnu open source not only to make it available to everybody but also to encourage the scheduling community to adopt it as a standard and to help develop it to the point where it realizes its promise.

Acknowledgments

This work was supported by DARPA contract MDA972-97-C-0800 under the Advanced Logistics Program. Thanks to Gordon Vidaver for his ideas and his help coding and to Todd Carrico for challenging us to build a generic scheduler.

References

[Fourer *et al.*, 1993] R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 1993.

[Hentenryck, 1999] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.

[Montana *et al.*, 2000] D. Montana, J. Herrero, G. Vidaver, and G. Bidwell. A multiagent society for military transporation scheduling. *Journal of Scheduling*, 3(4):225–246, 2000.

[Montana, 1998] D. Montana. Introduction to the special issue: Evolutionary algorithms for scheduling. *Evolutionary Computation*, 6(1):v–ix, 1998.

[Montana, 2001a] D. Montana. A problem representation framework for a reconfigurable scheduler. In *Proceedings of the Internation Joint Conference on Artificial Intelligence*, 2001. Submitted: under review.

[Montana, 2001b] D. Montana. A reconfigurable optimizing scheduler. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2001. To appear.

[Raggl and Slany, 1998] A. Raggl and W. Slany. A reusable iterative optimization library to solve combinatorial problems with approximate reasoning. *International Journal of Approximate Reasoning*, 19(1-2):161–191, 1998.

[Syswerda, 1991] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[Whitley *et al.*, 1989] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, 1989.