S12-82
83114

Analysis of the Access Patterns at GSFC Distributed Active Archive Center

Theodore Johnson[1]
Dept. of CISE, University of Florida
Gainesville, FL 32611
AT&T Research
Murray Hill, NJ 07974
ted@cis.ufl.edu

Jean-Jacques Bedet
Hughes STX Corporation
7701 Greenbelt Rd., Suite 400
Greenbelt, MD 20770
bedet@daac.gsfc.nasa.gov
phone: 301-441-4285

Abstract

The Goddard Space Flight Center (GSFC) Distributed Active Archive Center (DAAC) has been operational for more than two years. Its mission is to support existing and pre-Earth Observing System (EOS) Earth science datasets, facilitate the scientific research, and test Earth Observing System Data and Information System (EOSDIS) concepts. Over 550,000 files and documents have been archived, and more than six Terabytes have been distributed to the scientific community.

Information about user request and file access patterns, and their impact on system loading, is needed to optimize current operations and to plan for future archives (i.e., EOS-AM1). To facilitate the management of daily activities, the GSFC DAAC has developed a data base system to track correspondence, requests, ingestion and distribution. In addition, several log files which record transactions on Unitree are maintained and periodically examined.

This study identifies some of the users' requests and file access patterns at the GSFC DAAC during 1995. The analysis is limited to the subset of orders for which the data files are under the control of the Hierarchical Storage Management (HSM) Unitree. For example, orders on pre-mastered CD-ROMs, which account for a substantial proportion of the total volume of data distributed, were excluded because they are not managed by Unitree. The results show that most of the data volume ordered was for two data products. The volume was also mostly made up of level 3 and 4 data and most of the volume was distributed on 8mm and 4 mm tapes. In addition, most of the volume ordered was for deliveries in North America although there was a significant world-wide use. There was a wide range of request sizes in terms of volume and number of files ordered. On an average 78.6 files were ordered per request. Using the data managed by Unitree, several caching algorithms have been evaluated for both hit rate and the overhead ("cost") associated with the movement of data from near-line devices to disks. The algorithm called LRU/2 bin was found to be the best for this workload, but the STbin algorithm also worked well.

[1] Theodore Johnson is supported by a grant from NASA, #10-77556.

Introduction

On-line scientific archives are playing an increasingly important role in data-intensive research. These archives hide the internal physical organization of the data and automatically migrate files between near-line and on-line (disk) devices, making data more easily accessible. However, building such a large-scale archive can be an expensive proposition and system resources need to be carefully managed. To date, there has been little published research that studies the performance of on-line scientific archives.

The EOSDIS archive is expected to have an ingest rate of Terabytes per day when fully operational. This data will be available on-line for browse, order, and distribution. Careful planning is needed to handle the very large volume of data, the very large number of files, and the expected high user demands. Many studies have been made to predict archive use, based on surveys of the expected users (for example, see the studies at [ESDIS]). However, little empirical evidence has been collected.

In this paper, we first study some of the user-request patterns and their impact on the on the overall system loading. Rather than examining all orders submitted at GSFC DAAC in 1995, a subset has been selected that has direct impact on the archive controlled by Unitree and the robotic devices. Not all data are stored under Unitree. For example, some data was received on 8-mm tape and never ingested into Unitree because of the substantial effort required. Orders for these tapes are usually simple tape copies and are conducted "off-line" and do not affect Unitree. To satisfy some of the GSFC DAAC users, a large farm of disks has been installed where data can be retrieved via anonymous ftp. These anonymous ftp orders, off-line orders, as well as CD-ROM requests are not used in the analysis.

There will also be presented an analysis of the GSFC DAAC Oracle data base that contains information on the orders and the files requested, as well as the Unitree log files that provides some insight on the mounts and stages operations. Based on the statistics gathered in the analyses, we discuss issues related to the user request and file access patterns, caching, clustering, migration, and system loading. Because the user access pattern is related in part to the data set accessed and because of rapidly changing technology, we do not claim that all future archives will have experiences similar to that of the GSFC DAAC. However, we feel that this study will provide insight into the nature of user access to on-line archives. We make comparisons to a previous study of the NSSDC NDADS archive (see [Jo95]) to point out similarities and differences.

Previous Work

Several studies on the reference patterns to mass storage systems have been published. Smith [Sm81d] analyzes file migration patterns in hierarchical storage management system. This analysis was used to design several HSM caching algorithms [Sm81c]. Lawrie, Randal, and Burton [LRB82] compare the performance of several file caching algorithms. Miller and Katz have made two studies on the I/O pattern of supercomputer applications. In [MK91], they find that much of the I/O activity in a supercomputer system is due to checkpointing, and thus is very bursty. They make the observation that

much of the data that is written is never subsequently read, or is only read once. In [MK93], they analyze file migration activity. They find a bursty reference pattern, both in system load and in references to a file. Additional studies have been made by Jensen and Reed [JR91], Strange [Str92], Arnold and Nelson [AN88], Ewing and Peskin [EP82], Henderson and Poston [HP89], Tarshish and Salmon [TS93], and by Thanhardt and Harano [TH88]. However, all of these studies apply to supercomputer environments, which can be expected to have access patterns different from those of a scientific archive.

The access patterns to the NASA National Space Science Data Center's on-line archive, NDADS, is studied in [Jo95]. However, there are many qualitative and quantitative differences between the NDADS archive and the GSFC Version 0 DAAC. We make comparisons whenever possible between results in this report and the results of [Jo95].

Access and Distribution Methods

The GSFC DAAC receives data from science projects such as Sea-viewing Wide Field-of-view Sensor (SeawiFS), Coastal Zone Color Scanner (CZCS), Total Ozone Mapping Spectrometer (TOMS), Pathfinder AVHRR (Advanced Very High Resolution Radiometer) Land (PAL), Tiros Operational Spectrometer (TOVS), DAO (Data Assimilation Office), and Upper Atmospheric Research Satellite (UARS). These data are stored in a Mountain Gate Automated tape library system (RSS-600) using VHS tapes and an 1803 Cygnet jukebox using 12" WORM optical media. By submitting requests to the HSM (Unitree), the data can be retrieved to disks and made available to users.

In 1995, there were several ways by which a user could order data. A Graphic User Interface (GUI) based on X-windows, allowed a user to browse, select and order products. A Character User Interface (ChUI) was also available for users with VT100 terminals, but this interface has more limited capabilities. Orders could also be submitted by calling the GSFC DAAC, or by sending a fax, letter or email.

Orders can be filled by sending the data copied to tape (8-mm, 4-mm, 9 track) to the users or by transferring the requested data to a distribution staging area where the user has several days to ftp the files to her own machine. A size limit has been placed on the volume of data that can be transferred via ftp requests because of limited resources (disk space and network). Because of the high overhead associated with the retrieval of small files from the tertiary storage system, some specific datasets are also kept on a large farm of disks (200 GB) and are accessible via anonymous ftp. Some data sets of high demand have also been pre-mastered on CD-ROM for easy and quick distribution. As this study is intended to illuminate the nature of on-line access to tertiary-storage based archives, we limit the set of requests that we analyze to only those that access Unitree. In particular, we exclude requests for pre-mastered CD-ROMS, and accesses to anonymous ftp data, and internally generated requests. Internally generated requests includes testing, and do not reflect the nature of on-line user access.

The GSFC Version 0 DAAC uses several avenues to distribute data. There are two types of distribution orders: random orders and standing orders. The standing orders are requests by users for some or all of the data as it is being received at the DAAC. The

random orders are interactive requests for data that has been previously archived and is available for order.

Log Files and Databases

To identify some of the characteristics of the requests, we examine the GSFC DAAC Oracle data base, which contains information on the files and orders. The requests studied are only for external users of the GSFC DAAC and therefore do not include the test requests processed in 1995. The time of the requests used in this study corresponds to the time when the orders were submitted to the GSFC DAAC and may not be correlated with the time for processing and shipping. To analyze some of the system load, we examine the Unitree log files.

Aggregated User Analysis

In this section, we analyze user requests by aggregating requests according to an interesting classification. In particular we are trying to identify some patterns in the orders in terms of volume, number of files per request, products, data level, interfaces selected, and geographical locations of users.

In Figure 1, we aggregate user requests by month and data product. A *data product* is one of the 7 categories: ACRIM, DAO, PAL, CZCS, TOMS, TOVS, and UARS. A given data product can have multiple *data sets* (e.g. one per data level). The analysis shows that most data volume is for one or two data products (DAO and Pathfinder AVHRR Land (PAL) data). Figure 1 also shows that the volume ordered varies greatly between months. The result is consistent with observations of NDADS. Different data products have different average file sizes, so reporting volume alone tends to bias our results towards favoring data products with large files. For a comparison, we plot the number of files ordered each month by data product, in Figure 2.

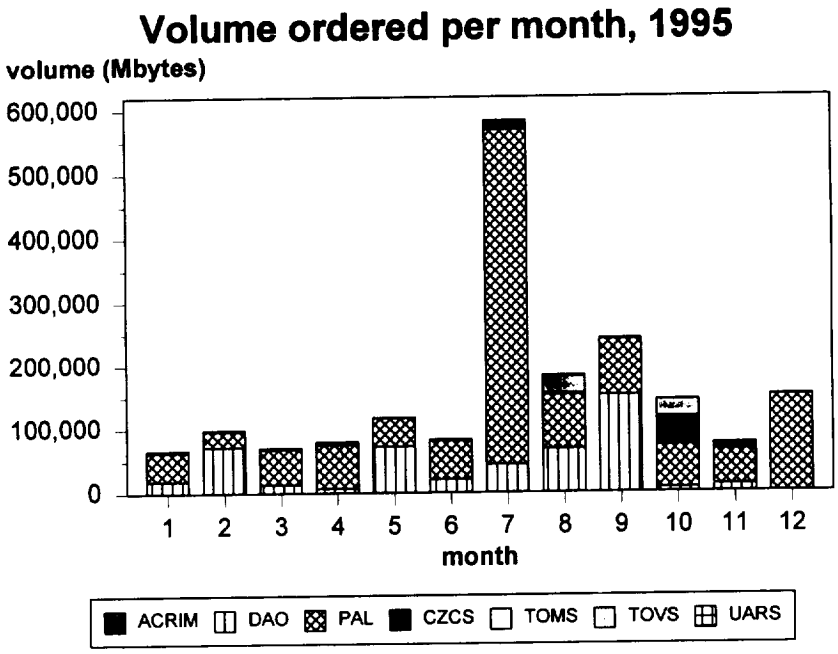

Figure 1. User volume ordered aggregated by data product.

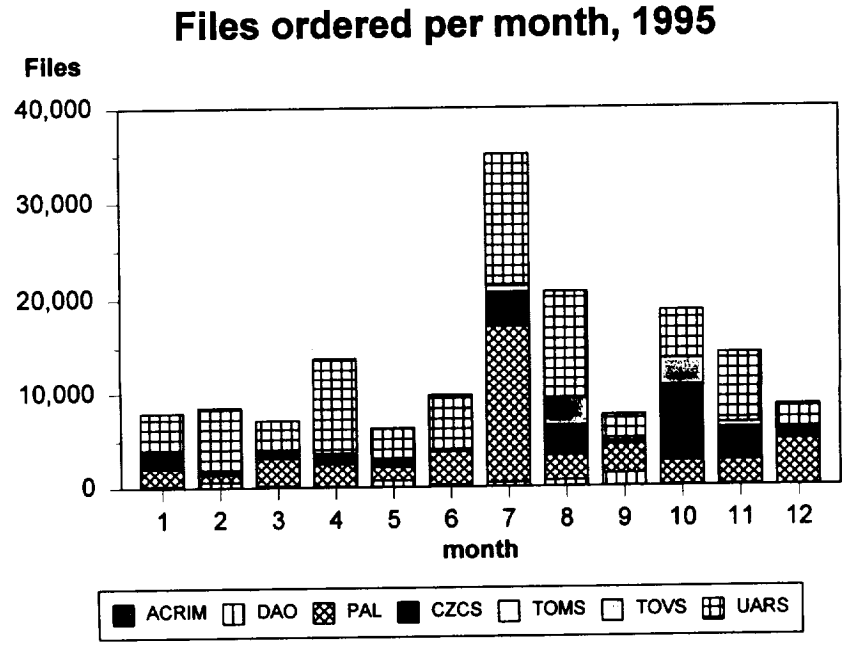

Figure 2. Files ordered aggregated by data product.

Next, we analyze requests according to data level (e.g. Level 1-4). The data that is ingested into the archive is in a variety of higher level data sets. Level 1 data is satellite data with corrections, while higher level data is binned and aggregated. Because of the processing, there is more volume in the lower level products (L1-2) than in the higher level products (L3-4). Figure 3 aggregates user volume by month and data level. As

expected most of the volume of requested data is for Level 3 data, and there is a substantial interest in Level 4 data.

Figure 4 aggregates volume ordered by month and by interface method. A substantial percentage of the total volume requested was from the Character User Interface (ChUI). The volume of standing orders is also important. Files that belong to standing orders are transferred to a distribution staging area soon after being ingested. This method of distribution reduces the load imposed on Unitree because the standing orders files are processed before the data is migrated to the near-line devices.

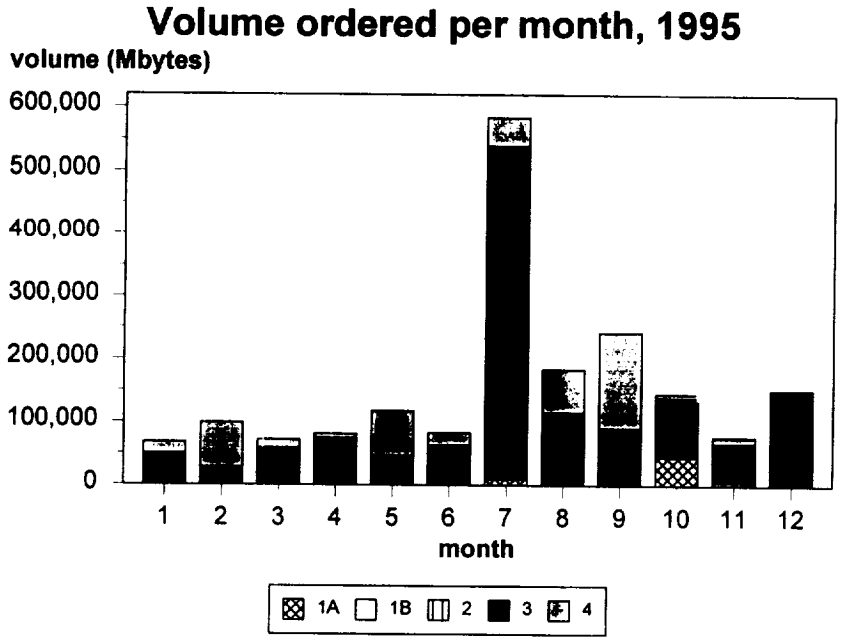

Figure 3. User volume ordered aggregated by data level.

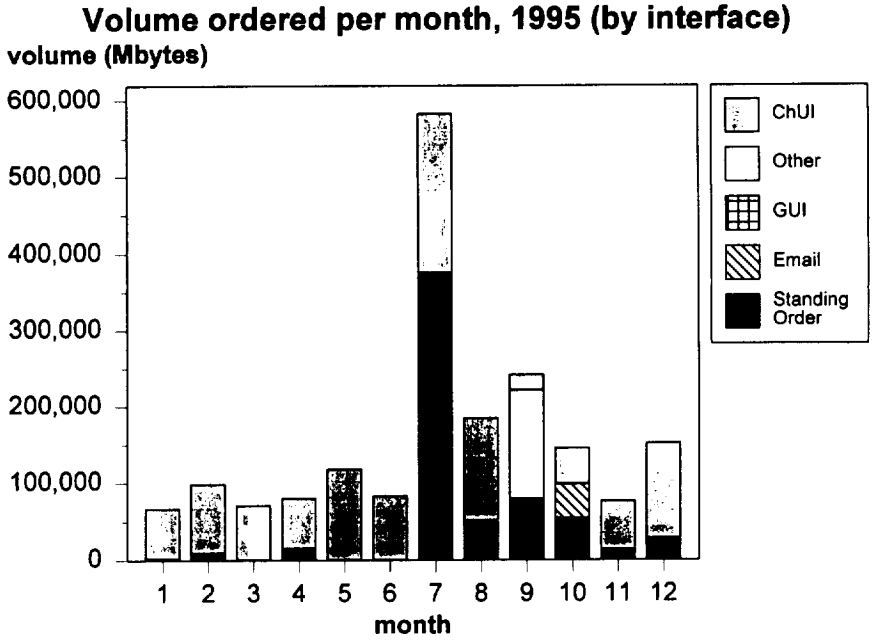

Figure 4. User volume ordered aggregated by interface and month.

In Figure 5 and Figure 6, we aggregate the volume ordered by interface and by hour of the day and day of the week, respectively. The volume ordered shows the typical pattern -- most requests are made during normal working hours. These results are consistent with observations of NDADS, although more requests are made to NDADS during weekends, and few requests are made to NDADS during early morning hours.

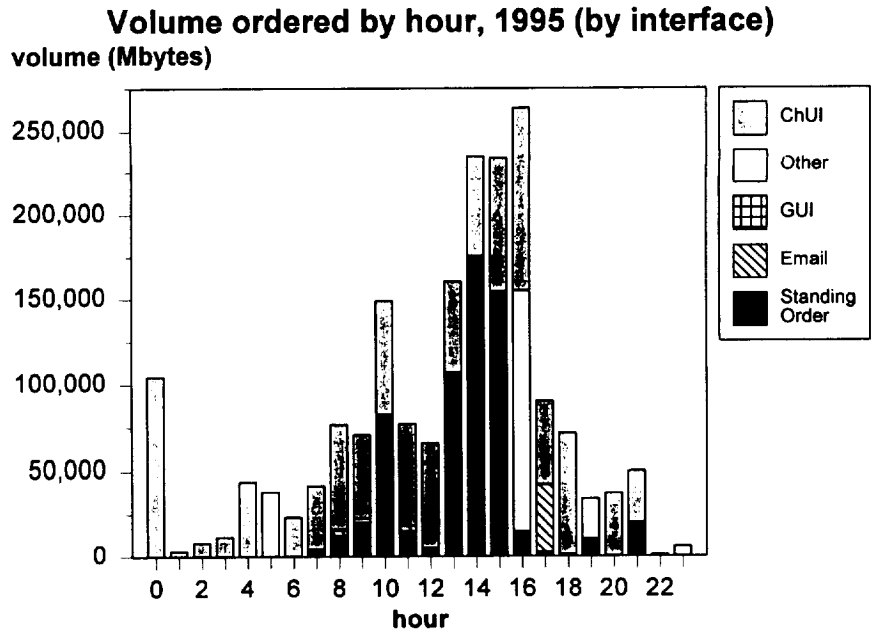

Figure 5. User volume ordered aggregated by interface and hour.

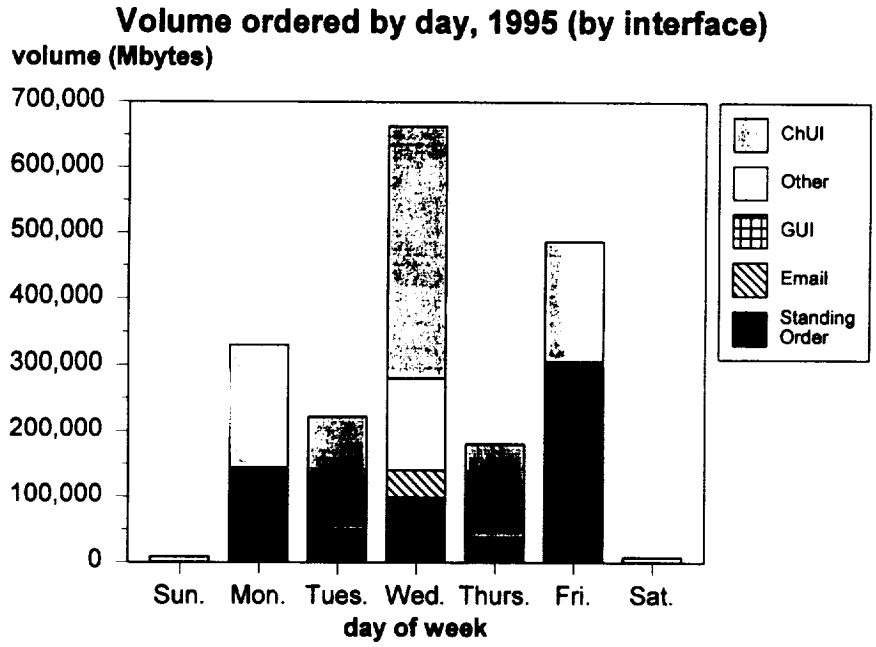

Figure 6. User volume ordered aggregated by interface and hour of the day.

The requested data can be distributed either electronically (via ftp) or on one of several different tape media. Figure 7 aggregates user requests by distribution media and by week of the year. Most data is distributed by creating 4-MM or 8-MM tapes. There is almost no request for 9-track tapes. The volume of ftp requests accounts for only a small portion of the data distribution. We should point out that, due to resource constraints (network and disk space), a limit has been placed on the volume of data that can be distributed via ftp for a given request.

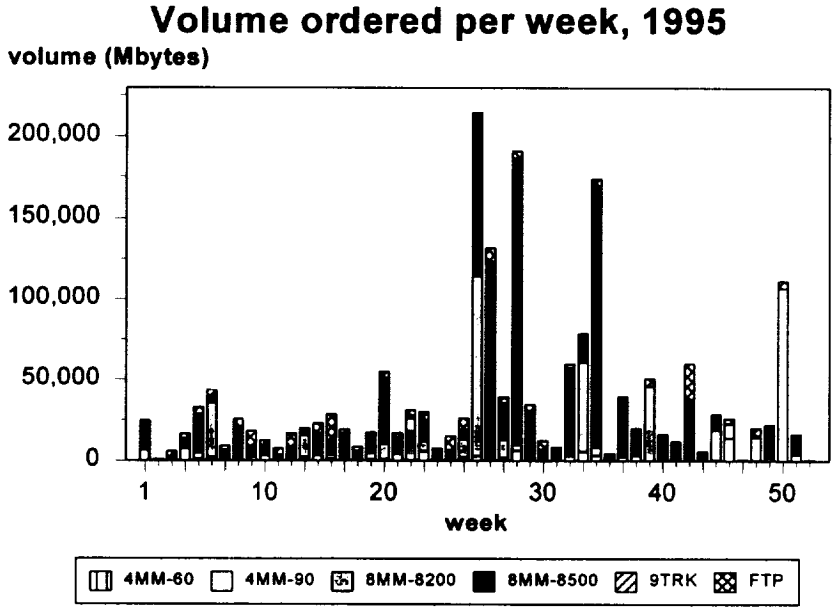

Figure 7. User volume ordered aggregated by media of distribution.

Finally, we plot the volume of data requested by different regions of the world in Figure 8. While most of the request volume came from North America, there is significant world-wide use of the DAAC.

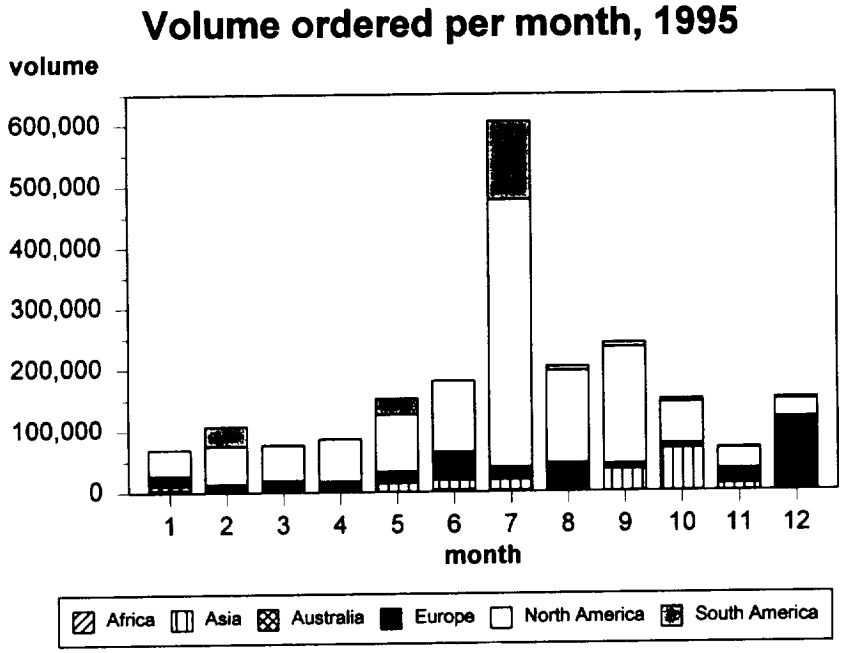

Figure 8. User volume ordered aggregated by region.

User Analysis

In this section, we analyze request size and user activity. The database records for every file accessed; the file size, the data set that the file is a member of (a refinement of data product), and a unique request ID. From this information, we can reconstruct the size of a request. In Figure 9, we plot the volume per request, in Figure 10, we plot the files per request. Figure 9 and Figure 10 are plotted on a log scale because of the very large range of request sizes. The wide range of request sizes suggests that request servicing should be aware of the size of the request and handle it accordingly.

Data in a data product can be divided into *data sets*, based on the type of the data (i.e., different levels, different sensors, etc.). In Figure 11, we plot the number of data sets requested per order. The number of files per request and the number of data sets per request are not correlated, as is shown in Figure 12. The average request in 1995 accessed 78.6 files and 1.6 data sets. These results show that most requests are clustered by the data set, with the implication that archive media should store files of a single data set. These results are consistent with our study of the NDADS archive.

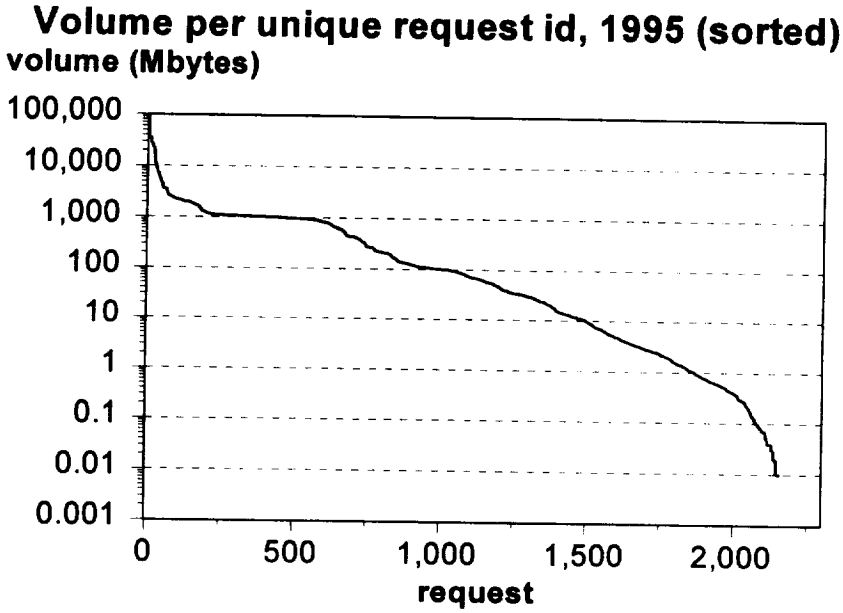

Figure 9. Volume per unique request id.

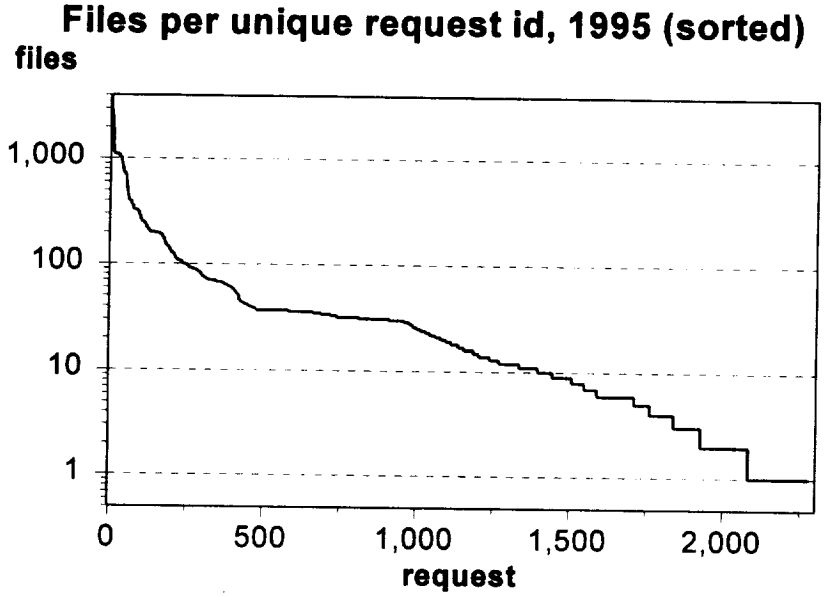

Figure 10. Files per unique request id.

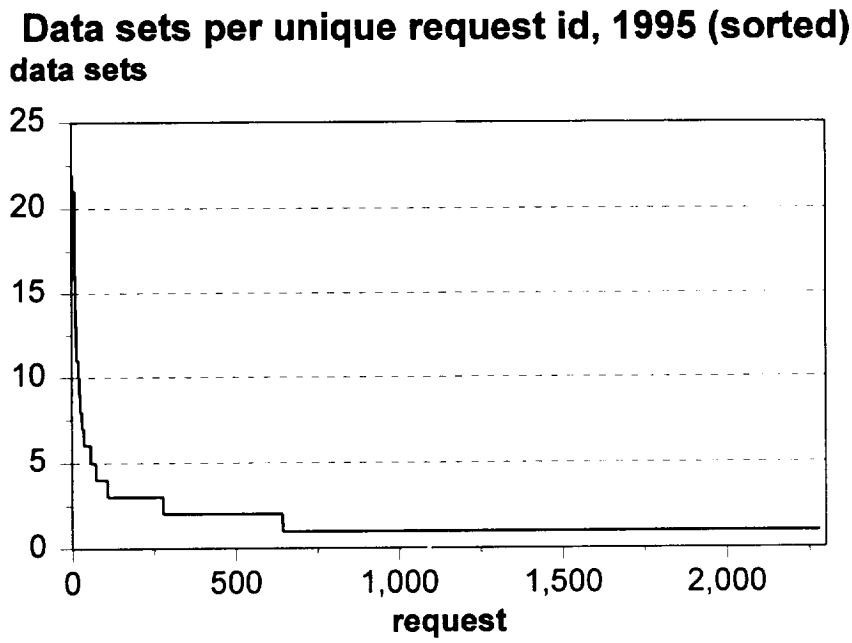

Figure 11. Data sets ordered per unique request id.

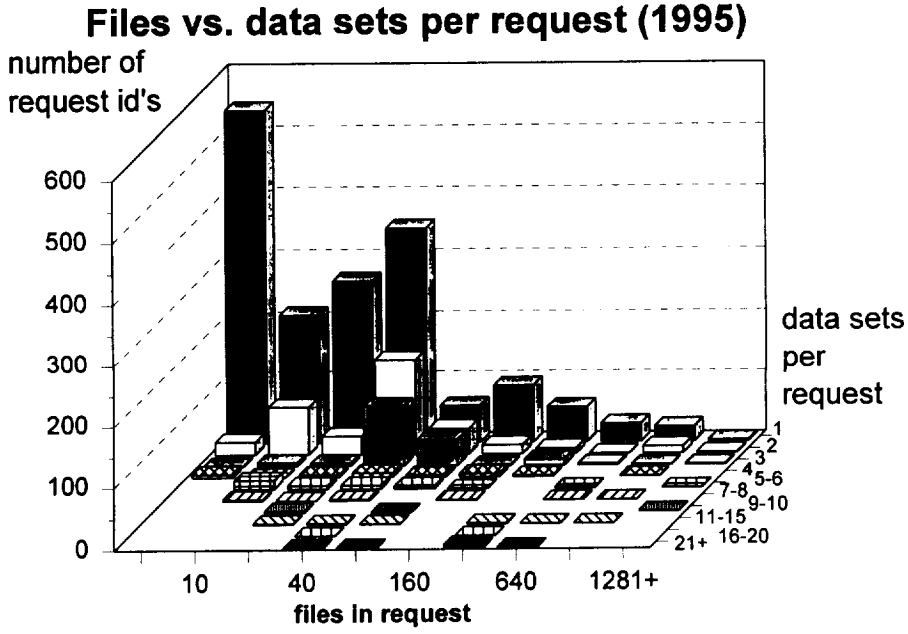

Figure 12. Files vs. data sets in a request.

We can also aggregate requested files based on the user ID. In Figure 13, we plot the number of files requested per unique user, in sorted order, and in Figure 14, we plot the volume requested per unique user in 1995. The curve is non-linear even when plotted on a logarithmic chart. We found that the top 20 users (of 442) requested 47% of all files and 70% of the data volume. We note that top 20 users, rated by files requested, is not the same as the top 20 users rated by volume requested. However, the correlation between

files requested and volume requested is strong (the intersection between the two top-20 sets contains 12 members). A scatter plot of volume requested vs. files requested is shown in Figure 15. These results are consistent with our study of the NDADS archive.

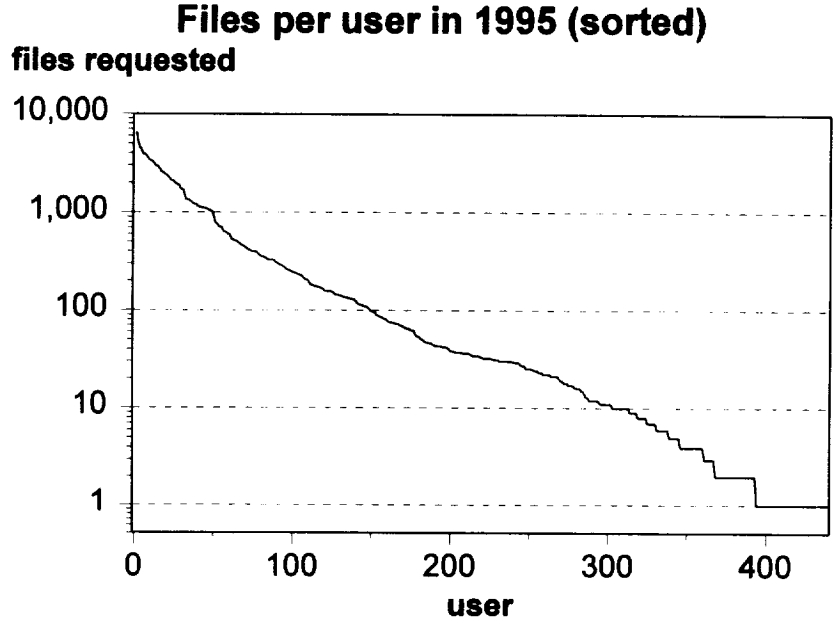

Figure 13. Files per unique user.

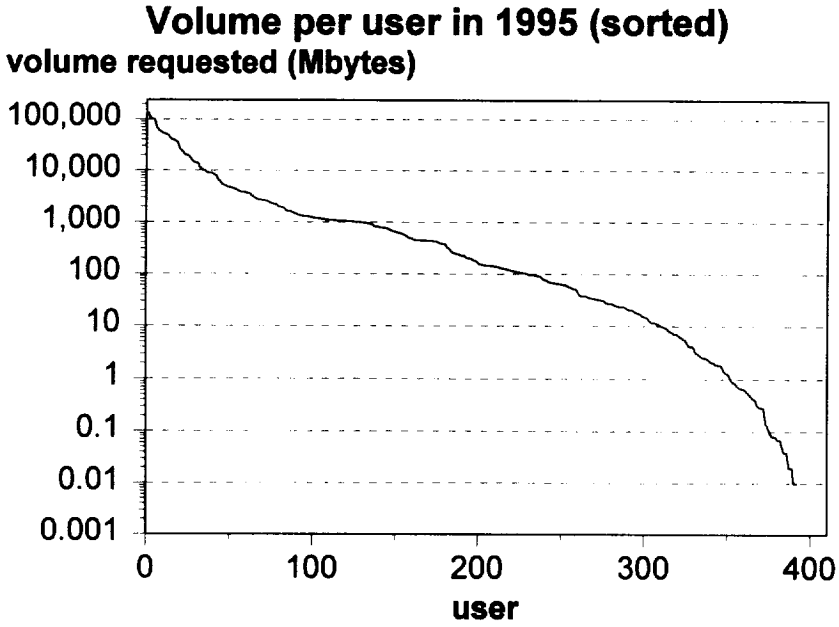

Figure 14. Volume per unique user.

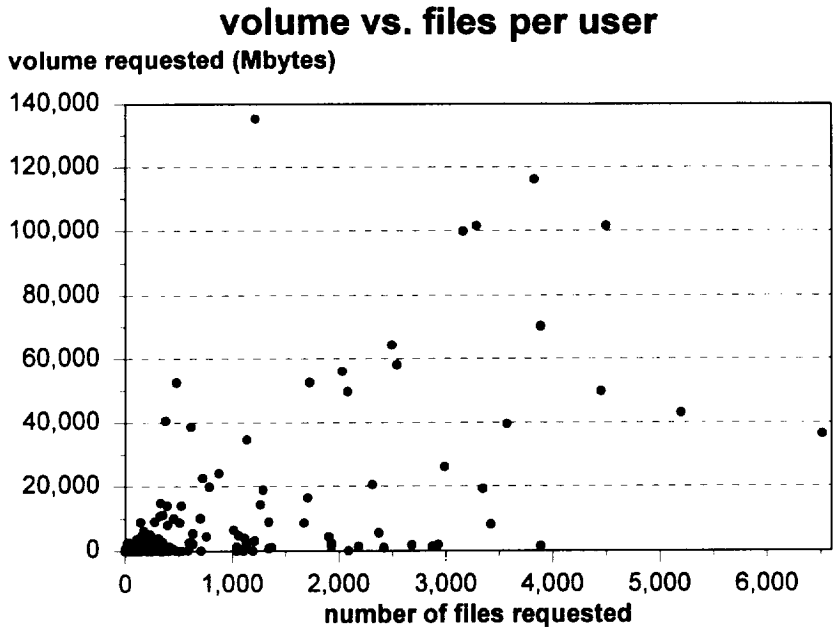

Figure 15. Scatter plot of volume ordered and files ordered per user.

Caching

When a user requests a file, the file is first searched in the on-line disk space. If the file is not located on the cache it is then fetched from tertiary storage into secondary storage and made available to the requester. The file typically has a minimum residency requirement to give the requester time to access the file. While the file is disk-resident, a second request for the file can be satisfied without fetching the file from tertiary storage. These cache hits can reduce the load on the tertiary storage system, and also improve response times. Fetching a file from the tertiary storage requires a tape to be picked by a robotic device, mounted, the file searched on the tape and then read. All this can take minutes before the file is ready to be read.

The archive systems should have enough disk storage to satisfy the minimum residency requirement. However, files referenced within the minimum residency may be deleted if the cache runs out of space. In this case, using a FIFO algorithm the oldest files are deleted first. The buffer might run out of disk space necessary to satisfy minimum residency due to a high request load, or due to a high *ingest* load (i.e., the ingested files must be stored on-line until they can be migrated to tertiary storage). Although the ingest load can interfere with the cache, we do not consider it in this study. However, the relative performance of the algorithms will be the same with or without the ingest load.

If the cached files are large (an average size of 12.8 Mbytes in this study), then the time to transfer referenced files not in the cache can be very long. We compute the *cost* of servicing a reference string to be the weighted sum of the number of cache misses and the number of bytes transferred. In these studies, we assumed that transferring 10 Mbytes is equal to the cost of a cache miss (The time to load a media is much larger than the transfer time, but this cost is amortized over all files loaded from the media). Let $cost_f$ be

the cost of transferring file f from the archive to on-line storage, and let S_f be the size of file f. Then, the (normalized) value of $cost_f$ is:

$$cost_f = 1 + S_f \ / \ 10 \text{ Mbytes}$$

We can evaluate the benefit of using a cache by looking at the *hit rate* of the cache, or by looking at the *cost reduction* of the cache. The cost reduction is the reduction of load on tertiary storage caused by the cache. More formally, let the *reference string* (i.e., the sequence of requests that pass though the cache) be $r=(f_1, f_2, ..., f_m)$. Let $Miss(f_i)$ have the value 1 if f_i was not in the cache when it was referenced, and 0 if it was in the cache. Then, the cost of processing the reference string when using a cache, *cost(cache)* is:

$$cost(cache) = \Sigma_{fi \in r} cost_{fi} * Miss(f_i)$$

Let *cost(tot)* be the cost of servicing r when no cache is used (i.e., $Miss(f_i)=1$ for every f_i). Then the *fraction of cost saved* by using the cache is:

$$\text{fraction of cost saved} = 1\text{-}cost(cache)/cost(tot)$$

A large body of caching literature exists when all cached objects are of the same size. The Least Recently Used (LRU) replacement algorithm is widely recognized as having good performance in practice. Caching objects of widely varying sizes is somewhat more complicated. If one wants to minimize the number of cache misses, then it is much better to choose large files than small files for replacement, because removing large files frees up more space. Let the set of files in the cache be F. The general scheme is to assign to each file f in F a *weight*, $weight_f$, and chose for replacement the file with the largest weight. Note that many files might need to be replaced on a cache miss.

The optimal replacement algorithm for variable size objects, with respect to cache misses, is the GOPT algorithm [DS78]: For file $f \in F$, let N_f be the time until the next reference to f and let S_f be the size of f. Set $weight_f = N_f * S_f$, and choose for replacement the file f in F such that $weight_f$ is the largest.

The GOPT algorithm cannot be implemented (because it requires knowledge of future events), but it can be approximated. The Space-Time Working Set (STWS) algorithm [Sm81c] approximates GOPT by substituting P_f the time since the last reference to f, for N_f.

While STWS can be implemented, it also requires a great deal of computation. For this reason, STWS is often approximated by what we call the STbin algorithm [Mi94]: A file is put into a bin based on its size. The files in a bin are sorted in a list using LRU. To choose a file for replacement, look at the file at the tail of each bin and compute its weight to be $P_f * S_f$. Choose for replacement the file with the largest weight.

The STbin algorithm does not account for the cost of transferring files, and may discriminate too strongly against large files. We examined two algorithms that modify the weight function to account for transfer costs. Let $cost_f$ be cost incurred if file f must be loaded. The *Costbin* algorithm computes the weight of file f to be $P_f * S_f / cost_f$, where $cost_f$ is defined above. Alternatively, we can use a non-linear function. The *Alphabin*

algorithm computes the weight of file f to be $P_f * S_f^{\alpha}$, where α is a real number. Note that setting α=0 gives LRU and setting α=1 gives STbin.

Another method for incorporating size and last-reference time into victim selection is to take a weighted sum. Let K_s be the *size factor* and let K_t be the *time factor*. Then, choose for replacement the file with the smallest weight $K_s * S_f + K_f * P_f$. Since the file weight is computed by a sum, we call the algorithm the *SUM* algorithm. It is used by several HSM products.

Recent work in caching algorithms has produced *statistical caching* algorithms [OOW93]. The LRU/2 algorithm chooses for replacement the object whose penultimate reference (instead of most recent reference) is the furthest in the past. We adapt LRU/2 to file caching by maintaining the bins in the ST-bin algorithm by LRU/2 instead of LRU. We call the new algorithm *LRU/2-bin*.

HSM systems typically use a ``watermark" technique to manage their staging disk. When the staging disk space utilization exceeds a high watermark, files in the staging area are migrated into tertiary storage until the staging disk utilization reaches a low watermark. The motivation for the watermark technique is to write back dirty files in a single burst, thus improving efficiency by exploiting write locality. The archive that we study contains read-only files, so the watermarks should be set has high as possible for maximum efficiency.

The minimum residence period is implemented by partitioning the cache into the regular cache and the *minimum residence* cache. When a file is referenced, it is placed in the minimum residency cache, where it remains until the minimum residence period has passed. After the minimum residence period, the file is placed in the regular cache. Normally, files in the minimum residence cache are not selected for replacement. However, if the minimum residence cache size exceeds the total cache size, the oldest files in the minimum period cache are chosen for replacement.

In our caching analysis, we assume a disk block size of 1024 bytes, and set a limit on the number of disk blocks that are available for caching. We trigger replacement when fetching a new file will cause the space limit to be exceeded, and we remove files until the space limit will not be exceeded. For the STbin and LRU/2-bin algorithms, bin i holds files that use between 2^i and $2^{i+1}-1$ blocks. We set the minimum residency period to 1 day. We retrieved from the database a listing of all files requested in 1995[2], and sorted the list of time of reference to create the reference string for our cache simulators. We report both the hit rate and the reduction in cost due to running a aching algorithm with a particular cache size.

We first test the STbin variants. In Figure 16, we plot the cost reduction as we vary α for different cache sizes. The best setting of α is approximately 1/2. However, the improvement over STbin is not large. Next, we compare Costbin against STbin in Figure 17. While Costbin has better performance than STbin, the difference is not large.

[2] Subject to the restrictions listed in Section 1.2

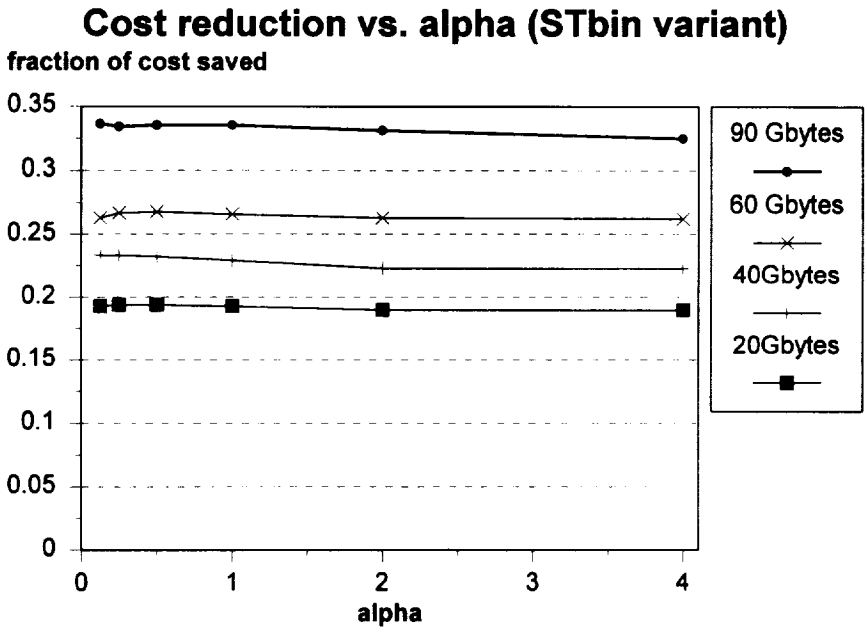

Figure 16. Finding the best value of α for alphabin.

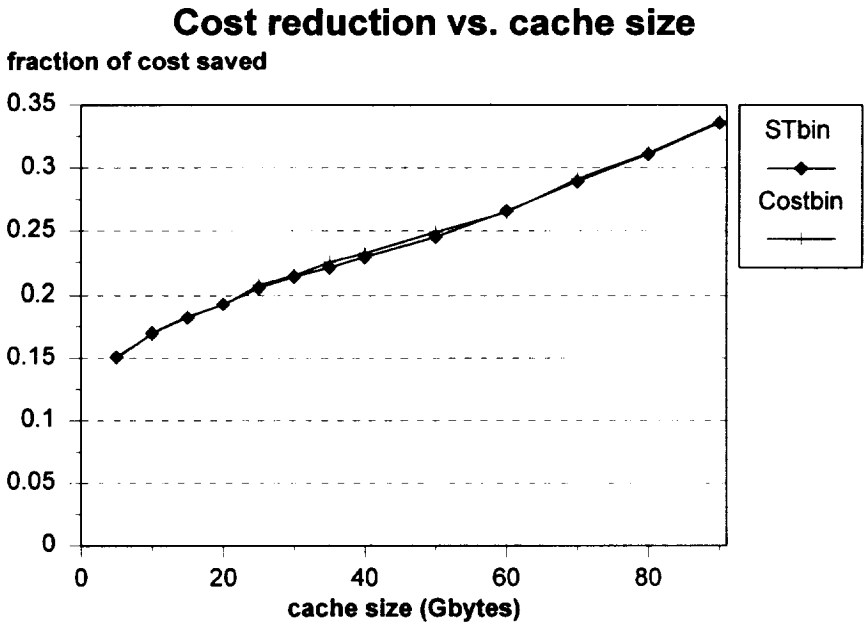

Figure 17. Comparison of Costbin to STbin.

In Figure 18 we plot the fraction of cost saved for the LRU, LRU/2, SUM, and STbin algorithms algorithms as we increase the cache size from 5 Gbytes to 60 Gbytes, and in Figure 19 we plot the hit rate. The results show that the STbin and the LRU/2-bin algorithms are significantly better than LRU, and somewhat better than the SUM algorithm. The LRU/2-bin algorithm had somewhat better performance than the STbin

algorithm. We note that STbin requires less CPU time for execution than either the LRU/2 or the SUM algorithm, and the SUM algorithm requires careful tuning.

The results show that caching can be effective in reducing the load on the tertiary storage device, in spite of the highly random nature of requests to an on-line archive. The GSFC Version 0 DAAC currently uses a 60 Gbyte distribution cache. Simulation results indicate that this size cache can provide a hit rate and cost savings of about 25%. The Unitree logs indicate that the internal Unitree cache (32 Gbyte) had an additional hit rate of 15.6%.

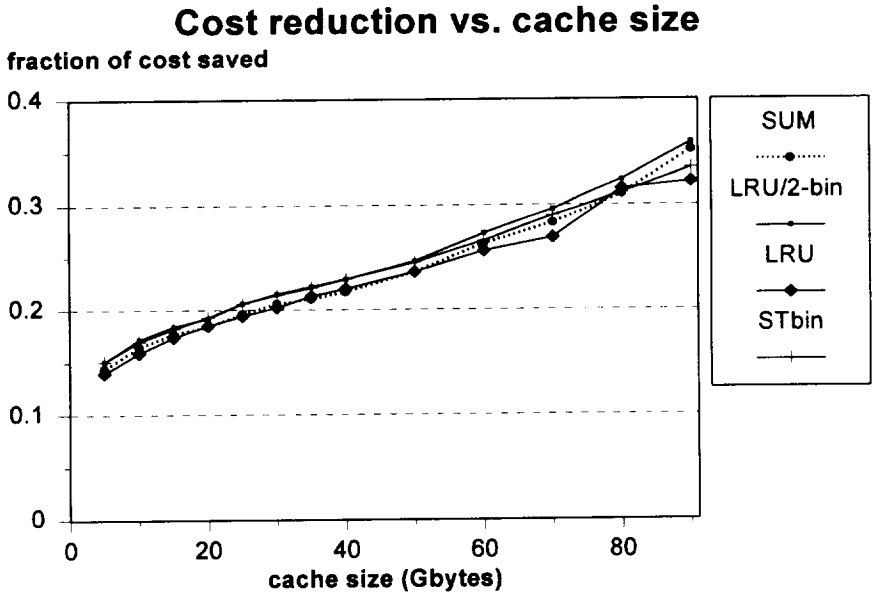

Figure 18. Cache algorithm comparison (cost reduction).

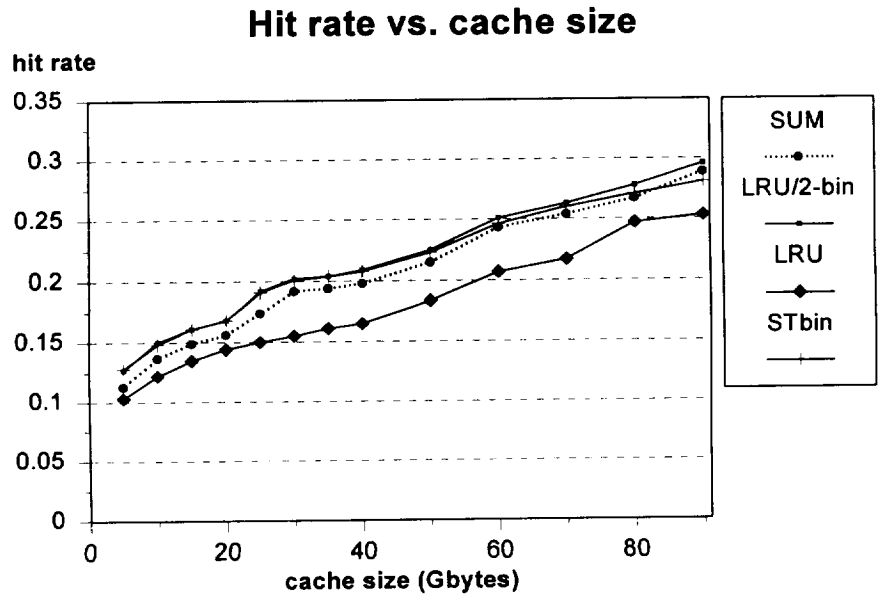

Figure 19. Cache algorithm comparison (hit rate).

We ran another experiment to determine the effect of changing the minimum residence period. Figure 20 shows the cost reduction of the STbin algorithm as the minimum residence period is varied from 10 minutes to 2 days. We varied the cache size between 10 and 60 Gbytes. Increasing the minimum residence time decreases the cost reduction, but the effect is small.

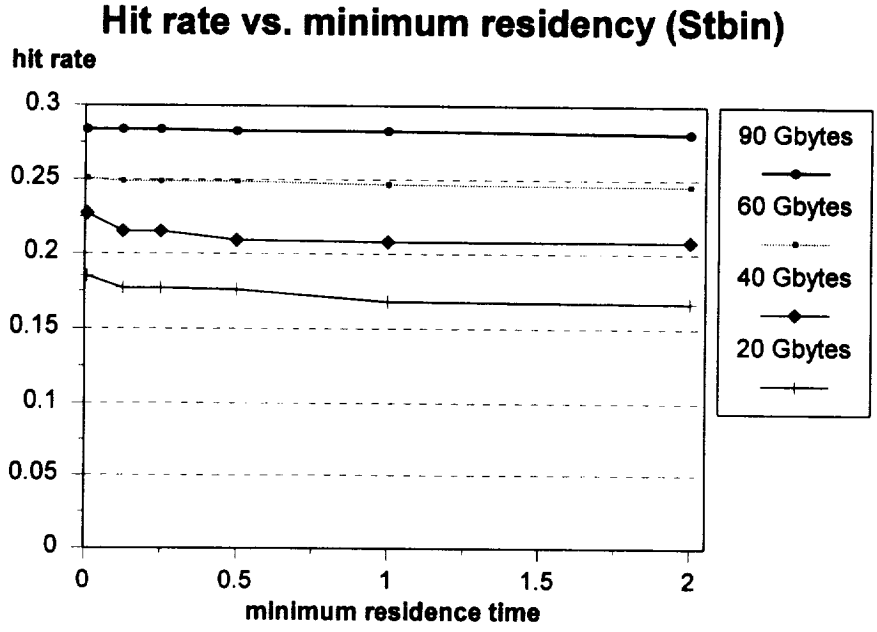

Figure 20. Effect of changing the minimum residence time.

File Access Pattern Analysis

The success of caching depends upon the file access patterns. In this section we examine some aspects of the access patterns. These results also have implications for archive design.

Most files are accessed only a few times, limiting the maximum cache hit rate. Figure 21 plots distribution of the number of times a file was referenced in 1995. The fact that so most files are referenced once limits the performance of statistical caching algorithms, such as LRU/2-bin. We note further that only 12% of the 550,000 files in the archive were requested during 1995. This result is consistent with observations of the NDADS archive.

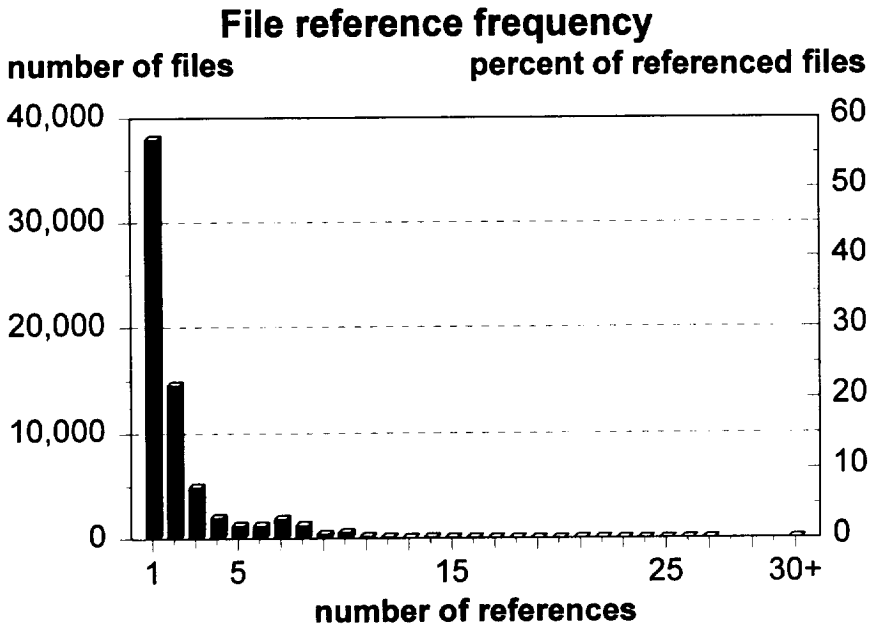

Figure 21. Distribution of the number of references to a file in 1995.

The effectiveness of caching also depends upon the average time between references to a file (the *inter-reference* time). In Figure 22 we plot the distribution of inter-reference times during 1995. To generate this plot, we scanned through all file accesses and searched for repeat accesses. Whenever a repeated reference was found, we incremented a histogram based on the number of days since the last reference. The plot shows that most repeat references occur shortly after an initial access, but that the inter-reference time distribution has a long tail. The rise at the end of the tail represents all repeat references with an inter-reference time of 186 days or larger. The average number of days between an access to a file, given that the file is accessed at least twice in 1995 is 46.1 days. This result is essentially consistent with observations of the NDADS archive, which has an average interreference time of 27.6 days. Both archives show a peak in the inter-reference time near 0 days, and at 1 and 2 months after the previous reference. However, these characteristics are stronger in the NDADS references. For both archives, the inter-reference time distribution has a long tail (i.e., represented by the point at ``185+'').

We found that many of the repeat references are due to the same user requesting a file for a second time. This is shown in Figure 23, which plots the fraction of repeat requests that are due to the same user, by time since last request. In total, 15.4% of the repeat references in 1995 are due to the same user as had submitted the previous reference. By contrast, 57% of the repeat references to NDADS are due to the same user. One explanation for this difference is that most requests to the GSFC Version 0 DAAC are submitted interactively, while most requests to NDADS are submitted by email. Network and mailer delays compound archive delays to cause the user to suspect that the request has been lost.

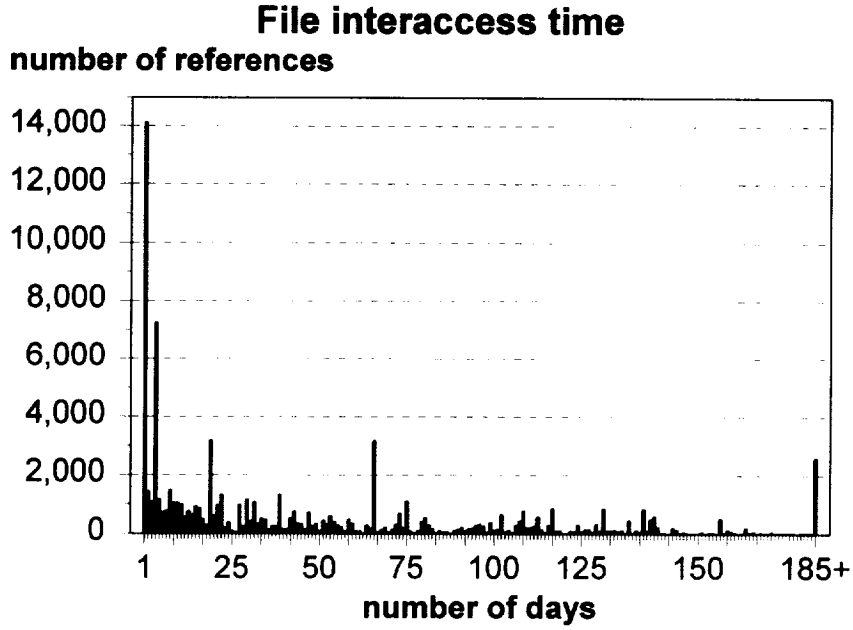

Figure 22. Distribution of file inter-reference times. The point at ``185+'' represents the tail of the distribution.

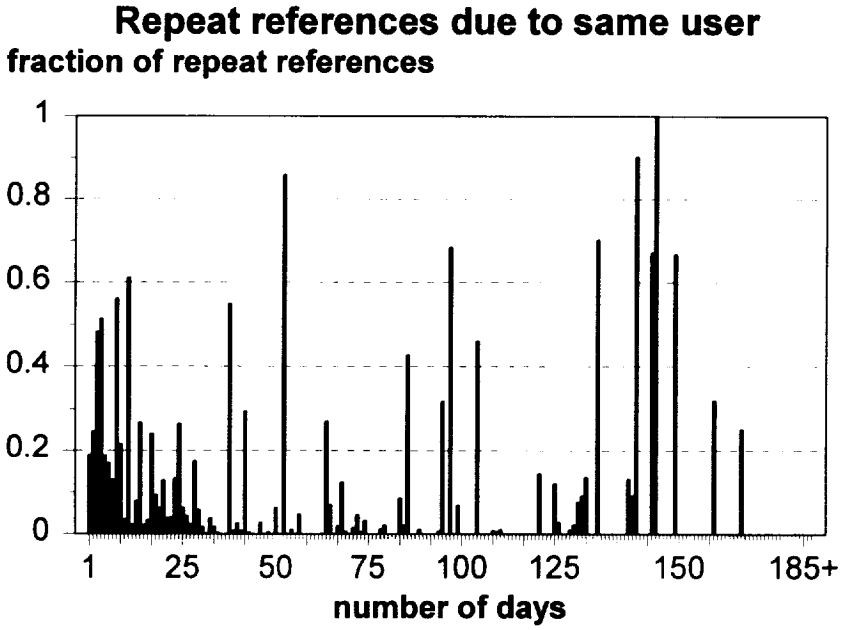

Figure 23. Fraction of repeat references in which both the current and the previous reference are submitted by the same user (binned on inter-reference time).

The performance of the STbin algorithm and its variants (Alphabin and Costbin) depends on the distribution of file sizes. In Figure 24, we plot the file references binned on the file sizes. Large files account for a large fraction of the accesses (the average size of a requested file is 12.8 Mbytes). Caching large files can be effective if caching large files is likely to result in a cache hit. In Figure 25, we plot the proportion of file references that

are to files previously referenced, binned by file size. We also plot the average interaccess time. Large files have high reaccess rates, and for this reason the STbin variants improve hit rates as well as cost reduction.

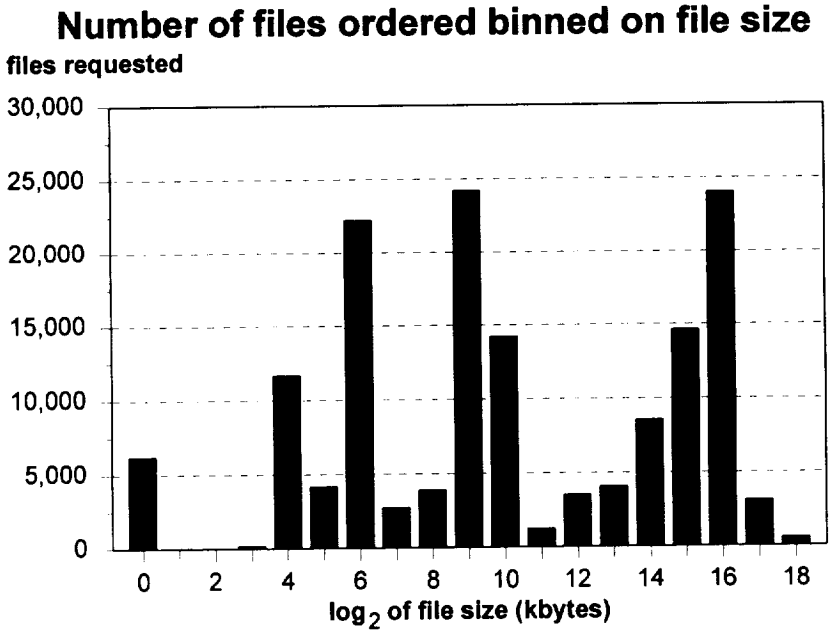

Figure 24. Ordered files binned on file size.

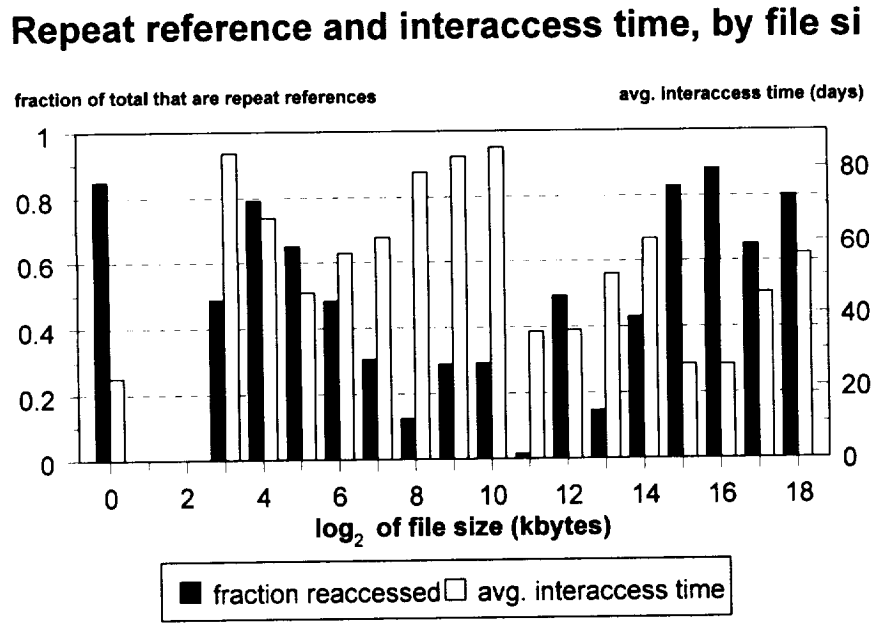

Figure 25. Repeat requests and interaccess times binned on file size.

The result that the average size of a requested file is 12.8 Mbytes was unexpected, because the average size of a file in the archive is 1 Mbyte. The bias towards ordering large files is due in part to the anonymous ftp archive, which serves small files. Users prefer small files, but there is a high volume of data ordered for the DAO and PAL data

products, both of which are stored in large files (an average of 16.5 Mbytes and 55.0 Mbytes, respectively).

In Figure 26, we plot the ``age'' of the files that are referenced (requested). We compute this distribution as follows. For every file referenced in the observation period, we compute the *file* age to be the difference between the reference time and the time that the file was archived. We bin the referenced files based on file age in weeks. Because there is a dependence between the time of reference and the file age, we plot the file age distribution for each quarter of 1995. The peak in the age of the referenced files in all four charts corresponds to roughly the same archiving dates (we note that many files were rearchived in early 1995). Recently ingested data does not show an unusually high user interest. One explanation for this result is that new data is not immediately known to most users, and it is only after some advertisement (newsletter, conference, word of mouth) that the data may be more frequently requested. This result is consistent with observations of the NDADS archive.

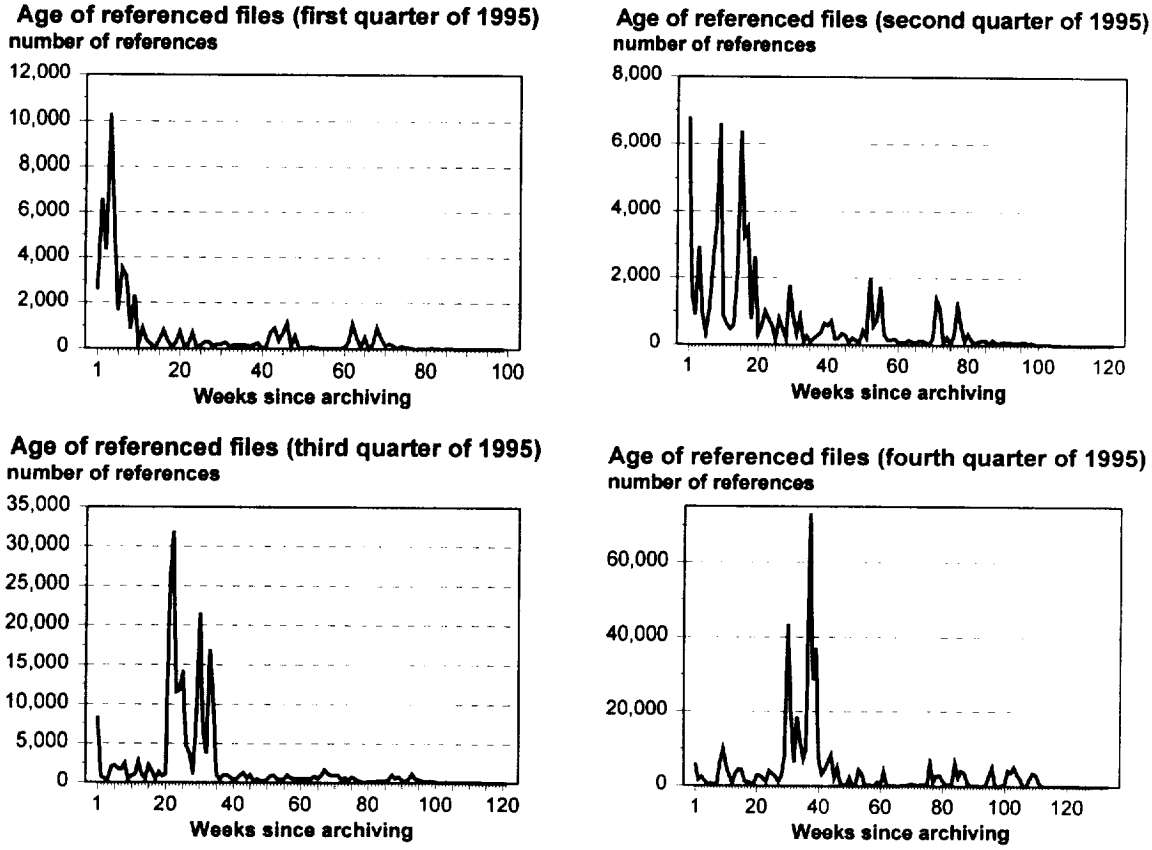

Figure 26. Distribution of time between file archive and file reference.

Internal Activity

We conclude with some observations of the work performed by the archive. In Figure 27, we plot the number of Unitree media mounts per week, and the average number of files transferred per mount (we obtained this data from the Unitree logs). The average

number of files transferred per mount for all of 1995 is 8.6. Unitree does not distinguish between mounts to read or write data. Consequently, the average number of files transferred includes both read and write operations. Migrations are executed periodically (e.g., one hour) to increase the chance of writing multiple files to the same media. Stage operations are performed as soon as the resources are available (e.g., tape drives). It would have been interesting to derive the average number of files retrieved for stage operations only, and to correlate the stages with the requests. This could have provided some insight on how well the stage operations are scheduled and how clustered the files requested are on tapes.

The reader might note that this chart does not correlate well with the results presented in Figure 1 through Figure 8. There are two reasons for this discrepancy. First, the data in Figure 1 through Figure 8 is based on time an order was requested, not time the order was processed or distributed. Request processing might be delayed due to heavy loads, or to handle very large requests (e.g., see Figure 9).

As described in the introduction, we have limited this study to only those orders that are filled by "pulling" data from the mass storage system Unitree. However, the entire activity for the DAAC is significantly higher (see Figure 28) because of the other distribution methods used at GSFC DAAC that are not included in this study (e.g. CD-ROM, anonymous ftp, off-line requests). It is interesting to note that the distribution volume is much greater than the ingest volume.

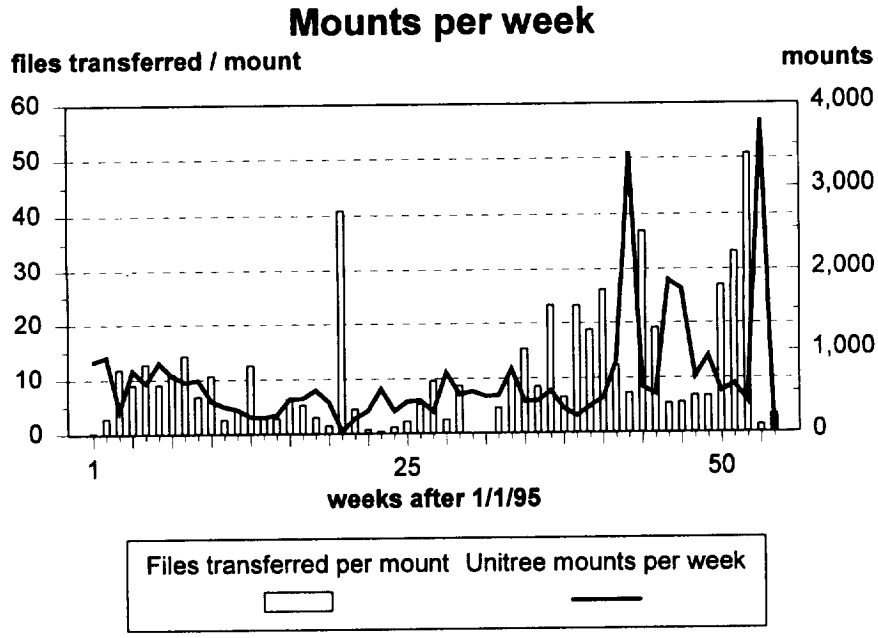

Figure 27. Unitree mounts per week and files per mount.

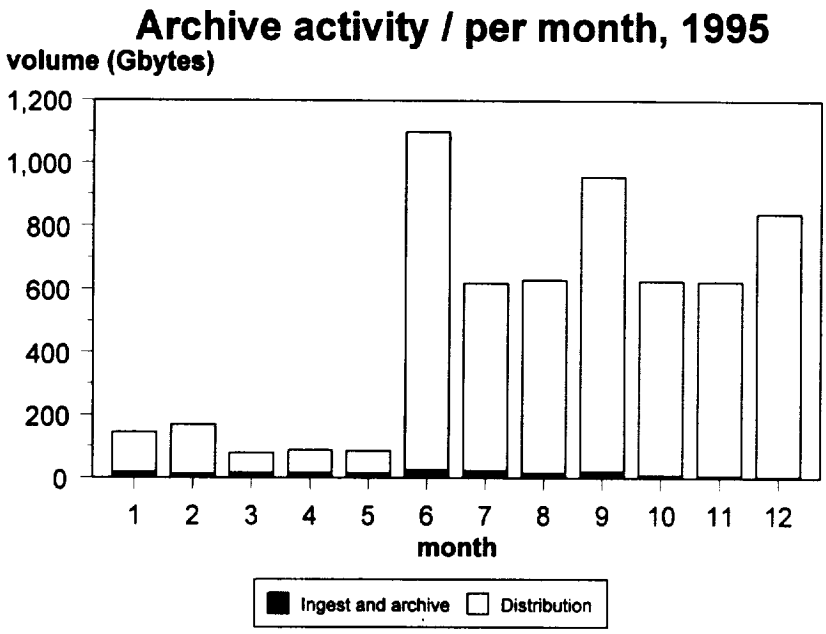

Figure 28. Total archive activity, per month.

Conclusions

We have presented a study of the external requests made to the GSFC Version 0 Distributed Active Archive Center. The analysis examined only a subset of all requests submitted. In particular orders for CD-ROMs, off-line requests and anonymous ftp are excluded because they did not affect the performance of Unitree and the near-line devices. A summary of the results are:

- Most of the volume of the data ordered is concentrated on two of the seven data products, and on higher level data.
- Most of user requests (by volume) were submitted via the Character-based User Interface (ChUI).
- Most of the volume of data is distributed via tape.
- The requested volume varies greatly between months. Most of that volume is submitted during normal working hours.
- There is a wide range of request sizes, and some requests are very large (100+ Gbytes).
- Most requests require service from a small number of data sets.
- A small set of hot users account for most of files and volume requested.
- LRU/2-bin is the best file caching algorithm on this workload, STbin also works well.
- The file interreference distribution has a peak at < 1 day, and a long tail.

- Interest in data is not correlated with the time of archiving.

Acknowledgments

We would like to thank Dr. Blanche Meeson for reviewing this paper, and we would like to thank Andy Griffin, Gary Gregorich, Frances Bergmann, Robert Swafford, and Hughes STX for their help with collecting data.

Bibliography

[AN88] E.R. Arnold and M.E. Nelson. Automatic Unix backup in a mass storage environment. In *Usenix - Winter* 1988, pages 131-136, 1988.

[DS78] P.J. Denning and D.R. Sluts. Generalized working sets for segment reference strings. *Communications of the ACM*, 21(9):750-759, 1978.

[EP82] C.W. Ewing and A.M. Peskin. The masstor mass storage product at Brookhaven national laboratory. *Computer*, pages 57-66, 1982.

[ESDIS] Esdis document catalog. http://spsosun.gsfc.nasa.gov/ESDIS_Docs.html.

[HP89] R.L. Henderson and A. Poston. MSS II and RASH: A mainframe unix based mass storage system with a rapid access storage hierarchical file management system. In *USENIX - Winter* 1989, pages 65-84, 1989.

[Jo95] T. Johnson. Analysis of the request patterns to the nssdc on-line archive. *In Proc. 4th NASA Goddard Conf. on Mass Storage Systems and Technologies*, 1995, NASA Conference Publication 3295

[JR91] D.W. Jensen and D.A. Reed. File archive activity in a supercomputing environment. Technical Report UIUCDCS-R-91-1672, University of Illinois at Urbana-Champaign, 1991.

[LRB82] D.H. Lawrie, J.M. Randal, and R.R. Barton. Experiments with automatic file migration. *Computer*, pages 45-55, 1982.

[Mi94] E. Miller, 1994. Private communication. Thanks also to comp.arch.storage.

[MK91] E.L. Miller and R.H. Katz. Analyzing the I/O behavior of supercomputing applications. In *Supercomputing '91*, pages 557-577, 1991.

[MK93] E.L. Miller and R.H. Katz. An analysis of file migration in a unix supercomputing environment. In *USENIX - Winter* 1988, 1993.

[OOW93] E.J. O'Neil, P.E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In Proceedings of the *1993 ACM Sigmod International Conference on Management of Data*, pages 297-306, 1993.

[Sm81d] A.J. Smith. Analysis of long-term reference patterns for application to file migration algorithms. *IEEE Trans. on Software Engineering*, SE-7(4):403-417, 1981.

[Sm81c] A.J. Smith. Long term file migration: Development and evaluation of algorithms. *Communications of the ACM*, 24(8):521-532, 1981.

[Str92] S. Strange. Analysis of long-term unix file access patterns for application to automatic file migration strategies. Technical Report UCB/CSD 92/700, University of California, Berkeley, 1992.

[TS93] A. Tarshish and E. Salmon. The growth of the UniTree mass storage system at the NASA Center for the Computational Sciences. *In Third NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 179-185, 1993, NASA Conference Publication 3262

[TH88] E. Thanhardt and G. Harano. File migration in the NCAR mass storage system. In *Mass Storage Systems Symposium*, pages 114-121, 1988.