NASA's 1995 Conference on Mass Storage Systems and Technologies

Constraint Based Scheduling for the Goddard Space Flight Center Distributed Active Archive Center's Data Archive and Distribution System

Nick Short Jr. - Information Science and Technology Branch
NASA - GSFC
Greenbelt Road
Greenbelt, MD 20771
301-286-6604
short@dunloggin.gsfc.nasa.gov

Jean-Jacques Bedet and Lee Bodden - Hughes STX
7701 Greenbelt Road, suite 400
Greenbelt, MD 20770
301-441-4285 Fax (301) 441-2392
{bedet,bodden}@daac.gsfc.nasa.gov

Mark Boddy, Jim White, and John Beane - Honeywell Technology Center
Honeywell Technology Center
3660 Technology Dr.
Minneapolis, MN 55418
612-951-7355 Fax 612-951-7438
{boddy,jwhite,beane}@src.honeywell.com

Abstract

The Goddard Space Flight Center (GSFC) Distributed Active Archive Center (DAAC) has been operational since October 1, 1993. Its mission is to support the Earth Observing System (EOS) by providing rapid access to EOS data and analysis products, and to test Earth Observing System Data and Information System (EOSDIS) design concepts. One of the challenges is to ensure quick and easy retrieval of any data archived within the DAAC's Data Archive and Distributed System (DADS). Over the 15-year life of EOS project, an estimated several Peta-bytes (10^15) of data will be permanently stored. Accessing that amount of information is a formidable task that will require innovative approaches. As a precursor of the full EOS system, the GSFC DAAC with a few Tera-bytes of storage, has implemented a prototype of a constraint-based task and resource scheduler to improve the performance of the DADS.

This Honeywell Task and Resource Scheduler (HTRS), developed by Honeywell Technology Center in cooperation with the Information Science and Technology Branch/935, the Code X Operations Technology Program, and the GSFC DAAC, makes better use of limited resources, prevents backlog of data, and provides information about resource bottlenecks and performance characteristics. The prototype which is developed concurrently with the GSFC Version 0 (V0) DADS, models DADS activities such as ingestion and distribution with priority, precedence, resource requirements (disks and network bandwidth) and temporal constraints. HTRS supports schedule updates, insertions, and retrieval of task information via an Application Program Interface (API). The prototype has demonstrated with a few examples, the substantial advantages of using HTRS over scheduling algorithms such a First In First Out (FIFO) queue. The kernel scheduling engine for HTRS, called Kronos, has been successfully applied to several other domains such as space shuttle mission scheduling, demand flow manufacturing, and avionics communications scheduling.

Introduction

The main objective of the Code X Operations Technology Program (X-OTP) is to provide advanced techniques in order to reduce NASA's operational costs by focusing on reusable software technology. In addition to numerous technologies such as electronic documentation, database management systems, system diagnosis, and data analysis tools to name a few, one of the successful areas of X-OTP has been the application of planning and scheduling technologies to missions operations throughout NASA. In cooperation with the GSFC DAAC and Honeywell Technology Center, X-OTP has initiated a program to apply scheduling technology to various areas within the EOSDIS. In addition to providing local management for this project, the Information Science and Technology Branch, which is part of the GSFC supercomputer facility or the Space Data and Computing Division, has been providing its Intelligent Information Fusion System (IIFS) as a modular, end-to-end, advanced prototype system for testing these new technologies. Free from many requirements of operational systems, this prototype system is being used to guide several of the technological extensions for this scheduling project.

This paper presents the first phase of this project by discussing the capabilities of the Honeywell Task and Resource Scheduler (HTRS) as they apply to the scheduling of operations in large mass storage systems. The GSFC DAAC architecture is briefly introduced and the main DADS functions are described as they relate to mass storage issues. The approach used to solve scheduling issues and the specific DADS scheduler requirements is then explained. The architecture of the scheduler, its domain model, and an application Program Interface (API) to communicate with the scheduler is also presented. In particular, the paper describes the application of a constraint-based scheduling to a mass storage system for the management of data ingestion, dissemination over a network environment, and distribution of datasets copied to tapes. Due to the large number of daily tasks and their dependencies, the slow seek time on tapes, and deadlines which must be met, a First In First Out (FIFO) scheduling algorithm, as well as other queuing approaches, is not adequate. HTRS increases the throughput of the various DAAC activities by making efficient use of the DAAC's computer resources. The HTRS is an adaptive, dynamic scheduler capable of modeling numerous system resources such as disk storage, robotic devices, processors, memory, and network bandwidth. HTRS handles resource contention, prevents deadlocks, and makes decisions based on a set of defined policies. By modeling database operations as tasks with priority, precedence, duration, resource requirements and temporal constraints, HTRS efficiently supports schedule updates, insertions, and retrieval of task information.

General Scheduling issues

Given many of the misconceptions about scheduling, this section will cover a brief summary of the common definitions and topics surrounding data processing scheduling for those readers not familiar with the terminology in the following sections. In general, most non-real-time Operating Systems (OS) handle task management by assuming that tasks operate independently of each other and that execution characteristics cannot be accurately determined a priority. Hence, simple queuing methods dominate this category, often providing sub optimal solutions (e.g., FIFO scheduling). The Unix OS, in fact, was designed for general purpose workstations where little is ever known about task characteristics.

Improvements to these approaches require an analysis of the operating characteristics of typical tasks, such as determining if tasks have priority or deadlines, arrive periodically or arbitrarily, operate in a uniprocessor/multiprocessor or heterogeneous/homogeneous environments (i.e., different or same processors), exhibit predictable resource properties, and organize into a data flow graph (i.e., tasks whose execution precedes and passes data to others) . These improvements are constrained by the operational requirements such as trying to minimize task completion time, demanding that most or all tasks meet their deadlines (i.e., soft real-time or hard real-time), and allowing tasks to be preemptable or nonpreemptable to name a few.

Based on these characteristics, the *scheduling problem* can be defined as given a set of tasks *T* associated with a subset *C* of the aforementioned constraints, determine the execution sequence, if possible, that best satisfies *C*. Two basic types of scheduling approaches exist: *static* (or deterministic) and *dynamic* (or non-deterministic). Static schedulers create schedules off-line after all task information has been collected while dynamic schedulers determine schedules on-line during continuous data collection. Any static scheduler is optimal only if it produces schedules that satisfy *C* whenever any other scheduler satisfies *C*. A dynamic scheduler, however, produces an optimal schedule if it always produces a feasible schedule when a static scheduler with complete information can create one. Obviously, static schedulers are always sub-optimal when the collected information changes before the schedule is produced, regardless of which scheduling algorithm is used. While dynamic schedulers suffer less from this problem, they incur a lot of overhead due to the cost of constantly collecting information. For this reason, many schedulers utilize a *hybrid* approach where scheduling is done off-line while adjustments are made on-line.

Related to this issue, schedulers are also classified as *adaptive* or *non adaptive* depending on whether the environment provides feedback to the scheduler. That is, the scheduler's control mechanism changes in response to system histories or trends. Dynamic schedulers are almost always adaptive. Of course, the type of information collected determines how well the scheduler performs. Estimates of task duration can be based upon best, average, or worst-case estimates depending on optimism or pessimism. Other statistics can include modeling the average number of tasks arriving for particular times, hot spots for resource usage, inter-task communications costs, etc. Determining how refined the statistics model always depends on the performance requirements, which often change to meet evolving bureaucratic policies.

Institutional requirements usually determine the control architecture of the scheduling environment. For example, *Centralized* systems such as shared memory models are those where processors essentially operate in a group where inter processor communication costs are minimal with respect to processor execution costs. By contrast, decentralized systems such as wide area networks (e.g., the DAAC's) imply high inter processor communication costs. Often times, centralized or distributed scheduling means that the computing environment is centralized or decentralized. This should not be confused with the much harder problem of using multiple schedulers to control a distributed environment versus using a centralized scheduler to control a distributed environment.

Adding to the confusion, the power of scheduling algorithms is often overestimated. For example, scheduling tasks with arbitrary precedence between tasks for multiprocessors is proven to be NP-hard (i.e., essentially known to take an exponential number of steps as a function of the number of tasks) with only unit execution time, regardless of whether tasks are preemptive or non preemptive. Hence, because most of the non NP-hard algorithms (i.e., polynomial) are too restrictive, schedulers realistically must utilize heuristic approaches (i.e., smart guessing) while searching for feasible schedules. This involves the construction of a function, often called an *objective function*, that encapsulates a notion of "goodness" for evaluating one proposed schedule versus another during the search through the space of possible schedules. Objective functions can be explicitly represented by numeric formulae for simple comparison or they can be implicitly captured in the scheduling policy algorithm. Regardless, the objective function or scheduling policy algorithm should be flexible enough to change as the institution governing the processing environment modifies its notion of a good schedule. For instance, an institution may want to guarantee that all or most task deadlines are met one day while on another day, it may wish to minimize completion time of tasks.

Scheduling issues with mass storage systems

Today's mass storage systems are critical resources that usually must operate in a complex and changing institutional environment. These institutions must process large volumes of data while providing efficient and reliable service to a large number of users, who typically request resources at unpredictable times. Satisfaction of these users is critical in order to justify the enormous investment required to run these large institutions. Also, proper decision making about which resources is absolutely necessary for controlling the high cost of these computing environment. Scheduling technologies allow institutions to provide services according to reasonable user deadlines while providing information about which resources are bottlenecks that must be alleviated with the purchase of appropriate hardware.

These characteristics are certainly true of the EOS architecture and, in particular, are being evaluated in the context of the GSFC DAAC -- a system that is intended as an operational testbed for EOS. Although nowhere near the size of the final EOS system, the GSFC DAAC is estimated to process 250 SeaWiFs orders per day, corresponding to 40 GB of data. In addition, 20 GB of non-SeaWiFS products are expected to be ordered each day while 26 GB of new data will be ingested. Due to the large number requests and the large volume of data to process, manually generating feasible schedules will not be possible. Moreover, using the FIFO queue approach is not an acceptable solution because it does not make the best use of the resources available (e.g., tape drive, disk space), it doesn't have the ability to guarantee that most deadlines are met, and it provides little information about resource bottlenecks.

Of particular note, each request (e.g., distribution) has several tasks that must be scheduled individually. For example, to process an order for data requested on an 8mm tape, the tasks may consist of retrieving the data from near-line devices, transfer the files to a staging area, and then copy the files to an 8mm tape. Overall thousands of tasks with predecessor and successor tasks, each with specific needs for resources, must be scheduled and tasks cannot be treated equally. Requests for data to be sent over the network may be given a higher priority than data requested on tapes. Hence, the schedule should reflect these DAAC policies that determine, for example, deadlines and priorities.

Given that many of the operations involve transfer from one medium to another, proper migration from slower devices such as mass storage to faster devices is necessary to minimize average access times. That is, anticipation of requests should cause data to be moved into faster devices "just in time" for the request to be satisfied. This, of course, is similar to the notion of "locality of reference" in any memory hierarchy where the storage management system prefetches blocks of data in anticipation of future access to those blocks. Only here, the prefetching is also based on models of the external task environment in addition to load characteristics of the tasks, implying that a powerful scheduler can reduce access times by performing tasks just in time for delivery.

Because of the need to quickly anticipate trends in the external and internal processing environment, another challenge is to have a schedule that can be dynamically and quickly updated when new orders are received, when some of the resources become unavailable during a period of time, or when a given resource must be restricted to improve the overall performance of the system. For an example of this last category, tests conducted at the DAAC have shown that the number of concurrent NFS actions between the Unitree cache and the distribution staging area had to be limited to six or seven in order to achieve an acceptable throughput. Thus, the scheduler should model resources such as NFS resources to anticipate the proper localities of reference.

While the anticipation of many actions can be automated, many external events to the system requires that a human operator be present to make adjustments. That is, in any symbiotic production environment involving both computers and humans, tools must exist to help operators identify the status of the orders and their respective tasks as they relate to policies provided by management. For example, an estimated completion time for each task could be presented to the operator in order for the operator to communicate information back to high priority users.

These estimates should be based not only on the approximate duration of each task but on the availability of the resources. In fact the estimates for each task could be complex, however, the actual and the estimated times can be continuously compared so that better statistical approaches can be introduced. After conducting several tests simulating next year's workload, it became clear that scheduling was very important.

GSFC DAAC architecture

The GSFC DAAC has been developed to support existing and pre-EOS Earth science datasets, facilitate scientific research, and test EOSDIS operational concepts. Its design is based on the EOSDIS functional requirements and the requirements generated by specific Science projects such as Sea-viewing Wide Field-of-view Sensor (SeaWiFS).

GSFC DAAC has three main components illustrated in Fig 1. The Product Generation System (PGS) receives low-level data products and generates higher level data products. The Data Archive and Distribution System (DADS) role is to archive all new data products and to distribute over the network or on a variety of physical media, data ordered by researchers. The Information Management System (IMS) is a data base of the data holdings which can be searched, browsed by researchers to help them identify and order data of interests.

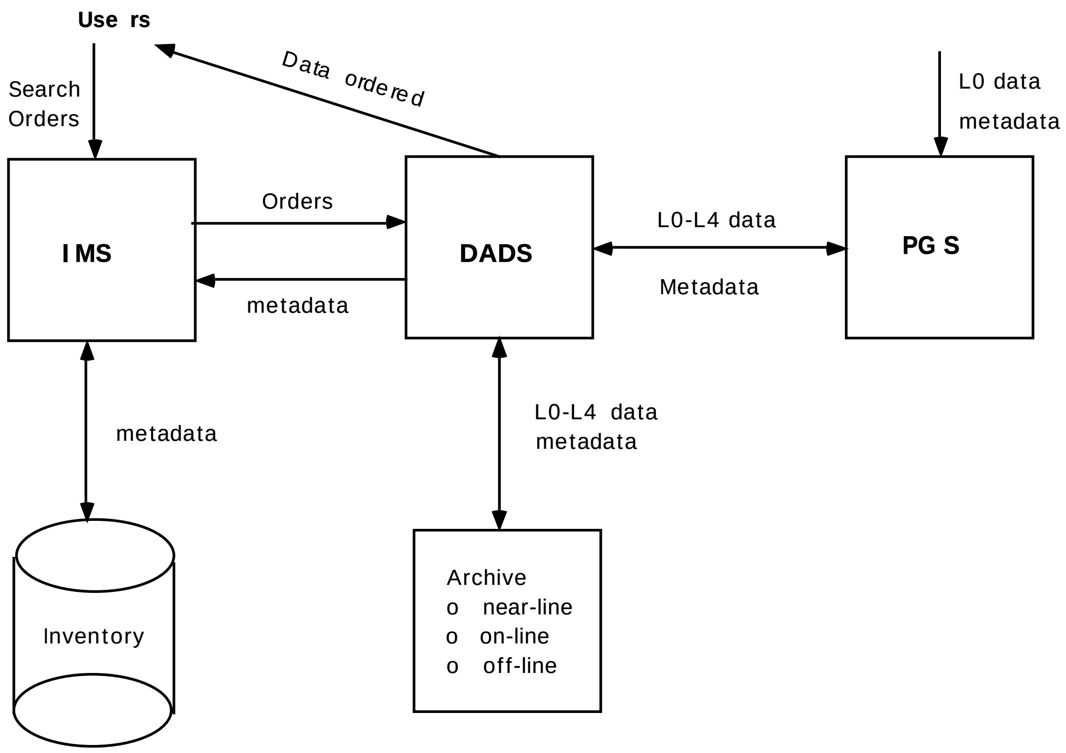


Figure 1 GSFC DAAC components

Although smaller than the overall facilities in the Space Data and Computing Division, the GSFC DAAC has currently 731 GB of data archive but this number is expected to increase to about 18 TeraBytes by FY97 [1]. To satisfy these requirements the GSFC DAAC has the following hardware architecture.

• The IMS system with its Oracle data base runs on a dedicated SGI 4D/440 VGX.

• The DADS software and the Hierarchical Storage Management (HSM) system Unitree to automate the migration and the stage operations, run on a SGI 4D/440 S. Data are archived either on a Cygnet 1803 jukebox (1179 MB) with 2 ATG WORM drives or an RSS-600 Metrum Automated Tape Library (ATL) (8700 MB) with 4 RSP 2150 VHS drives. The SGI 4D/440 S was too limited in terms of I/O bandwidth and ports. A SGI challenger L (DADS2) has been acquired to handle all the distribution copies on tapes. There are currently nine 8 mm drives, four 4 mm drives, and two 9 track drives attached to the EOSDADS2 machine.

• There is a future plan to build a Backup system that will run on an SGI Challenge S. Its function will be to keep a second copy of all data ingested at the DAAC.

• The PGS is composed of 3 SGI workstations. Two additional workstations are used to do Q/A on the data.

• The DAAC's distributed environment includes two ethernet Local Area Networks, and an FDDI network.

GSFC V0 DADS functions

The three main functions of the DADS are archive, distribution, and data management. The archive function consists of accepting data products from outside the system, extracting metadata, validating files, and updating the database. The distribution function retrieves files from archives, stages them to a distribution staging area, reformats the data if necessary (e.g., tar is the normal format for orders), and writes the data to tapes or to the FTP staging disk. The DADS management handles the schedules, tracks DADS activities, and allocates/deallocates resources.

DADS V0 Scheduler

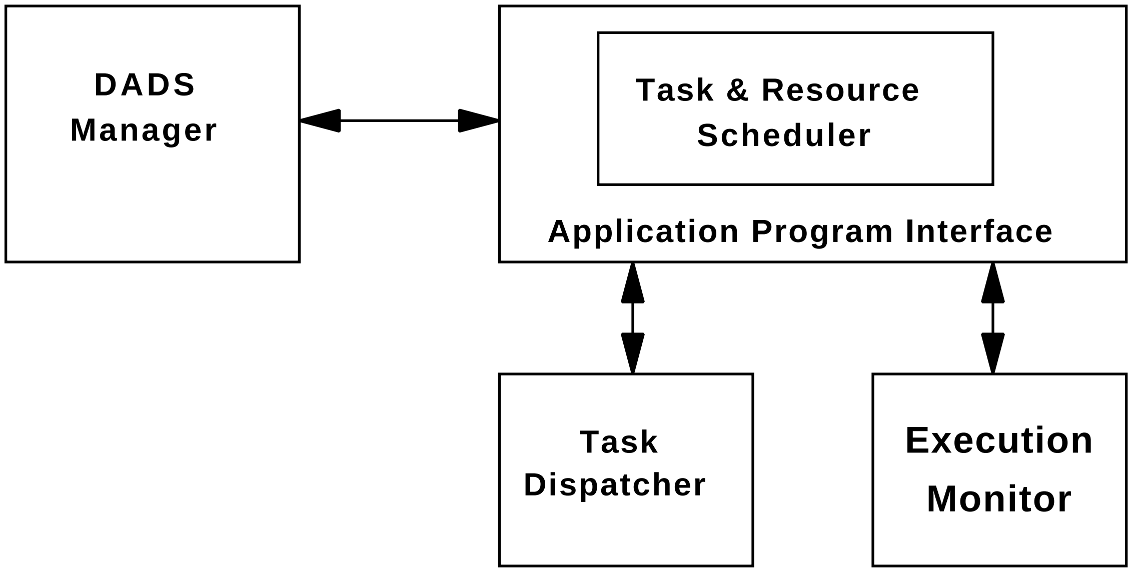


Figure 2. The HTRS Scheduler's Architectural Environment

The DADS V0 Scheduler is responsible for scheduling actions and resources to ingest data from a network to buffer disks, transfer buffered or cached data to a mass storage archive, and to retrieve archived data upon request. The scheduler was developed concurrently with the design and implementation of the GSFC V0 DADS. Consequently, the architecture and interfaces must tolerate changes as the system design evolved. The current version of the DADS software uses a multi-level priority queue algorithm, as a baseline system, to schedule its activities, however, there are plans to integrate the Honeywell Task and Resource Scheduler to the DADS for performance improvements. The baseline architectural environment of the HTRS scheduler is depicted in Figure 2. This environment continues to evolve, but its conceptual and functional characteristics remain stable, so many system changes can be accommodated in the Application Program Interface (API).

The DADS Manager submits scheduling requests, handles errors, and retrieves schedule information. The Task Dispatcher periodically queries the scheduler for a list of upcoming scheduled activities to be executed. The execution monitor notifies the scheduler of events that affect the schedule.

Approach

Constraint envelope scheduling technology offers an attractive, proven method of meeting the scheduling needs of data archiving and distribution. This technology, embodied in Honeywell's enhanced implementation of the Time Map Manager (TMM), supports the concept of a Temporal Constraint Graph (TCG) which can be used to represent multiple projections of future system behavior, thereby providing rapid rescheduling with minimal disruption in the presence of schedule uncertainty or changing policy situations.

The DADS V0 Scheduler is an application of the Kronos scheduling engine that is built on top of TMM and designed to be adaptive and dynamic. Kronos has been successfully applied to domains such as space shuttle mission scheduling, demand flow manufacturing, and avionics communications scheduling. It has handled scheduling problems involving 20,000 tasks and 140,000 constraints, with interactive response times for schedule modification on the order of a few seconds on a SPARC10.

Scheduler Requirements

Detailed scheduler requirements were initially established for the DADS application, then extended and adapted to encompass the scheduling needs of other NASA programs based upon feedback from the IIFS. The following paragraphs summarize requirements at a high level. They confirm the need to be appropriate to the application domain, to be compatible with the target system, and to provide responsive performance reliably.

<u>DomainAppropriate</u> - Commercial scheduling tools sacrifice domain relevance to extend their range of applicability, and hence their marketability. They often lack the capacity to efficiently handle the precise scheduling needs of large, complex applications such as those presented by EOS. In order to select or define a scheduling tool that is domain appropriate, application-driven requirements must be established. Whenever possible, these requirements should be based on multiple examples of domain operations and scheduling functions using realistic data sets. They must include a quantitative demonstration so that capacity and performance goals can be met simultaneously.

Since the GSFC V0 DADS is being developed concurrently with the prototype scheduler, we were careful to maintain a high degree of generality in the scheduler implementation. By first building a core scheduling capability derived from our Kronos scheduling engine, and then extending that capability through specialization, we were able to meet the specific needs of DADS while providing a scheduling tool that can easily be applied to similar problem domains in EOS.

Stated as a system requirement, the scheduling core domain model must be compatible with objects and functions required by the target application. Further, its customization capabilities must support accurate modeling of every schedule and relevant aspect of the domain. Care should be taken to ensure that this model reflects the intended scheduling policies and procedures of the application, and not the characteristics of analytical models used to project system performance.

Details of the scheduling core domain model are described in the Domain Model section. For the prototype scheduler, subclasses were created to capture application specific attributes and relationships. These attributes may be used to carry system data through the schedule or to support performance monitoring and analysis.

By creating persistent requirement and persistent resource profile classes as subclasses of the requirement class and resource profile class using an object-oriented model, respectively, we were able to provide the necessary scheduler functionality with a minimum of disruption. Persistent requirements have the option of specifying that they begin, use, or end with their associated activity. This allows the resource allocation to be open ended if desired.

To be effective, any tool must be functionally complete and be able to solve the problems for which it is applied. A scheduler must enforce structural constraints (i.e., predecessor-successor and parent-child relationships), temporal constraints (e.g., earliest start or deadline), and resource availability constraints while carrying out the desired scheduling and resource allocation policies in an automated fashion. In the prototype scheduler, policies are currently encoded as functions and a domain-specific algorithm (as described in the Scheduling Policy section.

We plan to eventually excise policy details from the scheduler by defining syntax for policy specification. One possible solution would be to utilize a rule- or knowledge-based approach to represent the numerous institutional policies. The major advantage of this approach is that rules (e.g., if-then statements) can naturally represent situations when a particular schedule is "good". Likewise, a dependence on rules allows for the incorporation of several knowledge acquisition tools. In the IIFS, for example, the Advice Taker/Inquirer (AT/I) allows users to enter and modify expertise in lucid forms such as natural language. Should a policy change, a tool like the AT/I could be used to quickly modify the appropriate rule governing that policy.

<u>Compatible</u> - The scheduling tool described here is designed be integrated as a functional component into the target application system. It cannot dictate requirements to that system, rather, it must adapt to the physical and logical demands of the encompassing system. The scheduler must execute on available hardware running the specified operating system. It must be able to communicate with asynchronous functional modules of application system via standard interprocess communication system facilities.

The scheduler must also be linguistically compatible with the surrounding system. It must be able to interpret and respond appropriately to requests for service and information. The prototype scheduler meets this requirement in several ways. The scheduler includes an API customized to the syntactic and semantic needs of the DADS modules with which it interacts. An underlying set of basic API functions facilitates this customization.

The scheduler supports the notion of activity state. The exact states and legal state transitions are defined for the application. In DADS, activities can be scheduled, committed, dispatched, executing, complete, or failed. Additional states and even additional state dimensions can be added as the need arises.

<u>Responsive</u> - Performance is often a critical requirement, but it is frequently overlooked in scheduling. There are often assumptions that scheduling will be performed once in an initial scheduling effort and that the resulting schedule will satisfactorily describe the actual execution of activities. This view is seldom correct and certainly incorrect in data processing scheduling.

We have segregated the total problem into two phases, planning (what to do) and scheduling (when to do it). In other words, planners are allowed to substitute similar tasks in order to find a set of tasks that have feasible schedules. Schedulers per se are given a fixed set of tasks and only leeway in the selection of resources and start/end times. Unlike the DADS and for that matter, the rest of EOSDIS, the IIFS utilizes the planning/scheduling approach to generate browse products in lieu of standard products when computational constraints are too great for standard product generation. For example, computationally cheaper, yet less accurate tasks can be intelligently substituted for expensive tasks in order to better meet deadlines or minimize resources. This situation occurs often in image processing where resampling routines can reduce the image size. The browse products can be used by users to decide whether to initiate a standing order request. In this way, just as it was one of the first systems to suggest object-oriented programming and databases for the EOSDIS domain, the IIFS has allowed for the testing of risky, new ideas that may not yet have been considered within the operational DAACs.

Nevertheless, by making this distinction, we have not only, made each aspect more manageable, but we can tailor the functionality and performance of each component's implementation to the needs of the application. Planning typically occurs before scheduling, though replanning may become necessary. In the GSFC V0 DADS application, there is a small set of functions to be performed (e.g., ingestion, distribution). These can possibly be pre-planned in advance and described to the scheduler as tasks (with subtasks).

The scheduler must, on demand and in near real time, fit each new instance of a task into the current schedule in accordance with task priorities and deadlines while ensuring that necessary resources will be available. As actual events occur in the execution of the scheduler, it must rapidly reschedule to reflect the impact of the event. It must provide data to support graphic presentation of the current schedule, and even allow operator manipulation of tasks.

<u>Reliable</u> - The fault tolerance approach employed by the target application must be supported by the scheduler. In the GSFC V0 DADS this translates to requirements for redundant archiving of schedule information and rapid recovery of the schedule after a failure. The prototype scheduler does not fully include these features at present. However, basic mechanisms needed for reload are present in the script processor described in the Prototype Environment section. Also, previous schedulers based on the Kronos engine have included schedule storage and reload capabilities.

Prototype Environment

The DADS V0 Scheduler is being developed concurrently with the GSFC V0 DADS. Consequently, a stand-alone environment was needed in which to test and demonstrate scheduler functionality. The operation of components external to the scheduler was simulated via a script processor as shown in Figure 3. The script processor is controlled from a demonstration Graphical User Interface (GUI) that displays schedule activities and resource utilization profiles. Snapshots of the demonstration GUI screen may be seen in Figures 6 and 7. The GUI supports selection and execution of an event script which the script processor translates into API commands that it sends to the scheduler.

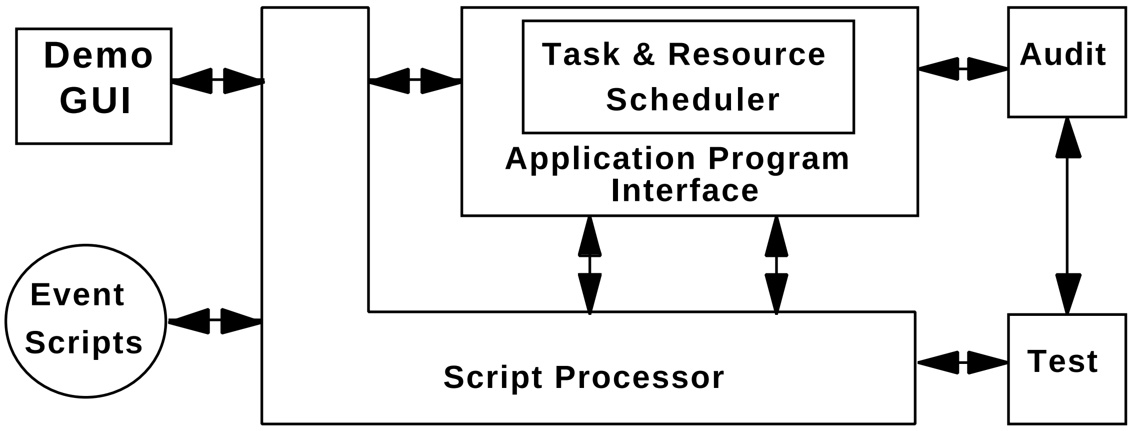


Figure 3. The Prototype System Architecture

A typical script initializes the scheduler by describing the resources available for scheduling, commands the creation of activities to be scheduled, and simulates execution events such as completion of execution. The script also notifies the GUI as objects to be displayed are created.

Graphical presentation of scheduler operation is visually convincing, but it is inconvenient for testing and benchmarking purposes. Recently, auditing and test functions were added to facilitate execution and validation of complex event scripts. The test function automates the execution of scripts and the invocation of the audit function, which checks the schedule for consistency and correctness.

Architecture of the Scheduler

The internal architecture of the scheduler is depicted in Figure 4. The base layer supplies basic temporal reasoning capability. This includes objects such as uncertain time-points and constraints, and functions for updating and querying the temporal knowledge base.

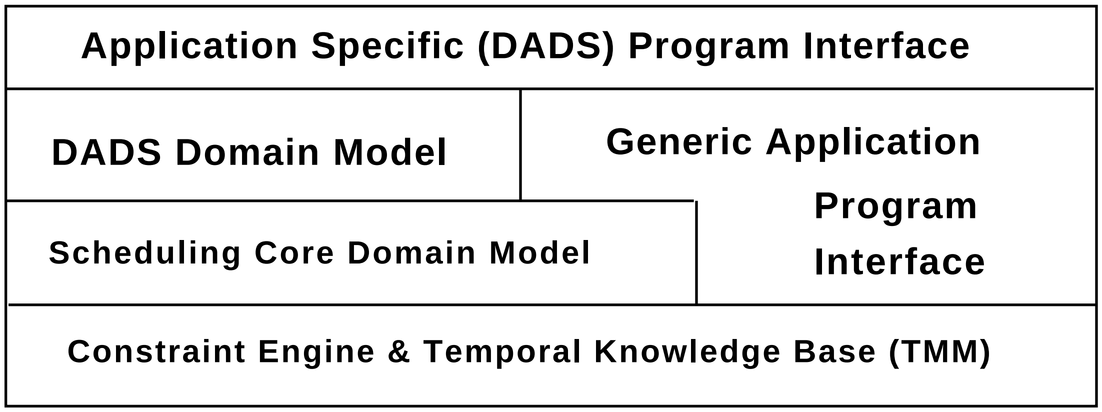

Application Specific (DADS) Program Interface
DADS Domain Model
Generic Application Program Interface
Scheduling Core Domain Model
Constraint Engine & Temporal Knowledge Base (TMM)

Figure 4. The Architecture of the Scheduler

The Scheduling Core Domain Model supplies the basic objects and functions needed for scheduling and resource management. Combined with the Generic API, these layers form a core scheduling capability that can be applied to various scheduling domains. In the DADS V0 Scheduler implementation, the base domain model was extended through specialization and extension to provide appropriate domain-specific capabilities, shown in the figure as the DADS Domain Model and the DADS API.

Domain Model

Key object classes of the scheduling core domain model include resources, requirements, activities and hierarchical activities. These are shown in Figure 5. along with related objects classes of the DADS scheduling domain model.

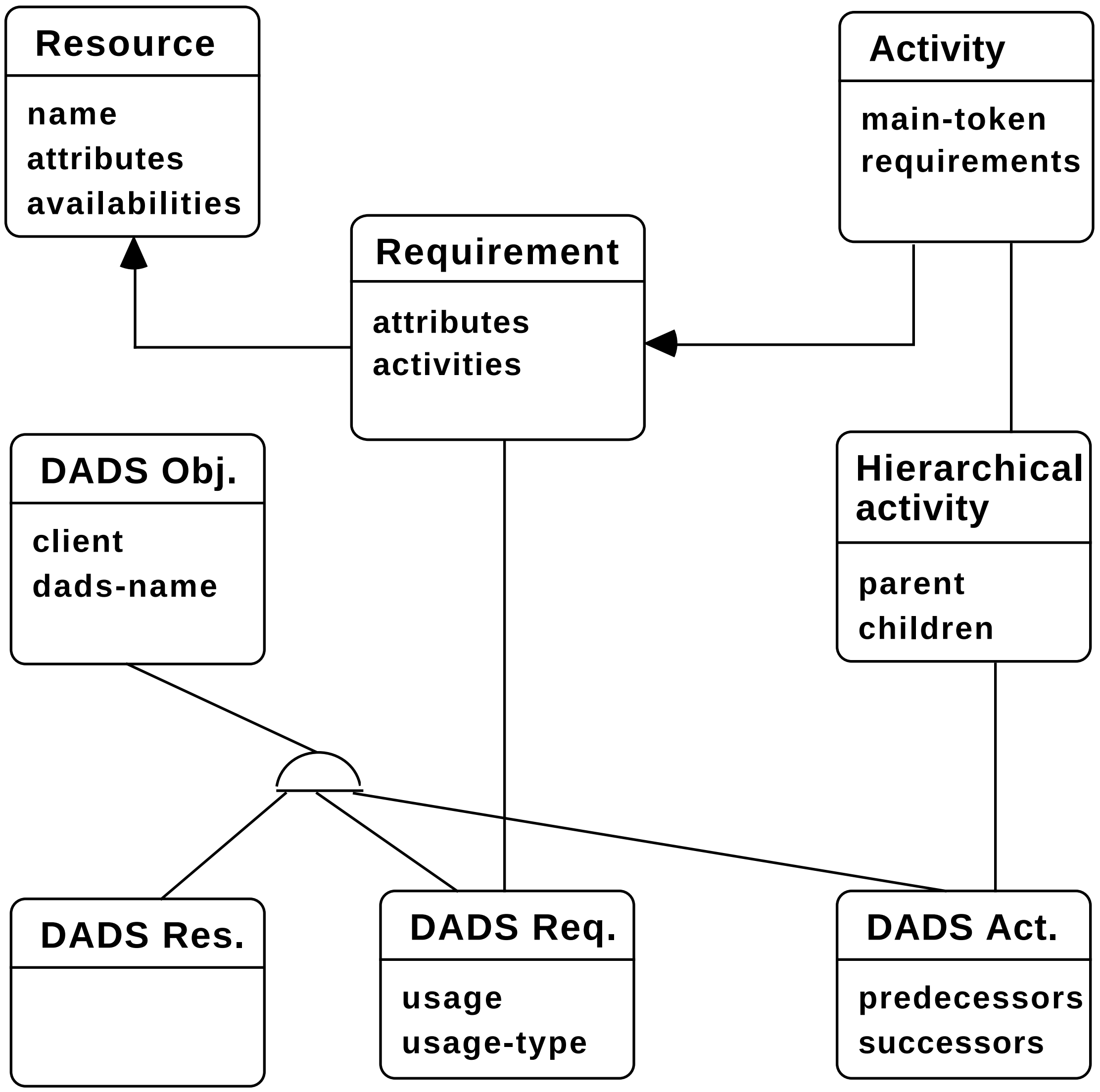


Figure 5. Key DADS Scheduling Object Classes

An activity represents an action to be scheduled. Each activity has an associated main-token which defines its end points in time and its possible duration range. An activity may be linked to multiple resource requirements. These abstractly define attributes that must be satisfied by the resources allocated to the activity. A subclass of the activity allows hierarchical activity structures to be defined. These were used in the DADS scheduler to implement tasks with component subtasks.

As an example, in the DADS application, a data ingestion task will have several subtasks. The data buffering subtask requires access to the FDDI network and a specific amount of space on one of the data ingestion magnetic disks. A subsequent archiving subtask requires access to the data on buffer disk and space on the UNITREE archive magnetic disk.

The core resource classes allow resources to be conceptually organized into pools using a hierarchical name structure (which permits wildcards) and using a list of resource attributes. Each resource has an associated availability that defines the maximum quantity of that resource and its temporal range.

Specialization of the core object classes extend the hierarchy to include characteristics of the target domain. In the DADS scheduler these specializations share a common parent class, the DADS object, which defines attributes every DADS activity, resource requirement, or resource must have. Only the client and dads-name attributes are shown in the figure.

Application Program Interface (API)

The Application Program Interface was specified formally by documenting data content (i.e. fields and forms) of the primary information components (i.e. tasks, subtasks, resources, etc.) exchanged between the scheduler and DADS subsystems. For each command, the documentation details the participants in the exchange utilizing the command, the conditions under which the command occurs, the intent (semantics) of the command, and the scheduler's response to the command under both normal and error conditions.

The following command categories describe the functions of the scheduler visible via the API. The categories have been intentionally kept rather abstract and high level here. Not all command categories have been fully implemented in the prototype scheduler.

Definition/Instantiation - Inform the scheduler of the existence of scheduling entities such as activities (i.e. tasks and subtasks), resources, and abstract resource utilization requirements. These commands do not cause scheduling to occur.

Modification - Change the specifics of information known to the scheduler. This category encompasses only changes to the scheduling problem (e.g. relaxation of a deadline). It does not include notification of real-world execution events.

Interrogation/Retrieval - Retrieve schedule and resource allocation information from the scheduler. This information is based on the scheduler's model of the problem space, its record of past events, and its projection of future events including resource utilization.

Scheduling/Rescheduling - Compute a new schedule with resource allocations. Commands in this category may be invoked indirectly by commands in the Update/Synchronization category. Update/Synchronization - Inform the scheduler of the occurrence of real-world events (e.g. activity execution completion) which may affect the schedule. This category also includes commands for the transfer of responsibility for an activity from the scheduler to another subsystem (e.g., an execution monitor or dispatcher).

Notification - Inform another subsystem that a problem (or potential problem) has been detected by the scheduler.

Communication Handshaking - Provide positive acknowledgment of information transfer.

Fault-Tolerance/Recovery - Support for information backup and recovery from failures.

Scheduling Policy

The operation of the scheduler is controlled by scheduling policies. These are currently captured in domain-specific, hard-wired algorithms for resource assignment and activity scheduling.

The baseline resource assignment and scheduling algorithm is:

```
For each activity to be scheduled:

    If the activity has component activities,
        Schedule each of its component activities (i.e., apply this algorithm recursively).

    If the activity is scheduleable,
        For each resource requirement of this activity:

            - If a satisfactory resource is available for use without causing it to be
                oversubscribed,
                assign that resource to meet the requirement.
                        Availability implies that the resource is part of the resource pool
                        specified in the resource requirement and has the attributes specified
                        in the resource requirement.

            -If no satisfactory resource is available,
                apply the following stratagems in sequential order,
                    using the possible resources until one of them successfully eliminates
                    the oversubscription:

                * Constrain the order of activities involved in the oversubscription:
                    Individually before the activity, or
                    Individually after the activity, or
                    Collectively before the activity, or
                    Collectively after the activity.

                * Relax the deadline of activities involved in the oversubscription and
                    constrain the order of activities (as above)

                * Constrain the order of parent activities of the activities involved in the
                    oversubscription (as above)

                * Report failure [and Exit]
```

```
If the activity is still scheduleable
and all component activities of this activity have been scheduled,
    Mark the activity scheduled.
```

Then update:

The schedule's temporal knowledge base,

The time bounds of all changed resource utilization profiles.

One thing to notice in the algorithm is the emergence of situations to control the scheduling. For example, take the situation where the scheduler should schedule activities if their resources won't possibly be oversubscribed. This was a DADS requirement that other domains need not be constrained to have. But, in its current incarnation, it is hard-wired into the algorithm. Should this change, then the algorithm must be modified, increasing scheduler maintenance costs. As new policies are incorporated, these costs will be untenable. Hence, changing over to other approaches such as rule-bases, will constrain costs and allow for evolvability.

Scheduling Example

The operation of the prototype scheduler is revealed in Figures 6 and 7. In this simple example, seven data ingestion tasks have been scheduled. Each task contains four subtasks (not visible) and is represented in the display as a horizontal timeline. The solid portion of the timeline indicates the earliest possible execution of the task. The dashed portion of the timeline indicates scheduling flexibility between earliest execution and the task's completion deadline.

At the bottom of the display, the resource utilization of a selected resource is shown. The black profile line indicates expected resource utilization if all tasks execute as early as possible. The gray profile line indicates possible resource utilization.

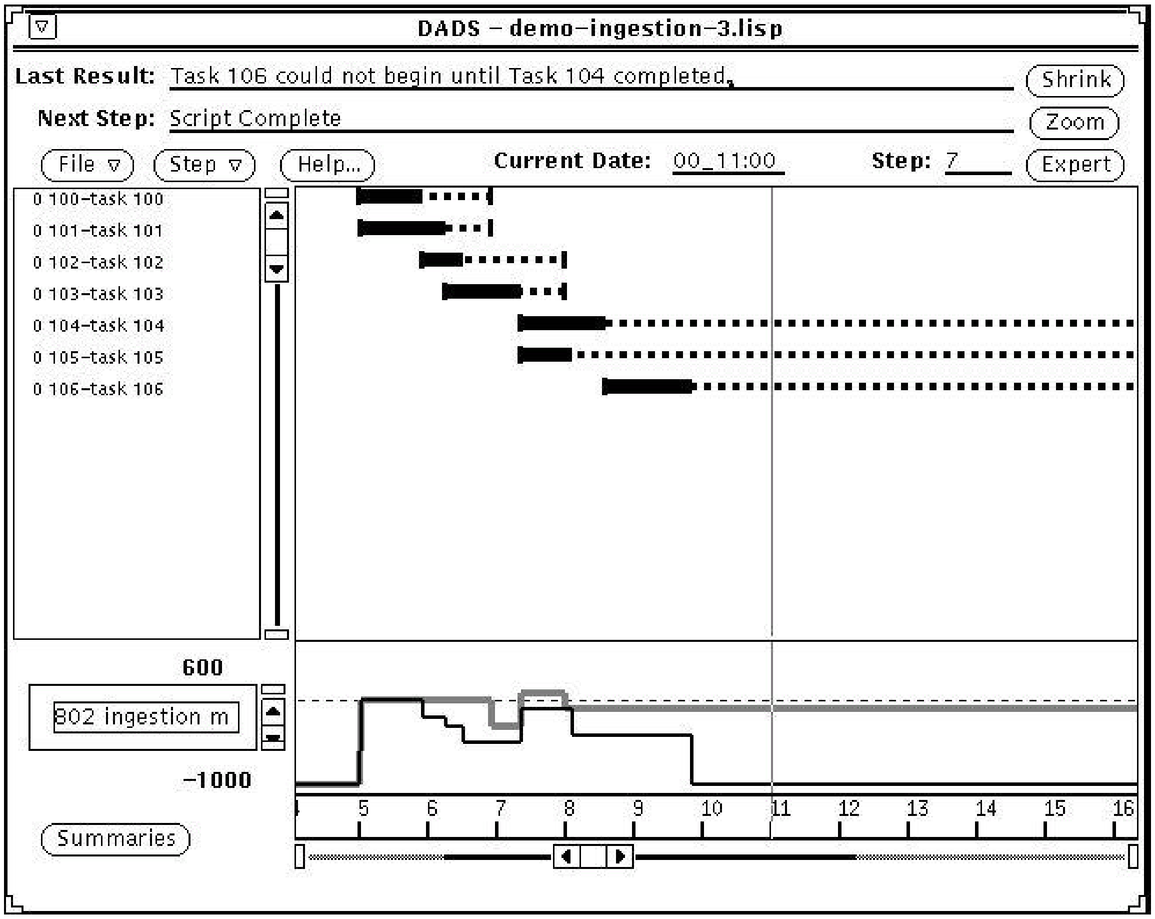


Figure 6. Simulation of the Baseline DADS V0 Scheduling Approach (FIFO Queue)

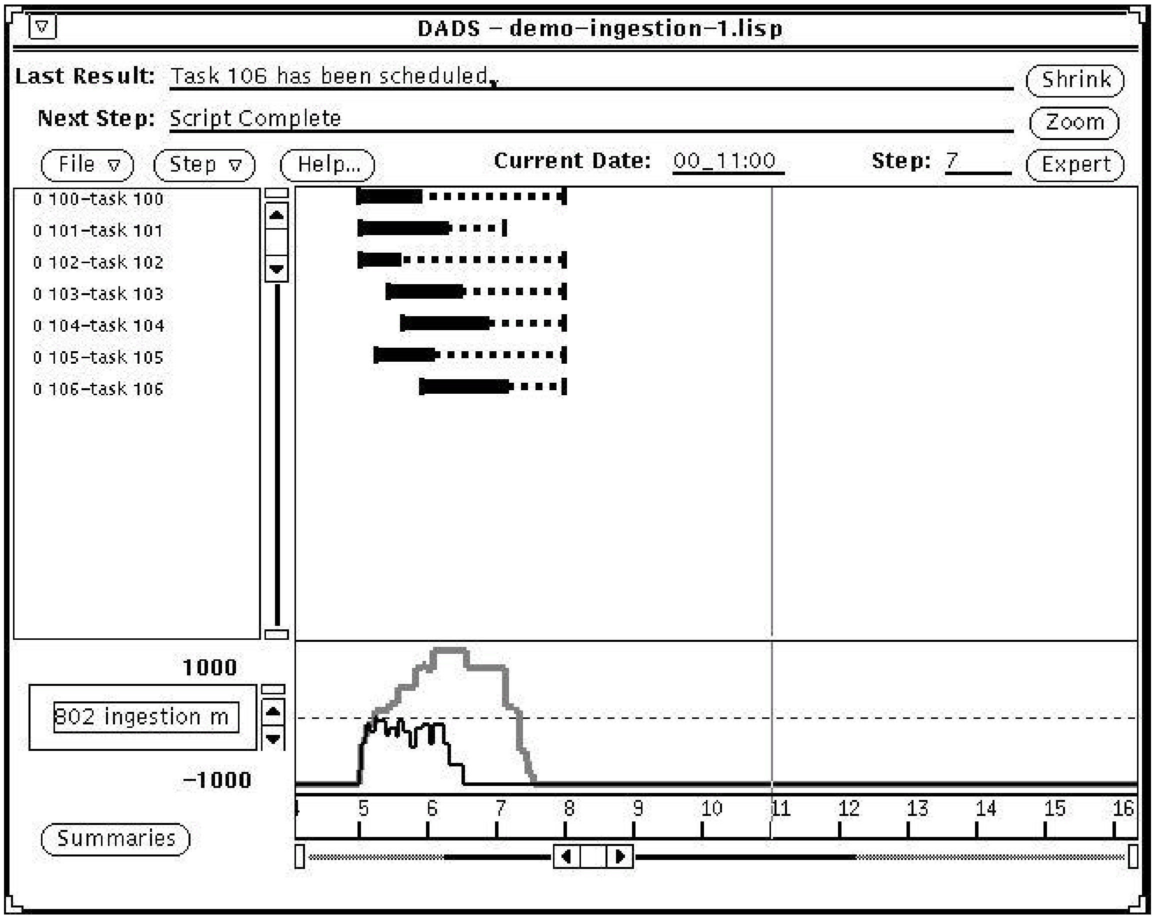
In this figure, the tasks have been configured to simulate the baseline DADS V0 scheduling approach. In the baseline approach, all resources needed by the component subtasks are allocated to the task. Then tasks are then scheduled using a First-In First-Out (FIFO) Queue. Additional constraints were added to enforce this queuing.

Parallel task execution occurs until resource utilization reaches 100%. The subsequent tasks must wait for ongoing tasks to complete.

The deadlines of tasks 104 through 106 could not be met. These deadlines were removed, causing the dashed portion of the timeline of these tasks to extend to infinity. Task 106 actually started AFTER it's completion deadline.

The resulting resource utilization is very inefficient. It has large regions of rather low utilization an is spread over almost five hours.

Figure 7. Shows the same tasks scheduled using the full power of the HTRS scheduler.


DADS – demo–ingestion–1.lisp

Last Result: Task 106 has been scheduled.

Next Step: Script Complete

Current Date: 00_11:00

Step: 7

Figure 7. Efficient Automated Scheduling and Resource Allocation

Resource requirements were specified with respect to individual subtasks, and FIFO queuing constraints were not imposed. The resulting schedule is clearly superior.

All tasks were scheduled within their deadlines. The scheduler has optimized resource utilization (as evidenced by the compact profile). And the entire group of tasks requires only slightly more than one hour. This leaves time for an opportunistic system to initiate several tasks such as trend analysis routines, maintenance tasks for adding new system components, or quicker service. Moreover, using less powerful hardware could lengthen the overall duration, but still beat the performance of the FIFO approach.

This simple example shows that a constraint-based task and resource schedule can provide substantial improvements in system performance over simple queuing schemes. What may not be evident are the benefits it provides through rapid rescheduling in response to unexpected events (e.g., resource failures), and through the automation of complex scheduling policies. These performance improvements could translate into cost savings through the use of less expensive hardware.

Conclusion

The HTRS scheduler prototype has been successfully developed for the GSFC DAAC. Special attention should be paid in the identification of the scheduler requirements and the DADS domain model. Even though the prototype is still in its initial phase and it has not yet been integrated into the DADS, the effort has been very informative. In particular, it was demonstrated using realistic examples of DADS activities that a FIFO queue algorithm can be extremely inefficient under certain conditions, and can drastically reduce the overall performance of the DADS. The scheduler cannot only make better use of limited resources and prevent a backlog of data, but it can also provide valuable information about resource bottlenecks and performance characteristics. The next challenge will be to integrate HTRS in the DADS , monitor its performance, and evaluate its benefits when running in a real operational system such as the GSFC DAAC.

In the context of mass storage systems, scheduling can help ensure that timely service is provided to users who expect a lot from these expensive computing facilities. Likewise, scheduling can be used as a simulation tool to predict the performance from adding particular hardware. By utilizing the same scheduling environment, these simulations can be based on real information from the operating environment and can provide quality information for decision makers. In some cases, decision makers may avoid costly hardware purchases by tweaking the scheduling policy algorithm. Hence, the scheduling policy algorithm must be flexible enough to be modified quickly in order to contain software maintenance costs. Certainly, the use of scheduling will provide <u>better</u> service for users, <u>faster</u> processing throughput, and <u>cheape</u>r costs.

In fact, many of the scheduling issues presented here have arisen throughout numerous NASA applications. Over the years, the X-OTP has provided scheduling expertise to various projects by focusing on rapid prototyping of new technologies for mitigation of risk, technology transfer through continued software development from prototypes, and reduction in cost through software reuse of generic tools. By working with Honeywell Technology Center, X-OTP is further reducing software development costs by providing difficult requirements to companies, who can then apply developed techniques to other commercial domains such as aviation communications scheduling. By helping companies expand into new markets, NASA, without incurring high maintenance costs, increases the likelihood that dual-use commercial software will survive over the lifetime of lengthy projects such as EOS.

X-OTP, on the other hand, requires feedback from projects whose requirements push the state-of-the-art. As intended, the GSFC DAAC, through Hughes-STX, has provided this feedback before the larger EOSDIS has gone into operational use. The GSFC DAAC, however, is an operational system that cannot be interrupted with technology that is too risky. Hence, prototypes such as the IIFS can quickly test very risky technology in an end-to-end framework without adversely affecting operations. For one thing, the IIFS was the first system at GSFC to suggest the use of object-oriented databases for the EOS domain. Likewise, the IIFS was the first system to suggest the use of neural networks for classifying remote sensing data -- a technique that is now widely accepted in remote sensing circles. And, finally, the use of this particular scheduling software was based upon a NASA internal R&D project (i.e., Directors Discretionary Fund) entitled "Near real-time generation of Browse Products" and incorporated into the IIFS. Because of the development of the IIFS and the close proximity to NASA projects, the Information Science and Technology Branch has provided in-house expertise regarding emerging technologies such as these. Moreover, in addition to applied research, the branch has developed one of the DAACs operational quality assurance routines for the TOVS pathfinder data sets. Likewise, the Space Data and Computing Division, for which the Information Science and Technology Branch is a part of, is currently GSFC's only supercomputing facility with an extremely large mass storage system (over 20 Terabytes); this enables feedback regarding technology integration of large, expensive systems. All in all, elaborate collaborations such as these will obviously be required to evolve one of the most ambitious engineering and information system projects, or namely, the Earth Observing System.

Acknowledgments

Primary funding for this scheduling project has been provided by the Code X - Operations Technology Program under Dr. Mel Montemerlo, NASA-HDQ. Funding has also been provided by the GSFC DAAC as well as Honeywell Technology Center. Special thanks are extended to members of the Information Science and Technology Branch, who have provided invaluable feedback through the use of the Intelligent Information Fusion System.

1. L. Bodden, P. Pease, JJ. Bedet, W. Rosen: Goddard Space Flight Center Version 0 Distributed Active Archive Center. In Third Conference on Mass Storage Systems and Technologies. NASA CP-3262, 1993, pp. 447-453.

2. H. El-Rewini, T. Lewis, H. Ali, Task Scheduling in Parallel and Distributed Systems. PTR Prentice Hall, Englewood Cliffs, New Jersey, co. 1994