Deterministic Scheduling In A Distributed Computing Environment

Song Yom, Karin Loya
Hughes Information Technology Corporation
1616 McCormick Drive
Upper Marlboro, MD 20774
{syom,kloya}@eos.hitc.com

Joe Hunt
Science Applications International Corporation
8301 Greensboro Drive
McLean, VA 22102
jdh@cip.saic.com

Abstract

A key application within NASA's Earth Observing System Data and Information System (EOSDIS) Core System (ECS) is the Planning and Data Processing System (PDPS) which enables fully-automated production processing of science data in a distributed, heterogeneous, multiprocessor, parallel computing environment. This paper describes the implementation of the planning and scheduling algorithm to generate efficient, near-optimal production schedules based on predicted data availability, site configuration, hardware/software resource availability, and on task load.

Introduction

NASA's Earth Observing System Data and Information System (EOSDIS) developed under the Mission to Planet Earth (MTPE) program is an open, distributed data and information system that will manage data from pre-EOS and EOS-era Earth observation satellites and other field measurement programs. EOSDIS will provide EOS instrument data collection, science data processing, and services for distribution of data holdings. The EOSDIS Core System (ECS) is the infrastructure of EOSDIS and includes the Planning and Data Processing System (PDPS) within which the science processing software is integrated.

The PDPS is comprised of three components: the first provides capabilities to support the integration of delivered science software into the production environment; second, the Planning Workbench, is the application framework for the planning and scheduling algorithm; and third, the science processing software execution manager which, with an embedded commercial-off-the-shelf (COTS) product, is the production engine.

The focus of this paper is the planning and scheduling algorithm, designed to:

- generate a near-optimal plan
- facilitate efficient use of resources in production
- enable forecasting of product availability
- allow analysis of alternative strategies to expedite product generation
- provide the base schedule to drive production over the operational window

The science software is profiled during the integration process, so that characteristics of the tasks to be planned have been pre-determined in advance of the planning and scheduling activity that is invoked through the Planning Workbench.

The Planning Workbench, implemented on an off-the-shelf C++ scheduling framework, performs deterministic scheduling based on the Highest Level First (HLF) List Scheduling algorithm otherwise known as the Critical Path Method (CPM). The task model consists of having precedence relationships between tasks enforced by data dependencies; non-preemptive, processor (CPU) allocation of task based on requisite runtime (size); communication delay embedded in task size; user definable relative priority for tasks; and earliest available start time for tasks based on external dependencies and forecast data arrivals. The resource model includes computers, processors, near-line/off-line storage system, and planned unavailability of these resources. No cache management of disks is considered by the algorithm. Internally, the system employs a linked-list implementation of a graph data structure for tasks; recursive graph traversal for calculating relative priority; and merge sort algorithm for task sorting based on user specified earliest start time, user specified priority, optimal start time, and calculated priority for tasks. The algorithm implements best-fit scheduling optimization by performing "look-ahead" and "look-behind" operations on an interval based multi-level linked-list resource data structure.

Problem Domain

The primary focus of the PDPS is the scheduling and execution of science processing software to automate the generation and archiving of data products. Output data product from an instance of a science processing software

(task) can be both an end product to be disseminated to the science community for analysis and an intermediate product to be used as input to another task.

The system is data- and event-driven. Request is made of the system to execute a task based on availability of required input data products which can be internally generated by another task and/or provided by an external data source such as raw spacecraft instrument data. As illustrated by Figure 1, tasks have precedence relationship enforced by data dependency. A task that depends on data produced by an upstream (predecessor) task must be scheduled after the upstream task. In addition, a task that is waiting on data from an external source must be delayed as necessary for data arrival. The task model also accounts for multiple tasks sharing one or more input data.

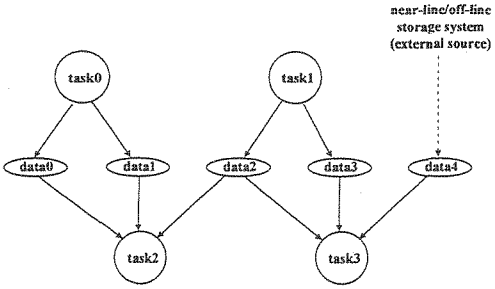

Figure 1. The Task Model describes the precedence relationship between tasks enforced by data dependency. A task can also depend on data arriving from an external source.

Tasks must be planned and scheduled on resources that consist of multiprocessor computing hardware, disks (local and network), and near-line/off-line storage system (dataserver). Dataserver is a hardware/software component of the ECS that provides data archiving and distribution services. For example, raw spacecraft instrument data ingested to the dataserver are retrieved by the PDPS as input to science processing software, and generated output data products are inserted back into the dataserver system for downstream processing.

The system must support planned resource unavailability. Resource reservations (also referred to in the ECS as ground events) can be made against processors, computers, and disks. Resources may be reserved for maintenance purposes. Forecast data arrival times can also be affected by scheduling ground events on the dataserver system. Load balancing is another factor for which the system must account. Over and under utilization of resources are to be avoided for efficiency of operation. Figure 2 describes the resource model.

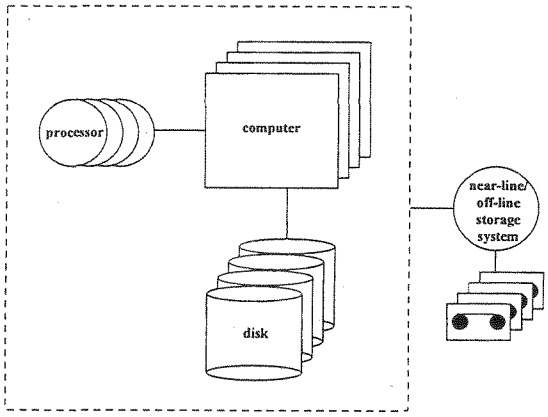

Figure 2. The Resource Model consists of multiprocessor computers with disks and a near-line/off-line storage system. It also includes planned unavailability of resources.

Merging the two models yields the basic problem domain view shown in Figure 3.

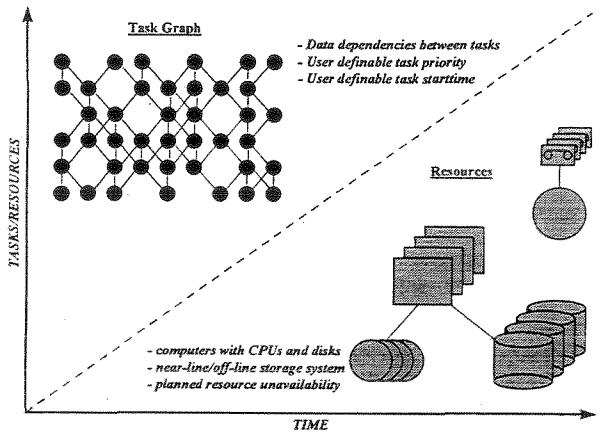

Figure 3. The overall problem domain view where tasks have to be scheduled on resources for optimal processing time and efficient resource utilization.

The challenge is the scheduling of competing tasks on resources with planned unavailability to achieve optimal processing time and efficient resource utilization. The solution must address user-definable task priority and earliest possible start time for tasks, based on predicted data arrival from an external source. These two comparison attributes must be used for task prioritization in order to support the requirements of the operations personnel. Furthermore, tasks must be scheduled around planned resource unavailability to not impact normal maintenance of hardware and software components of the system.

The approach taken to solve this problem was to implement the solution in two phases. First, research into the planning and scheduling problem domain was followed by the design and development of a prototype algorithm software application. Once the prototype was completed, the algorithm was incorporated into the Planning Workbench software component of the PDPS as the planning and scheduling engine.

Algorithm

The planning and scheduling algorithm implemented is based on the Highest Level First (HLF) List Scheduling algorithm or the Critical Path Method (CPM) [1][2]. The basic principle behind the algorithm is to assign nodes in the task graph a level (priority) which is the longest path from the node to the leaf node and scheduling the tasks to ready resources in descending priority.

As described previously, the primary use of the algorithm is to generate an initial production schedule from a strategic planning perspective and for forecasting product availability, since the actual task execution is managed by a COTS software product. Thus, a very complex and elaborate solution would not have been fully utilized.

Figure 4 depicts the overall scheme of the planning and scheduling algorithm.

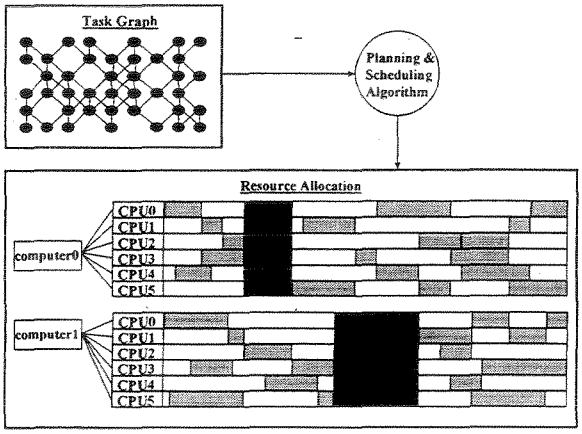

Figure 4. The algorithm sorts and prioritizes the tasks and performs non-preemptive, processor allocation of tasks around planned resource unavailability.

The algorithm performs non-preemptive, processor allocation of tasks. Communication delay between tasks is assumed to be embedded in the task size (runtime). Since the disk resource consumption is not controlled by the algorithm in real-time, no cache management of disk is designed into the algorithm. Therefore, aggregate size of all input and output data for a task is used primarily for scheduling around planned disk resource unavailability. Planned processor, computer, and dataserver resource unavailability is also taken into account during task-to-resource allocation phase in order to avoid possible resource contention between tasks and scheduled outages of resources. Finally, user-specified task priority and earliest start time for tasks based on external data arrival are used for sorting and prioritizing the task graph. Once the task graph and resources are loaded, the algorithm allocates tasks to earliest possible time slots on processors and disks to achieve optimal processing time, while attempting to load balance for efficient use of resources.

The overall data structure utilized in the algorithm is illustrated by Figure 5.

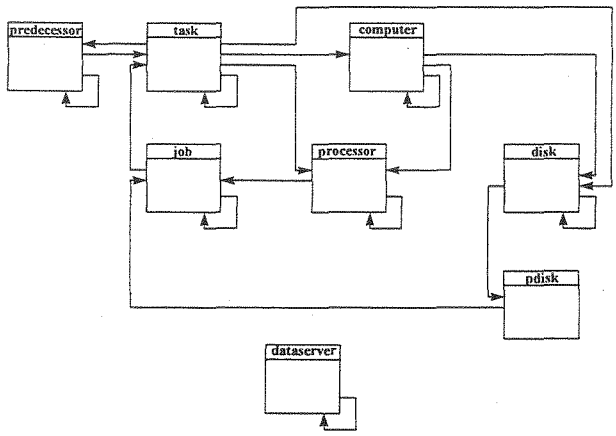

Figure 5. The algorithm uses a linked-list implementation of a weakly-connected, directed graph data structure for tasks and multi-level linked-list data structure for resources. Dataserver is a simple linked-list.

Linked-list implementation of a weakly-connected, directed graph data structure was chosen to represent the task graph for efficient traversal in loading, sorting, prioritizing, and scheduling operations. Multi-level linked-list computer hardware resource data structure was designed to logically map the available resources. Mapping of disks to physical disks was designed to support both local and network attached devices. Finally, a simple linked-list data structure was implemented for dataserver component of the resources for planned resource unavailability purpose.

There are four steps the algorithm performs in generating a schedule: load tasks, load resources and planned unavailability of them, sort and prioritize tasks, and schedule tasks on resources.

Tasks with attributes shown in Table 1 are loaded into the task graph with task as vertex and predecessor as edge.

Table 1. Task Definition

Attribute	Description
name	task name
priority	user specified priority
start time	estimated earliest start time for task based on data availability
duration	duration of task
size	aggregate size of data
predecessor	predecessor task name

Next, resources and planned unavailability are loaded into the multi-level linked-list data structure. Attributes are shown in Table 2.

Table 2. Resource Definition

Attribute	Description
name	computer name
processor	processor name
disk	disk name
size	capacity of disk
start time	resource unavailability start time
duration	duration of resource unavailability

Once tasks and resources are loaded, the algorithm sorts and prioritizes the tasks. The task graph is traversed to calculate and assign priorities and earliest possible start times to nodes. The merge-sort algorithm is used for sorting the task graph based on following comparison criteria (from highest to lowest order of precedence):

- earliest possible start time
- user-specified priority
- calculated priority
- number of immediate successor tasks

Now the fun begins. Traversing the prioritized task graph, the algorithm determines the earliest possible time slot a task can be executed on processor and disk resources. This time is verified against the dataserver unavailability to ensure no conflict exists. If it encounters a situation where the task can be allocated to more than one processor at an identical time, total processor runtime allocation is used as a tie-breaker for load balancing purposes.

The logic behind the finding of the earliest possible time slot on processors and disks is a simple one. Processor and disk resource data structures are traversed in sequence starting from time zero reference point. When a suitable time slot is located, it stops. This design ensures that resources are saturated as much as possible with running tasks so that best-fit scheduling optimization is achieved.

As tasks get allocated to available resources, linkage between task and resources are established via an abstract object called the job. This way, the generated schedule can be retrieved by traversing either the task graph or the resource data structure.

To summarize, the implementation of the planning and scheduling algorithm was presented in this section. Research into the problem domain was followed by a prototype algorithm development approach. Extensions were added to the HLF/CPM algorithm to meet the requirements of the ECS science data processing system. It was shown that (1) user-definable task priority, (2) earliest task start time based on predicted data arrival, and (3) planned resource unavailability in a distributed, multiprocessor, parallel computing environment were all addressed by the algorithm.

Implementation

The planning and scheduling algorithm described in the previous section was incorporated into the Planning Workbench software component of the PDPS. The Planning Workbench system is used to prepare production schedules in order to forecast start and completion times of data processing activities at the Distributed Active Archive Center (DAAC).

Using object-oriented methodologies, the Planning Workbench was implemented on the Hughes Delphi™ off-the-shelf C++ class library infrastructure. Delphi™ is based on a system of distributed, modular components where each component represents a distinct planning function.

The object-oriented architecture of Delphi™ is a variant of the model-view-controller (MVC) paradigm [3]. The MVC is an idealized view of the heterogeneous world of real systems. The *model* mirrors the structure of the user's conceptual model. The *view* shares the same structure and presents the model in a tangible form. The *controller* allows the user to interact with the model.

Delphi™ provides classes for an abstract model which are known collectively as the Resource Model. The Resource Model consists of resources, resource states, and all relevant resource constraints. Resources necessary for planning and scheduling are implemented as objects within the Resource Model. Besides modeling real world objects, another function of the Resource Model is to keep track of resource states over time.

In addition to resources and resource states, the Resource Model contains activities which are schedule-able entities known as tasks. During planning and scheduling, resources are assigned to activities and resource states are updated to include activities. Plans are aggregation of activities and resources that represent the science data processing production plans of the PDPS. Figure 6 describes the Resource Model in the Planning Workbench.

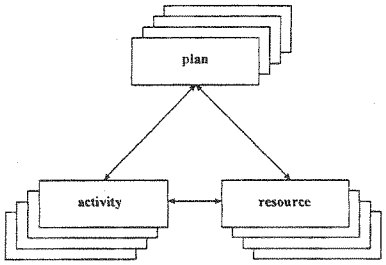

Figure 6. The Resource Model view of the Planning Workbench system where plans consist of activities and resources.

The Planning Workbench system consists of the Resource Model, the Scheduler, and the Timeline. The Resource Model constructs the resource configuration of the system by querying the PDPS database. The Scheduler retrieves user-specified science data processing tasks from the PDPS database, performs static scheduling, and directs the COTS scheduler product to execute tasks on target computers. To complete the loop, the COTS scheduler product updates the PDPS database with status of tasks. Finally, the Timeline component of the Planning Workbench system provides a GANTT style graphical view of activities and resources over time. The internal architecture of the Planning Workbench system and the external interfaces to COTS products are shown in Figure 7.

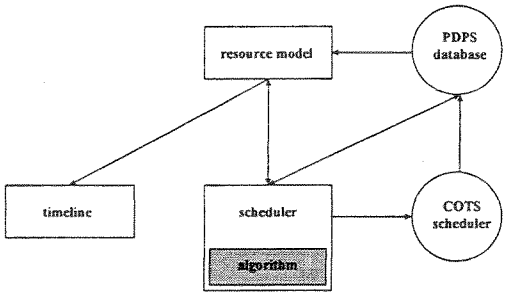

Figure 7. The inner workings of the Planning Workbench system with external interfaces to COTS products for production plan activation and management.

The implementation of the Planning Workbench system involved the incorporation of the planning and scheduling algorithm as the engine and the development of concrete classes which represent objects in the PDPS domain such as computers, processors, resource states, resource unavailability, and science data processing requests.

Currently, the PDPS is deployed and operational as a Science Software Integration and Test (SSIT) testbed at four Distributed Active Archive Centers (DAACs): the EROS Data Center (EDC), the Goddard Space Flight Center (GSFC), the Langley Research Center (LaRC), and the National Snow and Ice Data Center (NSIDC). It is the basis for the enhanced ECS which is scheduled for deployment in 1998.

Conclusions and Future Work

This paper has presented the implementation of the planning and scheduling algorithm to satisfy the science data processing requirements of the ECS. The overall problem domain was described, and details of the algorithm development were presented. A heuristic approach was taken to design the algorithm, and systematic software engineering steps were followed to implement the final solution.

The algorithm was designed for planning and forecasting purposes. To take full advantage of the effectiveness of the algorithm, and to realize its potential as a production engine, it should be enhanced to perform dynamic scheduling where the system manages task execution in order to perform decision making in real-time.

Acknowledgments

This work was funded by NASA Contract NAS5-60000. The authors would like to express appreciation to Tom Atwater for comments.

References

[1] El-Rewini, H., Lewis, T., Ali, H. 1994. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, pp. 56-81

[2] Pinedo, M. 1995. *Scheduling Theory, Algorithms, and Systems*. Prentice Hall, pp. 61-92

[3] Goldberg, A. July 1990. Information Models, Views, and Controllers. Dr. Dobb's Journal, pp. 54-60