Astronomical Data Analysis Software and Systems VI
ASP Conference Series, Vol. 125, 1997
Gareth Hunt and H. E. Payne, eds.

A Database-driven Cache Model for the DADS Optical Disk Archive

T. Comeau and V. Park

Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218, USA

Abstract. The Data Archive and Distribution System (DADS) manages the Hubble Data Archive (HDA), a WORM Optical Disk Archive which contains over two terabytes of Hubble Space Telescope data. One fortunate side effect of retrievals from the HDA is that all retrieval requests are permanently logged in database tables. Queries against these tables provide a complete history of requests serviced by DADS. We describe a system which uses the database request logs as input to a flexible cache model. The model permits changes to the size of the cache, replacement strategy, and preloading the cache. We also discuss possible cache sizes and replacement strategies, and their effect on DADS performance in the post-SM97 system.

1. Introduction

Throughput of the Hubble Data Archive (HDA) is an important bottleneck for overall Data Archive and Distribution System (DADS) performance. If a significant fraction of retrieve requests could be serviced from a cache, this bottleneck would be bypassed.

Additionally, retrieve requests currently interfere with the process of ingesting new datasets. This is a potentially serious problem in the post Servicing Mission 97 (SM97) era, when ingest rates are close to the limit of this same archive bottleneck.

All retrieval requests are permanently logged in two database tables. The *requests* table contains information about each DADS retrieval request; the *request_files* table contains information about each file retrieved to satisfy that request.

Queries which select from the *requests* table by *req_date*, and from the *request_files* table by request number (where *req_reqnum* = *rqf_reqnum*) and sequence number (where *rqf_reqnum* specifies a dataset for that request) provide a complete history of requests serviced by DADS.

Likewise, the table *archive_data_set_all* includes all the datasets ingested into DADS. Selecting from this table in *ads_data_receipt_time* order provides a complete history of DADS ingests. Additional information about ingests, including the program for which science data were taken, is kept in the *proposal* table.

2. Model Components

The Cache Model consists of three major components. First, a small set of C routines parse input parameters and call database stored procedures. Second, stored procedures implement the cache management scheme. Finally, two tables implement the cache: a single cache control record with global information, and a cache index table which describes the current contents of the cache.

Keeping the cache index and the routines which manage the index in the database makes modifying the cache behavior simpler. Since a stored procedure is simply a collection of SQL statements with flow-of-control language, writing such a procedure is at least as simple as equivalent C code. Additionally, stored procedure queries are precompiled to improve performance. Stored procedures can easily be modified and reloaded. Keeping the cache index in a database allows queries to monitor and evaluate the effectiveness of the cache, both during runs and postmortem. It also makes the cache index persistent: later runs can use all or part of the cache left over from previous runs.

The model provides the basis for an operational cache index since it uses the same data as DADS uses to obtain datasets from optical disks.

3. Requests Caching

When examining datasets for possible inclusion in a cache, we quickly identified three broad types of requests:

- **DADS Operations requests**, submitted by DADS Operations staff, principally for data verification. Since one primary goal of these requests is to verify that the data are correct on optical disk, the request must be serviced directly from optical disk, and the use of a cache is inappropriate. DADS Operations retrievals are excluded from the model.
- **Auto PI requests**, generated by DADS itself, using database information about recently ingested data, to automatically send recently completed observations to the Principal Investigator of an HST Science Program. Since these requests by definition use recently ingested data, they should be effectively serviced by a cache of recently ingested science datasets.
- **Everybody else**, including internal users retrieving recently ingested data to monitor instrument or observatory health, or to respond to user questions, internal users retrieving non-science data (including science classes for non-science proposals) or older data for a variety of reasons, and internal and external users retrieving older data.

The datasets can also be divided into three broad categories:

- **Calibration data**, used to calibrate science data.
- **Science and science-related data**, the "interesting" datasets to most users. Interesting datasets can be further divided into science programs and non-science programs. The latter include Calibration, Engineering, Orbital Verification, Science Verification, and Early Release Observation programs.

- **Nonscience data**, including everything from intermediate products of the OPUS pipeline, to copies of previous releases of the DADS software.

Calibration data are by far the most popular data in the archive, used by internal, external, and Auto PI requests. Nonscience data is rarely retrieved.

4. Cache Performance Scenarios

Eight cache scenarios were executed against the data for August, 1996:

			Hit		Datasets		Total	
Case	Req's	Hits	Rate	Unhit	1 Hit	Multi-hit	Datasets	MB
1	19052	5338	.28	0	10711	2997	19052	95287
2	19052	8598	.45	7095	10717	2997	20809	137466
3	19052	8578	.45	1339	10616	2993	14948	99350
4	4807	4066	.85	1928	3095	126	5149	18994
5	4807	4066	.85	569	3095	126	3790	12868
6	14245	2055	.14	0	7867	1534	9401	73311
7	14245	3163	.22	2065	7867	1534	11466	79413
8	19052	6976	.37	509	3072	695	4276	29996

Case One simulates an unpreloaded cache with all retrieved datasets inserted into a cache of unlimited size. This is the unfiltered retrieval case, with most datasets retrieved only once. Calibration data are, as expected, the most hit datasets, with the interesting datasets accounting for all but a tiny fraction of the remaining requests.

Case Two duplicates Case One with the addition of preloading new datasets as they arrive, adding an additional unretrieved 42 GB to the cache in order to save one disk access for the datasets which will be retrieved by Auto PI.

Case Three preloads only interesting and calibration datasets. This eliminates from the cache the non-science classes, leaving only the very popular calibration data, and the interesting classes. Only 20 fewer hits are obtained by eliminating uninteresting data from the cache, and the 42 GB of uninteresting data are never inserted.

Case Four examines the option of an Auto PI only cache, preloading the same interesting datasets, but including only Auto PI retrievals. This is the best scenario on a hit rate basis, but there are still datasets being preloaded that are uninteresting, despite being in science classes.

Case Five is also an Auto PI only cache, but only science *proposals* are preloaded. This is the "pure" Auto PI cache. This is both the smallest cache, and the best hit rate cache. Of the 15 percent of the datasets unretrieved, none is more than four days old.

Case Six examines the same cache insertion rule with non Auto PI retrievals. Not suprisingly, 84 percent of these datasets were retrieved exactly once. Also, data retrieved more than once were usually re-retrieved within 14 days of being loaded into the cache. Since this scenario did not preload, this means that if a dataset was not re-retrieved within two weeks, it probably never would be.

Case Seven duplicates Case Six with preloading of new, interesting, science proposal generated datasets. There are additional hits against preloaded data, and these are the same data that would generally be hit by Auto PI: Newly ingested science datasets.

Based on the age of likely-to-be-hit datasets, we selected a cache size of 30 GB, of which roughly half would be Auto PI data. Some of the Auto PI data will also be retrieved by other (Institute internal) users.

Case Eight suggests that preloading datasets for Auto PI also preloads a few datasets for other users. (These are presumably internal users monitoring instrument health or assisting PIs with interpreting their data.) There are still a significant number of one hit datasets older than four weeks, which suggests that a 30 GB cache is more than adequate for a month of retrievals, but that a larger cache would not be much more effective in servicing archive requests.

5. Conclusions

Of the 20,809 datasets distributed in the model scenarios, 436 were calibration datasets used by both Auto PI and other archive users. These datasets total just 1663 megabytes, but account for 2466, or 13 percent of all requests. A cache preloaded with substantially all of the current calibration datasets would give a significant boost to retrieval performance while consuming about ~8–10 GB. Auto PI always uses these datasets; Starview users may now get the appropriate calibration datasets automatically.

While the 37 percent hit rate obtained in the final scenario is disappointing, it is largely a function of how users make requests of DADS. Internal users, including Auto PI, retrieve the data while they are new. Other users request the same kinds of data, but the ages of those data vary, which makes them difficult to cache.

The two calibration classes and the five science related classes account for all but a tiny fraction of retrieval requests. Additionally, science *proposals* account for the vast majority of retrieval requests. Thus, a retrieval cache can be limited to these seven classes for Science Proposals without significantly reducing the hit rate.

A modest cache (~30 GB) preloaded with new (~2–4 weeks old) datasets, and calibration datasets, would service essentially all Auto PI requests. Such a cache would also serve the many internal users that request roughly the same data. A caching scheme that serves older data, however, is probably not possible.

Acknowledgments. I wish to thank Lisa Gardner (STScI), who was extremely helpful in setting up the cache database and assisting in query design.