THE CHANDRA AUTOMATIC PROCESSING/ARCHIVE INTERFACE

Sreelatha Subramanian and David Plummer

Harvard-Smithsonian Center for Astrophysics, 60 Garden St. MS-81, Cambridge, MA 02138, Email: latha@head-cfa.harvard.edu

Abstract. The Chandra Automatic Data Processing System (AP) requires quick access to previously generated data. Potential inefficiencies are avoided by introducing a layer between the pipelines and the archive. This archive interface layer includes an archive request queue, a data archiving server (darch), and an archive "cache". The design and functional operation of each of these components is the focus of this paper.

1. Introduction

Pipelines in the Chandra AP system often use data output by other pipelines, which implies that the system must wait for both the ingest and retrieval of the data from the archive. The archive interface layer de-couples data processing from data product ingest, thereby eliminating the possible inefficiency of these scenarios. The implementation of a caching system, consisting of an inactive cache system and virtual files, also improves the performance of the system and allows for greater flexibility.

2. Cache

The first step in improving the AP system is the implementation of a cache. In theory, the cache operates as a typical cache - data is retrieved from a slow-access storage facility and is stored to a higher-speed retrieval location, with the expectation that it will be used again.

In our implementation, the cache is actually a directory where the data files reside, as shown in Figure 1. While a pipeline is processing, it will store its data products to the cache directory. These data products can then be stored to the archive at any time, separating data product ingestion from processing. When another pipeline requires those data products, the cache is checked first. If the data products are not in the cache, then the pipeline retrieves the files from the archive database. This method is especially significant when processing data for the first time since the system can access all the data products from cache, avoiding the archive database. If reprocessing is required, data will be retrieved from the archive if it is no longer in cache.

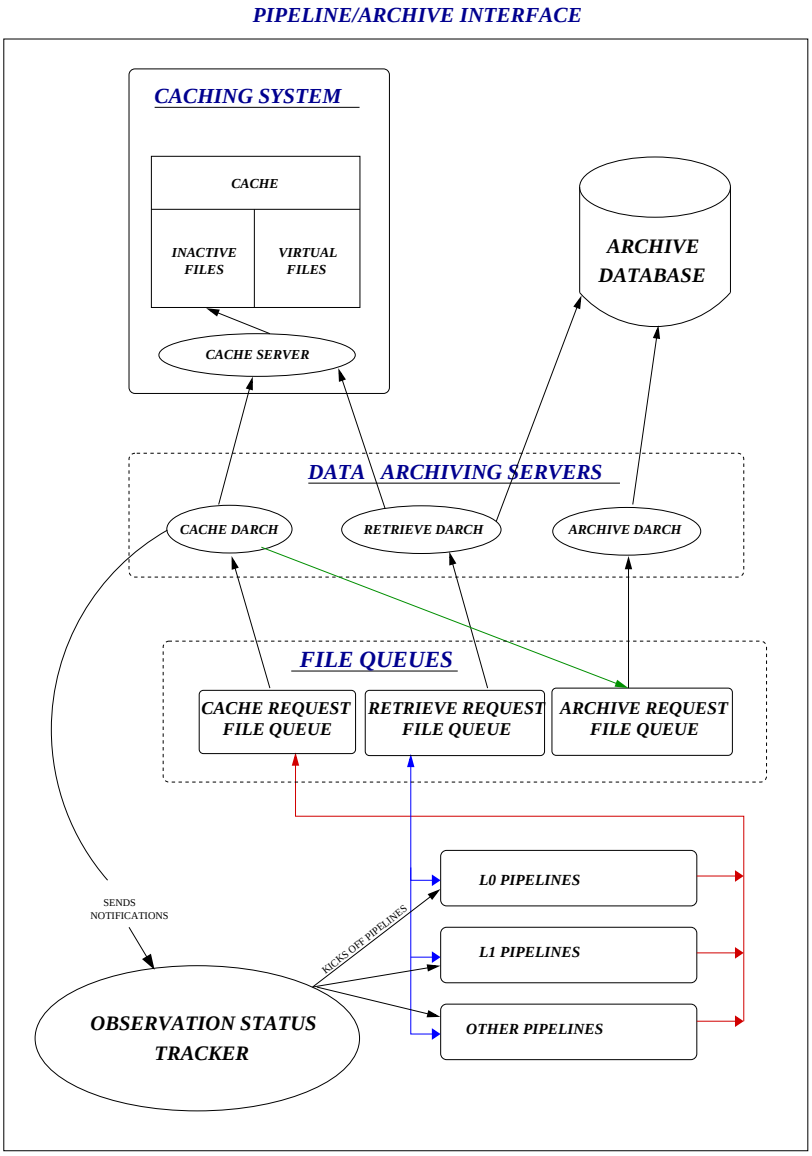

Figure 1. AP/Archive Interface

2.1. Inactive Cache

The implementation of a cache introduces the issue of memory usage. The cache server loads all the files in the cache directory into its memory, which allows for quick search and retrieval of the data. However, as the size of the cache increases, the cache server requires more memory. To reduce the number of files in cache, an inactive cache directory structure is created. The cache server does not load files that exist in the inactive directory into its memory. It stores a few pieces of information about the file along with the filename, in order to facilitate retrieval. The implementation of the inactive cache eliminates the issue of memory usage, while maintaining the usefulness of the cache.

2.2. Virtual Cache

A significant improvement to the cache system is made through the implementation of virtual files. In our system, virtual files are simply files that do not exist in cache, but whose filename and metafile still exist. With this information, cache can be queried to retrieve a file based on time, data product id or other information. Cache returns the filenames of the files it found, and also returns an error code which indicates that the files are virtual. The AP system, recognizing that the retrieved files are virtual, can then use the filename to retrieve the file from the archive database. This modification is significant because retrieving a file from the archive using the filename as the key is more efficient than retrieving using another key (such as data product id, or time). The other important factor is that the implementation of virtual files allows us to search for file(s) by using cache specific tag information. Tag information, which is stored in the metafile, is not entered into the archive. Therefore it was previously not possible to retrieve files from the archive using tag information. With virtual files, however, this becomes possible. A search by tag function searches cache, using the tag. As this information is in the metafile, the function will return a corresponding filename, which can then be used to retrieve the file from the archive.

3. File Queues

Another important modification to the AP System is the implementation of file queues. File queues were implemented to eliminate the time a pipeline must wait for its data products to be ingested into the archive database. Instead of waiting for a file to be ingested, the ingest request is sent to a file queue. The request is then picked up at a later time by the data archiving server, or darch, and processed. The implementation of file queues is simply a file which contains all the requests. The pipeline will not need to wait for the ingest since it has already stored the file to cache. Therefore, the ingesting of the file does not need to occur simultaneously, and the file queue allows for another process to handle the requests at any time.

4. Conclusion

The archive interface layer was not part of the original design. However, the necessity for it was identified during the processing of ground calibration data of the system before launch. This allowed us to implement and integrate the archive interface layer into the AP system, to avoid possible bottlenecks during the mission.

5. Acknowledgments

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

References

Plummer, D. , 2000, this volume, [P1-32]