Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

Scheduler, a distributed batch controller

D. Nguyen, R. Edgar, T. Gaetz, D. Jerius, M. Tibbetts

Harvard-Smithsonian Center for Astrophysics

Abstract. As part of our effort to support the calibration of the Chandra X-ray Observatory's High Resolution Mirror Assembly (HRMA), we often have to run many computationally intensive simulation programs. In order to maximize our efficiency we have written a suite of programs to distribute batch jobs by exploiting idle workstations during non-work hours. We discuss the system architecture of the Scheduler programs which were written to facilitate the tedious task of submitting a large number of batch jobs. As a demonstration of the capabilities of the Scheduler program, we describe its use in the analysis of ground calibration data. We also compare the efficiency of the Scheduler to the Sun's Grid Engine.

1. Introduction

Personal workstations are often idle for extended period of time, especially during non-work hours. We often have to run a large number of ray-traces, up to 1 million jobs, so Scheduler was written to exploit the computational power of idle workstations during non-work hours. Our batch jobs may last only a few tens of seconds so using a backend SQL database will yield unacceptable long transaction times, instead the batch queues are now kept in memory to reduce the per-job scheduling costs. Scheduler is a distributed batch job controller using the client/server paradigm for batch jobs submission, Figure 1 shows the architecture of the Scheduler system. The program has the following features: A centralized batch job submission and job control across heterogeneous systems. Workstations are assigned to classes, and can be members of several classes. The user can specify the class of workstations on which to run a batch-job. The user can dynamically add and remove workstations to the pool of workstations. The user can kill and suspend jobs. Each workstation has its own non-work hours schedule. The owner of each workstation can override its current non-work hours schedule.

2. The Server

Scheduler was implemented using the Remote Method Invocation (RMI) capabilities of Java. RMI was chosen because the server and the clients are Java applications and RMI allows applications to call object methods located remotely

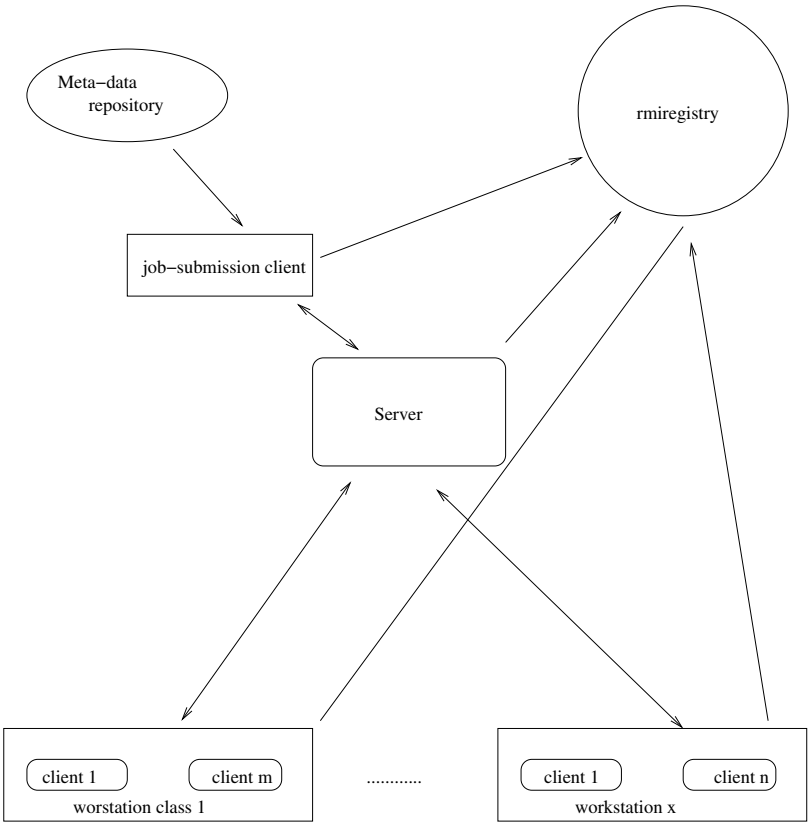

Figure 1. Architecture of the Scheduler.

without having to deal with the low level networking details. The architecture and runtime dynamics of the system can be summarized as follows:

The remote object registry, rmiregistry, must first be started on a specified port on the current host. The remote object registry is a naming service used by RMI server to bind objects to names. Once an object has been registered, the clients can look up remote objects and make remote method invocations. The server can be started once the remote object registry is up and running. The server makes its methods available for remote invocation by binding it to a name in the RMI registry. The server is the centralized processor to carry out the following tasks: The server is the central repository for batch jobs submission. The server dispenses the appropriate batch job upon request from idled clients. The server relays commands for the clients to suspend or terminate processing.

3. The Clients

The client obtains a reference to the server remote object by looking up the name which the server had register itself as with the rmiregistry. Once the client has the server remote object, it can invoke methods on it as if it were a local object. The tasks of the clients are: The client must verify that it is

not within the black-out period before requesting work. If the client determines that it is within the black-out period, it sleeps until the next available time zone. While sleeping, the client must periodically check to make sure that it has not been sent a signal to be terminated. The client continues to request work from the server until the list of batch jobs is exhausted.

There are numerous programs and scripts to monitor the progess of the batch jobs and to summarize the workload distributions based on the log file generated by the server.

4. Disadvantages of Scheduler

We do not have the software development resources required to implement a system secure enough to run with superuser privileges. In order to avoid security problems, Scheduler runs as a non-privileged user, this means that a separate instance of the server and the clients must be started for each user in order that jobs be run with correct access permissions to disk space, etc. Scheduler does not implement a mechanism to arbitrate resources between users; each job "network" is ignorant of any other. Thus, cooperation between users is required to prevent the client machines from being overburdened. We have a small group of users, so this has not been a problem. The drawbacks with the current design of the current Scheduler are: It is a single user cooperative multi tasking system. The batch job queue is non-persistent. All jobs must complete before the batch queue can be changed. The Java Virtual Machine (JVM) can be a memory intensive application. It implements primitive first in first out (FIFO) job and batch queues.

5. Application

As a demonstration of the Scheduler's capabilities, we compare the ground based calibration measurements to the SAOsac model (Jerius 2003) to determine the pointing of Chandra's High Resolution Mirror Assembly(HRMA) during testing at the X-Ray Calibration Facility (XRCF).

The HRMA was mounted to multiple stages which could raise or lower the front and rear of the mirrors or move them side to side. Changes in the position of the source or inaccuracies in the HRMA stage positions could lead to uncertainty in the absolute pointing of the HRMA. As the shape of the line response function (LRF) changes as a function of HRMA pointing, it is possible to use the LRF measurements to determine the pointing. Measuring the HRMA pointing during the XRCF testing proved to be a nontrivial task[1].

We determine the pointing of the HRMA by optimizing the fit between the data and simulated LRF measurements, letting the simulations explore the parameter space consisting of the HRMA yaw and pitch and the aperture XY position in the focal plane. Presumably, the best fit pointing will indicate the absolute pointing of the HRMA at the XRCF. In order to get a fair sampling of the mirror models, each HRMA pointing was simulated with 100 different realizations of the model, see Figures 2.

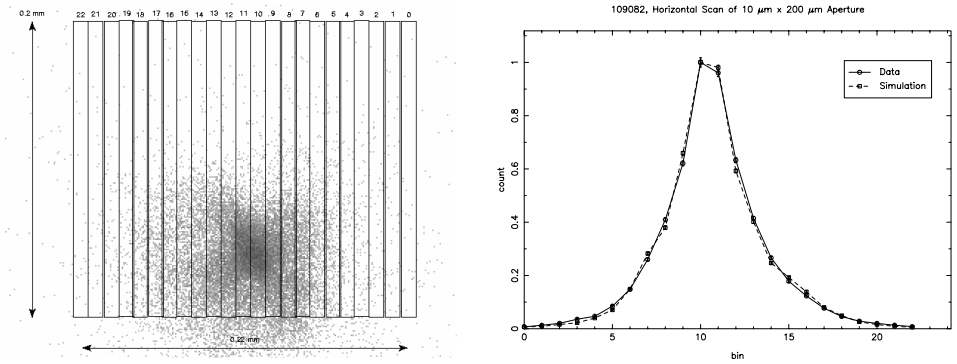

Figure 2. Simulated focal plane image of the horizontal scan with apertures overlaid. Line response function for the simulation.

6. Performance Analysis

The Scheduler is a fairly efficient program, the following table is a comparison of the execution time of 100 realizations of raytraces using Scheduler versus the Sun's Grid Engine.

Table 1. Scheduler versus Sun's Grid Engine.

Manager	Average Time	Standard Deviation
Scheduler	00:01:57.70	2.003 secs
Sun's Grid Engine	00:03:06.09	55.827 secs

Keeping the list of batch jobs in memory reduces the Scheduler's overhead. The following table compares the time of Scheduler versus the Sun's Grid Engine when the list of batch jobs contains jobs which are short in duration.

Table 2. Scheduler versus Sun's Grid Engine.

Num	Scheduler	Sun's Grid Engine
100	00:00:17.61	00:00:26.00
1000	00:01:18.81	00:04:50.00
10000	00:12:01.90	01:13:03.00

Acknowledgments. This work is supported by the Chandra X-ray Center under NASA contract NAS8-39073.

References

Edgar, R. XRCF Phase 1 Testing: Analysis Results, Appendix D: HRMA Pointing at XRCF

Jerius, D. et al. The Role of Modeling in the Calibration of Chandra's Optics, Proc. SPIE, Vol. 5165, X-Ray and Gamma-Ray Instrumentation for Astronomy XIII