Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets

Ioan Raicu[*], Ian Foster[*+], Alex Szalay[#]

*Computer Science Dept.
The University of Chicago
iraicu@cs.uchicago.edu

+Math and Computer Science Div.
Argonne National Laboratory
foster@mcs.anl.gov

#Dept. of Physics and Astronomy
The Johns Hopkins University
szalay@jhu.edu

Abstract

Grid computing has emerged as an important new field focusing on large-scale resource sharing and high-performance orientation. The astronomy community has an abundance of imaging datasets at its disposal which are essentially the "crown jewels" for the astronomy community. However, these astronomy datasets are generally terabytes in size and contain hundreds of millions of objects separated into millions of files—factors that make many analyses impractical to perform on small computers. The key question we answer in this paper is: "How can we leverage Grid resources to make the analysis of large astronomy datasets a reality for the astronomy community?" Our answer is "AstroPortal," a gateway to grid resources tailored for the astronomy community. To address this question, we have developed a Web Services-based system, AstroPortal, that uses grid computing to federate large computing and storage resources for dynamic analysis of large datasets. Building on the Globus Toolkit 4, we have built an AstroPortal prototype and implemented a first analysis, "stacking," that sums multiple regions of the sky, a function that can help both identify variable sources and detect faint objects. We have deployed AstroPortal on the TeraGrid distributed infrastructure and applied the stacking function to the Sloan Digital Sky Survey (SDSS), DR4, which comprises about 300 million objects dispersed over 1.3 million files, a total of 3 terabytes of compressed data, with promising results. AstroPortal gives the astronomy community a new tool to advance their research and to open new doors to opportunities never before possible on such a large scale. Furthermore, we have identified that data locality in distributed computing applications is important for the efficient use of the underlying resources. We outline a storage hierarchy that could be used to make more efficient use of the available resources, which could potentially offer orders of magnitude speed ups in the analysis of large datasets.

Keywords: *AstroPortal, web portal, grid computing, astronomy, stacking, SDSS, data locality*

1 Introduction

The astronomy community is acquiring an abundance of digital imaging data, via sky surveys such as SDSS [2], GSC-II [3], 2MASS [4], and POSS-II [5]. However, these datasets are generally large (multiple terabytes) and contain many objects (100 million +) separated into many files (1 million +). Thus, while it is by now common for astronomers to use Web Services interfaces to retrieve individual objects, analyses that require access to significant fractions of a sky survey have proved difficult to implement efficiently. There are five reasons why such analyses are challenging: (1) *large dataset size*; (2) *large number of users* (1000s); (3) *large number of resources* needed for adequate performance (potentially 1000s of processors and 100s of TB of disk); (4) *dispersed geographic distribution of the users and resources*; and (5) *resource heterogeneity*.

We propose to use grid computing to enable the dynamic analysis of large astronomy datasets. The term "Grid" denotes a distributed computing infrastructure for advanced science and engineering. Grid is distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and high-performance orientation [1].

Current grid computing environments are primarily built to support large-scale batch computations, where turnaround may be measured in hours or days – their primary goal is not interactive data analysis. While these batch systems are necessary and highly useful for the repetitive 'pipeline processing' of many large scientific collaborations, they are less useful for subsequent scientific analyses of higher level data products, usually performed by individual scientists or small groups. Such exploratory, interactive analyses require turnaround measured in minutes or seconds so that the scientist can focus, pose questions and get answers within one session. The databases, analysis tasks and visualization tasks involve hundreds of computers and terabytes of data. Of course this interactive access will not be achieved by magic – it requires new organizations of storage, networking and computing, new algorithms, and new tools.

As CPU cycles become cheaper and data sets double in size every year, the main challenge for a rapid turnaround is the location of the data relative to the available computational resources – moving the data repeatedly to distant CPUs is becoming the bottleneck. There are large differences in IO speeds from local disk storage to wide area networks. A single $10K server today can easily provide a GB/sec IO bandwidth, that requires a 10Gbit/sec network connection to transmit. We propose a system in which each 'node' (perhaps a small cluster of tightly coupled computers) has its own high speed local storage that functions as a smart data cache.

Interactive users measure a system by its time-to-solution: the time to go from hypothesis to results. The early steps might move some data from a slow long-term storage resource. But the analysis will quickly form a working set of data and applications that should be co-located in a high performance cluster of processors, storage, and applications.

A data and application scheduling system can observe the workload and recognize data and application locality. Repeated requests for the same services lead to a dynamic rearrangement of the data: the frequently called applications will have their data 'diffusing' into the grid, most residing in local, thus fast storage, and reach a near-optimal thermal equilibrium with their competitor processes for the resources. The process arbitrating data movement is aware of all relevant costs, which include data movement, computing, and starting and stopping applications.

Such an adaptive system can respond rapidly to small requests, in addition to the background batch processing applications. We believe that many of the necessary components to build a suitable system are already available: they just need to be connected following this new philosophy. The architecture requires aggressive use of data partitioning and replication among computational nodes, extensive use of indexing, pre-computation, computational caching, detailed monitoring of the system and immediate feedback (computational steering) so that execution plans can be modified. It also requires resource scheduling mechanisms that favor interactive uses.

We have identified a few possible scenarios from the anticipated use of the National Virtual Observatory [cite] data that are currently used to experiment with this approach. These include image stacking services [cite], and fast parallel federations of large collections [cite]. These experiments are already telling us that, in contrast to traditional scheduling, we need to schedule not just individual jobs but workloads of many jobs.

The key question we answer in this paper is: "*How can we leverage Grid resources to make the analysis of large astronomy datasets a reality for the astronomy community?*" Our answer is "AstroPortal," a gateway to grid resources tailored for the astronomy community. We have implemented our prototype as a Web Service using the Globus Toolkit 4 (GT4) [10] and deployed this service on TeraGrid [8]. The astronomy dataset we are using is the Sloan Digital Sky Survey (SDSS), DR4, which comprises about 300 million objects dispersed over 1.3 million files adding up to 3 terabytes of compressed data.

The AstroPortal prototype supports the "stacking" operation, the summing of image cutouts from different parts of the sky. This function can help to statistically detect objects to faint otherwise.

Astronomical image collections usually cover an area of sky several times (in different wavebands, different times, etc). On the other hand, there are large differences in the sensitivities of different observations: objects detected in one band are often too faint to be seen in another survey. In such cases we still would like to see whether these objects can be detected, even in a statistical fashion. There has been a growing interest to re-project each image to a common set of pixel planes, then stacking images. The stacking improves the signal to noise, and after coadding a large number of images, there will be a detectable signal to measure the average brightness/shape etc of these objects. While this has been done for years manually for a small number of pointing fields, performing this task on wide areas of sky in a systematic way has not yet been done. It is also expected that the detection of much fainter sources (e.g., unusual objects such as transients) can be obtained from stacked images than can be detected in any individual image. AstroPortal gives the astronomy community a new tool to advance their research and opens doors to new opportunities.

2 Background Information & Related Work

This section's purpose is to describe related work similar to the AstroPortal and to cover background material (existing astronomy datasets, the TeraGrid testbed, and science gateways) necessary to make this paper as self contained as possible.

2.1 Astronomy Datasets

Astronomy faces a data avalanche, with breakthroughs in telescope, detector, and computer technology allowing astronomical surveys to produce terabytes (TB) of images. Well-known large astronomy datasets that could potentially be used by AstroPortal include the Sloan Digital Sky Survey (SDSS) [2], the Guide Star Catalog II (GSC-II) [3], the Two Micron All Sky Survey (2MASS) [4], and the Palomar Observatory Sky Survey (POSS-II) [5]. Such astronomy datasets are generally large (TB+) and contain many objects (>200M). For example, SDSS has 300M objects in 10TB of data; GSC-II has 1000M objects with 8TB of data; 2MASS has 500M objects in 10TB of data; and POSS-II has 1000M objects in 3TB of data.

2.2 Related Work: Analysis of Large Astronomy Datasets

Other work is underway to apply TeraGrid resources to explore and analyze the large datasets being federated within the NSF National Virtual Observatory (NVO) [6, 21].. For example, Montage [7] is a portable, compute-intensive, custom astronomical image mosaic service that, like AstroPortal, makes extensive use of grid computing. Our work is distinguished by its focus on the "stacking" problem and by its architecture that allows for flexible and dynamic provisioning of data and computing resources.

2.3 TeraGrid & Science Gateways

TeraGrid [8] is an open scientific discovery infrastructure combining leadership class resources at eight partner sites to create an integrated, persistent computational resource. TeraGrid provides over 40 teraflops of computing power and nearly 2 petabytes of rotating storage, interconnected at 10-30 gigabits/second via a dedicated national network. The initial AstroPortal prototype is deployed at the University of Chicago (UC) site in the TeraGrid. We will deploy future implementation iterations over the entire TeraGrid.

AstroPortal is an example of what the TeraGrid community calls a Science Gateway [9]: an user-oriented problem solving system that makes use of TeraGrid resources to deliver the value of high-performance computing to a large community. Science Gateways signal a paradigm shift from traditional high performance computing use.

3 AstroPortal Architecture

The AstroPortal implementation uses components of the Globus Toolkit version 4 (GT4) [10], and has been deployed in TeraGrid [8]. GT4 components used include WS GRAM [10], GridFTP [12], and WS Core [10]. Our current implementation's focus has been on the AstroPortal functionality as a science gateway, along with the basic resource management functions required for astronomy analysis codes to be run efficiently on large datasets. In the rest of this section, we focus on our use of the SDSS DR4 [13] dataset as the first supported dataset in our prototype deployed on the ANL/UC TeraGrid system.

As shown in **Error! Reference source not found.**, AstroPortal (AP) includes (1) the AstroPortal Web Service (APWS), (3) the Astro Workers (AW) running on the compute nodes, and (4) the Astro Users (AU). The communication between all these components is done using Web Services (WS). Furthermore, we have leveraged GT4 functionality which offers persistent state storage for Web Services; the persistent state makes the APWS more robust to failures as it allows us to continue execution of unfinished jobs after a system restart, bringing the AstroPortal implementation a step closer to being a production ready service.

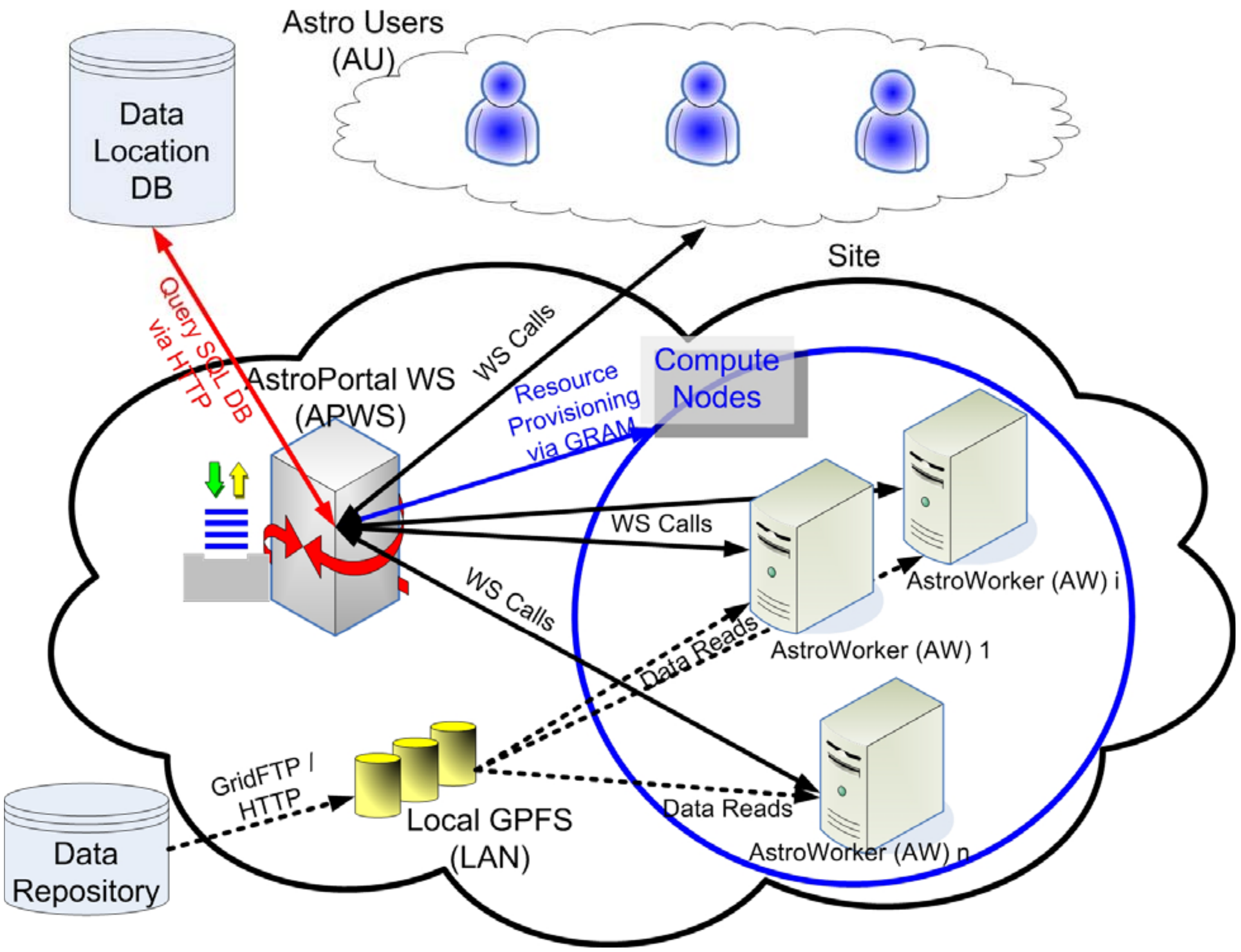

Figure 1: AstroPortal Architecture Overview

The APWS is the main component of the system where the resource management innovation needs to occur to extract the best performance possible from the TG resources; an additional component could be a data manager which would manage the data placement, replication, synchronization, and expose an interface to locate the data with the least expensive access method. Both of these components are rather generic, and with minor tuning, could be used in the analysis of other large non-astronomy related datasets. The AW and AU are specific to the astronomy community, and will offer the analysis and visualization functionality needed make the AstroPortal system useful to astronomers.

APWS is the centralized gateway to which all AUs submit their analysis requests. Once the APWS has started, it could register itself with a well-known MDS4 Index, so that the AU can dynamically find the location of the APWS; however, in the current implementation, the AU must discover the APWS via an out-of-band mechanism. A AU can use any of many existing tools offered by the SDSS/SkyServer [14] to find the location (i.e., sky coordinates – {ra dec band}) of the objects of interest. The AU then sends the list of locations along with the analysis to be performed to the APWS for processing as a job.

The APWS is responsible for dispatching each incoming request to one or more worker resources, which were created via the GRAM API. The number of workers can be varied over time, increasing under heavy loads and decreasing when load reduces. Astro Workers (AW) register with the APWS, and it is the APWS responsibility to notify AW of new work that needs to be completed. Upon the APWS receiving the work from the AU, it finds the necessary data that will need to be accessed to perform the analysis and it notifies the AW that work is available. When the APWS receives the results from an entire job (it could have been fragmented into smaller pieces, with each small piece done independently by multiple concurrent AW), it packages or aggregates them (depending on the analysis) and sends the results back to the AU. Stacking operations normally produce relatively small results, but for larger results, we can return just their location, leaving the actual results to be retrieved via GridFTP, thus avoiding XML processing costs.

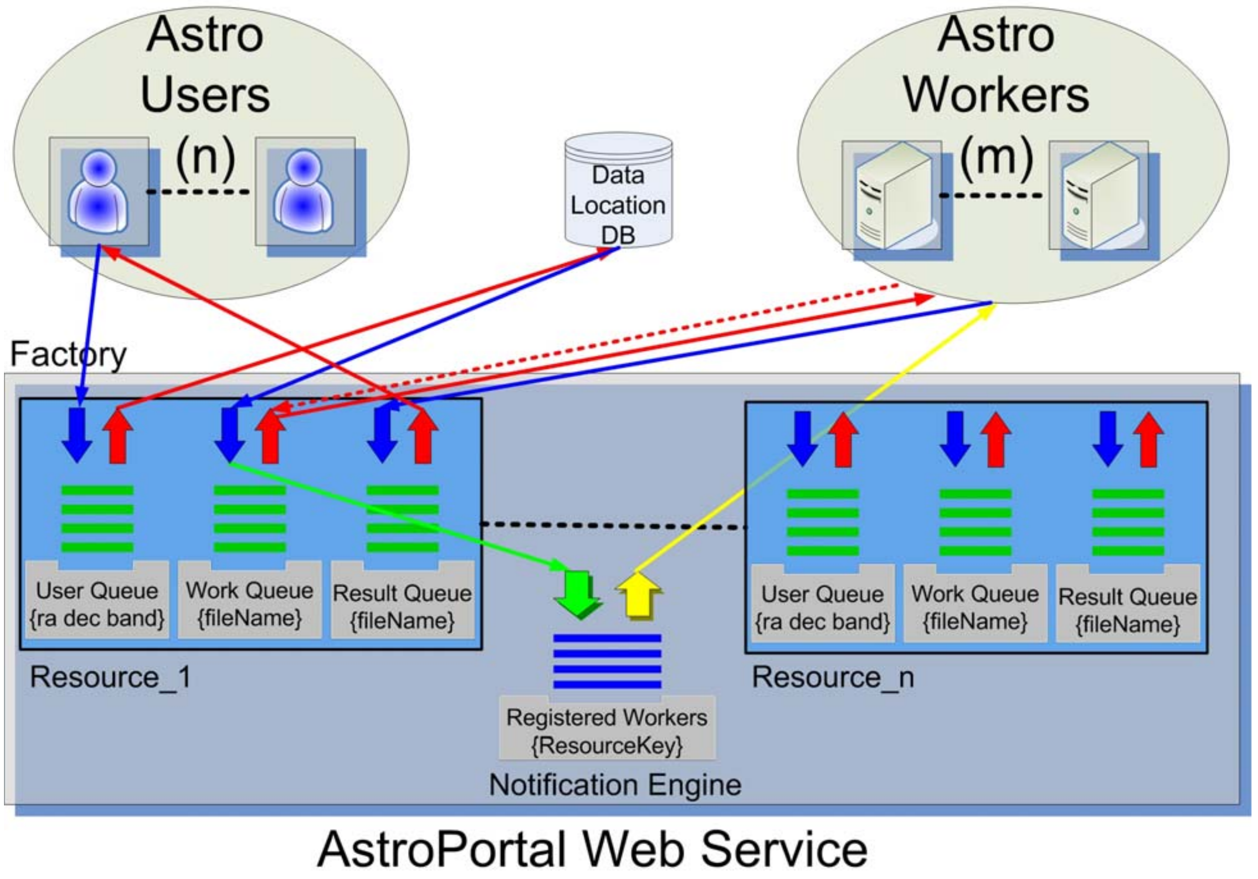

Figure 2: AstroPortal Web Service internal design

AstroPortal's internal design is depicted in Figure 2. The AstroPortal Factory maintains a list of workers that have registered as available for work. (Workers keep their registrations valid by periodic updates.)

Every user creates a WS-Resource before it submits the stacking operation; this design allows for easy client monitoring and management of the submitted work, as each task is represented by a unique WS-Addressing Endpoint Reference (EPR).

Once a user has created a resource to use for the “stacking” jobs, it submits the stacking description to the resource. The APWS places the list of tuples {ra dec band} into the User Queue. A translation thread takes elements from the User Queue and contacts a database to get the data location where the data needed to access the object in question can be found, and the result is placed into the Work Queue. When there is work available in the Work Queue, the Notification Engine sends a notification to some workers specifying which specific Resource has work to get done. Once a worker receives a notification, the interaction is all between the worker and the particular resource that had the work. Once the worker completed an assigned stacking, it sends the result back to the corresponding resource, and the result is inserted into the Result Queue. Meanwhile, the user polls the corresponding resource for the progress of the stacking; when the stacking is complete, the user invokes a WS call to retrieve the result.

3.1 Proposed Storage Hierarchy

Figure 1 from the previous section only showed the current implementation of the AstroPortal, which uses a centralized file system (GPFS) that is accessible by all the compute nodes. We have identified that data locality in distributed computing applications (especially in the stacking service in the AstroPortal) is important for the efficient use of the underlying resources. To increase the efficiency of the utilized resource (and at the same time, improve the observed performance of the AstroPortal), we propose a storage hierarchy that can be used as an integrated part of the AstroPortal.

The storage hierarchy is one of the key design choices that differentiates the AstroPortal from other related work that is part of NVO efforts. The storage hierarchy consists of 4 layers: persistent data repository (HTTP), TeraGrid GPFS (WAN), ANL/UC GPFS (LAN), and local storage (LOCAL). Each storage layer gets the data closer and closer to the computational resources making the analysis faster, but at the same time, each layer is smaller than its slower counterpart. The typical available storage size we expect to have at each layer is: 100 GB of LOCAL space, 1+ TB of LAN space, 10+ TB of WAN space (enough for both compressed and uncompressed datasets), and enough persistent storage to keep the entire dataset of 3TB in compressed format at the HTTP layer.

Figure 3 shows the proposed storage hierarch integrated into the AstroPortal. A new component (the Data Manager) will keep track of usage statistics on each object form the dataset, which will later be used to keep the most likely items in the fastest storage layer (which will also be the most space constrained as well), optimizing the time to access the most popular data. Notice the dotted red line encompassing the all the storage elements (LOCAL, LAN, WAN, and HTTP); this denotes that the Data Manager will be responsible for keeping the entire dataset synchronized; the dataset could be spread over the various storage layers and contain multiple copies of the same data at various layers.

The APWS could use RLS to maintain a coherent state between the replica location among the different layers (LOCAL, LAN, WAN, HTTP). Ideally, as the data gradually flows in (from HTTP, to WAN, to LAN, to LOCAL) as AstroWorkers access the data, stackings would run faster and faster over time. This would be true if we were able to keep the resource reservations of the worker resources indefinitely, which would mean that the data would be accessed more and more over the local disk as the system was used more and more. The HTTP layer will offer persistent storage, the WAN GPFS and local GPFS (LAN) should offer relatively persistent storage, but the LOCAL disk storage will be fairly dynamic (worker resources will start-up and terminate regularly due to advanced reservation policies). In a simple implementation, if the advanced reservations are not long enough, then the LOCAL disk will never get to be populated enough that the system gets jobs reusing the LOCAL data, and hence the system would read most of the upper layers.

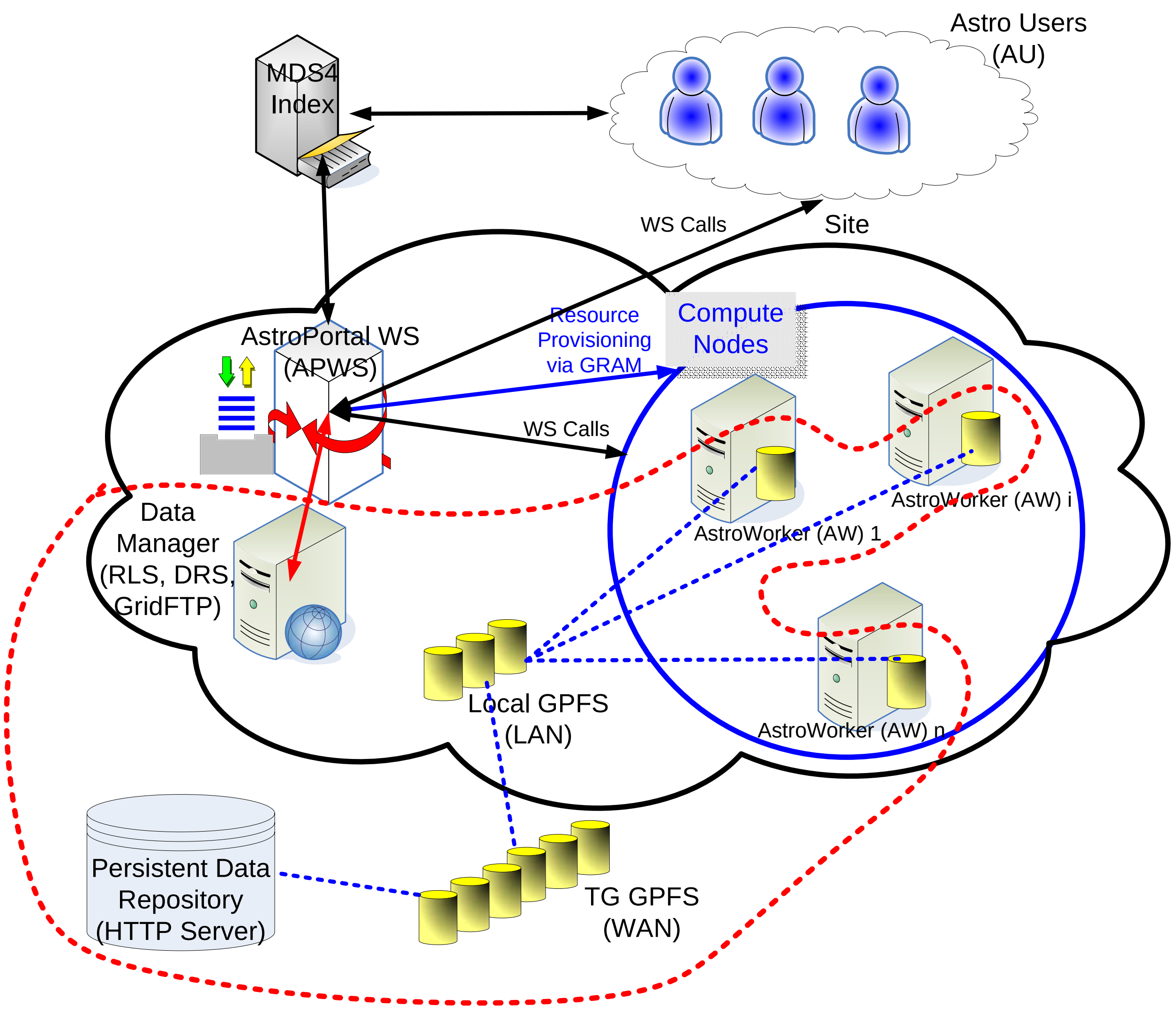

Figure 3: Storage Hierarchy used to build data caches

In order to truly reach that best performance, there would be a need for a worker resource to transfer its state (work queues and locally cached data) from one resource (i.e. node) to another. As time progresses, we could see this transferring of state take longer and longer due to a growing local cache of data. For example, if we had 3TB of data, and 100 clients (distributed over 100 nodes), then each client could potentially have up to 30GB of local cache data, which would take around 10~15 minutes to transfer from one node to another over a 1Gb/s link and modest hard drive read/write speeds of 30~40 MB/s. This is high cost of transferring state can be amortized to a relatively low cost over the life of the running system as long as it does not occur too often; however, this means that the system will not be very dynamic and will not be able to respond to "hot-spots" of large number of queries for a short period of time without being wasteful of resources. Furthermore, when the system is idle because of no new incoming work, the APWS will eventually de-allocate all the reserved resources except for a few. What happens to the state of all the resources that get de-allocated? If the system actually purges the state as well, then we are back to the simple implementation, in which most of the cached data will be deleted when the system is idle, and the performance of the entire site will really be limited to the performance of the upper layers. We believe innovative resource migration mechanisms could help keep the LOCAL layer available for longer periods, and hence improving the likelihood that data is read from the fastest layer (LOCAL) during the majority of the analysis.

3.2 Distributed Resource Management

The distributed version of the APWS is even more interesting with its added complexity offering a more scalable solution. The majority of the intra-site communication remains unchanged, with the exception that the MDS4 Index need not be specifically associated with any particular site. Each APWS from each site would register with the MDS4 Index; when users would query the MDS4 Index, users could pick a possible APWS at random, or based on some simple metrics (i.e. AP WS load, latency from the AP WS to the user, etc) that MDS4 exposes to the users about each AP WS.

The distributed architecture gives us a more scalable system with potentially higher performance. The key to the enhanced performance is the ability to harness all the resources across various sites in the TG; the interaction between the various APWS from each of the various sites is critical. Each APWS would get its share of jobs (or even partial jobs), but depending on what data is needed, and the load at the various sites, jobs could be handled locally within the site, or they could be forwarded to another site that could offer faster performance due to data locality, more available resources, faster hardware, etc. We hope to leverage from previous work done in DI-GRUBER [19], a distributed resource broker based on GT4.

4 AstroPortal Performance Study

We conducted a study of AstroPortal performance (as described in Figure 1). The proposed hierarchical storage implementation has not been completed, and therefore we cannot offer any insight to how well the proposed hierarchical storage would actually perform. However, we have tested the performance of each individual layer in the hierarchical storage, and hence have a very good idea to the kind of performance gains that we should expect to achieve once the hierarchical storage and data manager implementation is complete.

4.1 Worker Performance across Different Storage Hierarchy

The experiment consisted of a single multi-threaded worker (Dual Xeon 2.4 GHz CPUs with 4GB RAM) performing stackings while (1) varying the number of concurrent reading threads from 1 to 10, and (2) varying the access mechanism (compressed - .GZ and uncompressed - .FIT) across the four layers of the storage hierarchy (local disk – LOCAL, ANL GPFS – LAN.GPFS, TG GPFS – WAN.GPFS, and the persistent archive – HTTP). We measured the performance in stackings per second (higher is better); the dotted lines represent the compressed data tests while the solid lines are the results from the uncompressed data. Each stacking was done over a 100x100 sub-image from a 2048x1489 16-bit image that was approximately 2.3 MB in compressed format or 6 MB in uncompressed format. The worker was implemented in Java and relies on a JAVA FITS library [24] that provides efficient I/O for FITS images and binary tables. This library supports all basic FITS formats and gzip compressed files.

Notice that with many parallel reads, there is a clear performance difference between the persistent storage (HTTP.GZ), GPFS (both WAN.GPFS and LAN.GPFS), and the local FS (LOCAL). The most important observation here is that the performance of the LOCAL FS will scale linearly with the number of physical resources used as each machine has its own independent local disk; however, the same thing cannot be said about the shared GPFS and HTTP server. With this said, the performance gap will only widen as the number of utilized resources increases in the AstroPortal.

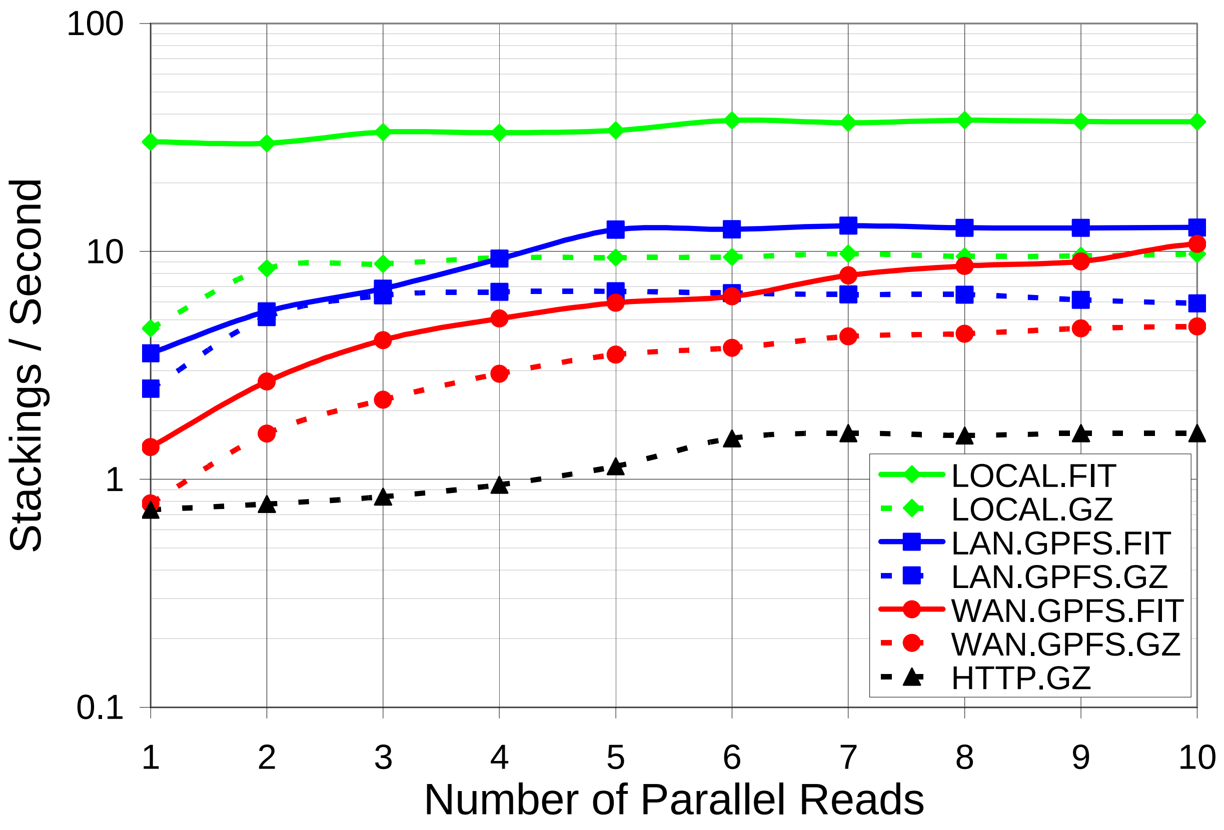

Figure 4: Single worker performance across different storage hierarchy

4.2 AstroPortal Performance on LAN GPFS

The experiment consisted of a single user using the AstroPortal and all its resources. We performed experiments while varying (1) the number of images from 1 to 4096, and (2) the number of workers (Dual Xeon 2.4 GHz CPUs with 4GB RAM) from 1 to 48. Data was accessed on the GPFS parallel file system at Argonne National Laboratory over a LAN (Gbit/s connectivity) in GZIP compressed format.

Error! Reference source not found. shows our performance results of the AstroPortal on the ANL GPFS. Both seem to scale well up to 32 workers, but performance decreases slightly as we increase the number of workers to 48. We tentatively blame the GPFS for this performance limit. Each worker runs multiple parallel stacking threads that perform parallel reads from the SDSS dataset; in our experiments, we had each worker run 10 threads in parallel in order to utilize the local computational resource fully. With 32 concurrent workers and each worker running 10 parallel reads, we have 320 parallel reads from the ANL GPFS, which only has 8 servers running in the back-end. Further tests need to be done before small differences (such as those observed between 32 workers and 48 workers) become statistically significant, especially as the ANL GPFS is a shared file system and could see transient loads generated by other users and services.

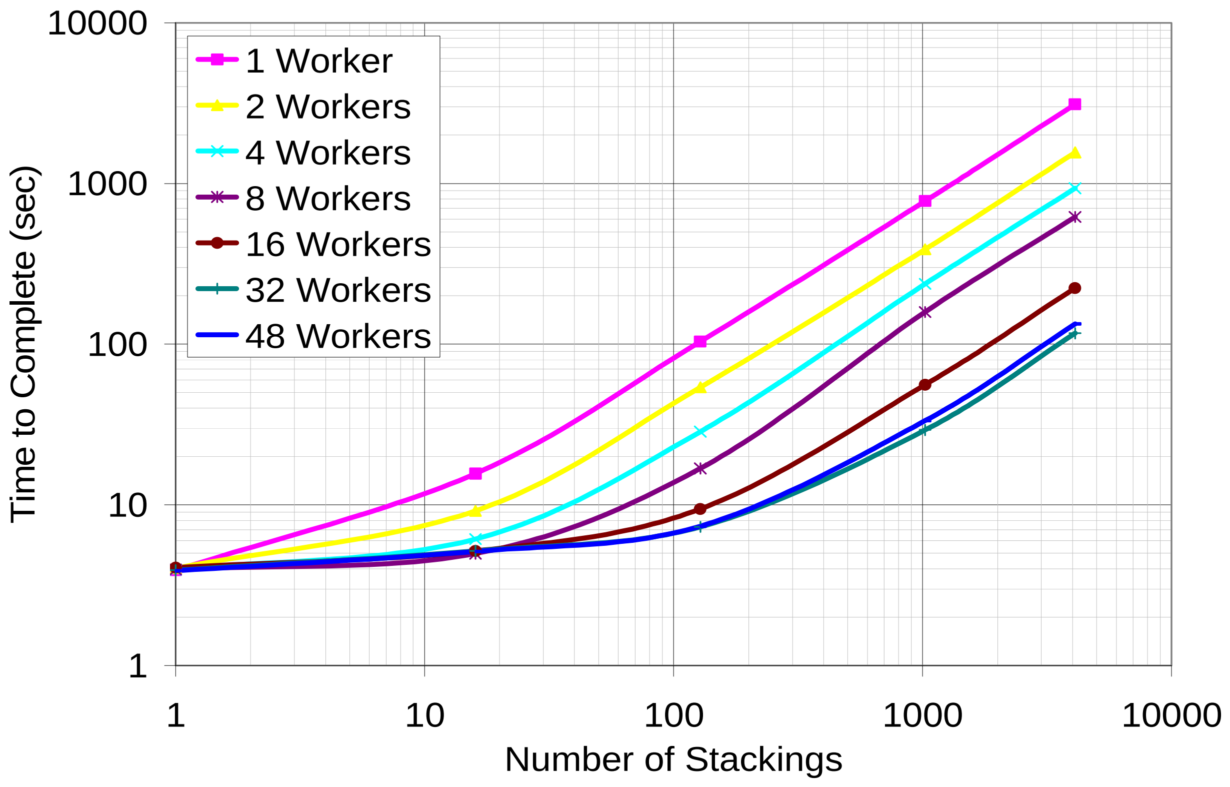

Figure 5: Stacking Performance on compressed images on the ANL GPFS

We see that AstroPortal is most efficient with large number of stackings, when it can take the most advantage of the resulting massive parallelization. For example, we can perform 1024 stackings in under 30 seconds with 32 workers, considering the fact that the 1024 objects needed to be accessed were contained in 1024 files summing to almost 2.4 GB of compressed data. Based on the results shown in **Error! Reference source not found.**, using 32 workers, we could perform a stacking of 1.3 million objects that would touch (reading of 100x100 sub-image from every compressed image) the entire SDSS dataset (1.3 million data files and about 3TB of compressed data) in just over 10 hours. Unfortunately, stacking(s) of 300 million objects (all objects from SDSS) would take about 100 days, a relatively large time period. We expect to see significant improvements in performance on the decompressed dataset; based on the results from Figure 4, we expect at least twice better performance for the same stacking operations over the ANL GPFS without compression. It is worthwhile to mention that the stacking operations are most likely to be I/O bound as many small reads must occur for a single stacking to be completed successfully. Having data that is locally cached at the workers could alleviate the GPFS back-end servers from the large number of concurrent I/O calls, and better utilizing the available disk read performance. We intend to pursue better data access and management techniques in order to get better utilization of the raw available hardware, and at the same time, better performance with faster response times for the users.

Performance with fewer stackings is relatively poorer, presumably due to (inter and intra process) communication among the various AstroPortal components, including the initialization of needed resources (e.g., initializing local state and queue creation, thread creation). One way to improve the performance of small number of stackings is to perform small stackings locally within AstroPortal Web Service, without dispatching it to other resources.

Further analysis is needed to evaluate AstroPortal's ability to handle concurrent users, memory consumption (per user), and robustness in the face of concurrency and failure. For the final version of the paper, we will conduct an extended performance evaluation via DiPerF [17], a DIstributed PERformance testing Framework, that simplifies and automates service performance evaluation. DiPerF coordinates a pool of machines that test a single or distributed target service, collects and aggregates performance metrics from the client point of view, and generates performance statistics (we will collect similar performance metrics at AstroPortal for validation of the obtained results). The aggregate data collected provides information on service throughput, service response time, on service 'fairness' when serving multiple clients concurrently, and on the impact of network latency on service performance.

5 Conclusions and Future Work

The key question we have addressed by the implementation of AstroPortal is: "*How can we leverage Grid resources to make the analysis of large astronomy datasets a reality for the astronomy community?*" AstroPortal is a science gateway to grid resources, tailored for the astronomy community. It gives the astronomy community a new tool to advance their research and opens doors to previously inaccessible opportunities. As the astronomy community uses AstroPortal, we will evolve its design and implementation to optimize it for the particular workloads and access patterns observed.

In summary, the typical distributed computing application we are targeting has (a) some data at a persistent but remote location, (b) a flow of requests for computation on that data, and (c) a set of storage and computing resources that we can allocate dynamically to process requests. This paper focused on the proof of concept of the viability of such systems, as well as identifying the potential improvements that could be made in the future. We have identified that data locality in distributed computing applications is important for the efficient use of the underlying resources. Although it is not directly addressed in this paper, the most interesting innovation will come from the techniques/algorithms/heuristics that optimize some cost metric M by (i) allocating computing and storage resources and (ii) mapping data to storage and requests to computing; the cost metric(s) M still needs to be defined. In a simple case, we will work in a totally reactive mode, in which data is only cached when it is needed. In a more sophisticated instance, we may have knowledge of future requests and/or predictive models of expected future traffic, in which case we can populate the data caches prior to the data actually being needed. In future work, we will define the general models that can be used to drive (i) and (ii) for a wide range of (a), (b), (c), and M.

5.1 Resource Management

We believe that there are at least three main areas with open research problems that the architecture design of the AstroPortal exposes. These areas are all in the broad context of resource management; they include: resource provisioning, data management, and distributed resource management.

Resource provisioning includes everything from advanced reservations, to resource allocation, to resource de-allocation issues in large scale systems. Different techniques and heuristics will apply for managing efficiently the set of resources depending on the problem we are addressing; some of the important things will be workload characteristics, number of users in total and number of concurrent users, data set size and distribution, computational intensive analysis, and I/O intensive analysis. The resource provisioning will be very important in order to achieve efficient use of existing resources, yet maintain a responsive and good performance system.

There are some very interesting problems around data management, in which we have a very large data set that we want to break up among various sites, but also do some level of replication among the sites for improved performance. Furthermore, doing data movement based on past workloads and access patterns might prove to offer significant performance gains. We envision that a new component (the Data Manager from Figure 3) will be defined in the AstroPortal that will keep track of usage statistics on each object from the dataset, which will later be used to keep the most likely items in the fastest storage layer (which will also be the most space constrained as well), optimizing the time to access the most popular data. Another significant challenge will be how to perform efficient state transfer among worker resources while maintaining a dynamic system. The more dynamic the system will be, the less likely it will be that the local (fastest) storage layer will get utilized significantly without an efficient way of keeping the local storage layer in tact despite computational resource migration.

Finally, we see that distributed resource management is one of the answers for improving the scalability of systems such as the AstroPortal. For instance, the TeraGrid has 8 different sites, with locally administered resources. We envision the deployment of the AstroPortal Web Service across every site in the TeraGrid; each APWS would control a set of local resources at a particular site. The inter-site communication among the AstroPortal Web Services and its effects on the overall system performance is very interesting; work can be performed at the local site, or it could be delegated to another site that in theory could complete the work faster; the algorithms, the amount of state information, and the frequency of state information exchanges all contribute to how well and evenly the workload is spread across the various APWS, which ultimately decides the response time that the user observes, and the aggregate throughput the entire distributed AstroPortal system can sustain.

5.2 Other Domains

Although this paper used an astronomy specific application (stacking) and dataset (SDSS), we believe that the exact same architecture and optimizations could be suitable for a wide range of applications that have some data at a persistent but remote location, have a flow of requests for computation on that data, and have a set of storage and computing resources that we can allocate dynamically to process requests. One such application from the medical field is Computer Aided Diagnosis (CAD) that is used to screen large number of patient images for cancers; the data set is very large (millions of images of the human body from different patients), and the analysis (i.e. segmentation, classification, etc) normally performed on these images is both compute intensive and I/O intensive.

6 References

[1] I Foster, C Kesselman, S Tuecke, "*The Anatomy of the Grid*", International Supercomputing Applications, 2001.

[2] SDSS: Sloan Digital Sky Survey, http://www.sdss.org/

[3] GSC-II: Guide Star Catalog II, http://www-gsss.stsci.edu/gsc/GSChome.htm

[4] 2MASS: Two Micron All Sky Survey, http://irsa.ipac.caltech.edu/Missions/2mass.html

[5] POSS-II: Palomar Observatory Sky Survey, http://taltos.pha.jhu.edu/~rrg/science/dposs/dposs.html

[6] R Williams, A Connolly, J Gardner. "*The NSF National Virtual Observatory, TeraGrid Utilization Proposal to NRAC*", http://www.us-vo.org/pubs/files/teragrid-nvo-final.pdf

[7] JC Jacob, DS Katz, T Prince, GB Berriman, JC Good, AC Laity, E Deelman, G Singh, MH Su. "*The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets*", Earth Science Technology Conference, 2004.

[8] TeraGrid, http://www.teragrid.org/

[9] TeraGrid: Science Gateways, http://www.teragrid.org/programs/sci_gateways/

[10] I Foster. "*A Globus Toolkit Primer*," 02/24/2005. http://www-unix.globus.org/toolkit/docs/development/3.9.5/key/GT4_Primer_0.6.pdf

[11] JM Schopf, I Raicu, L Pearlman, N Miller, C Kesselman, I Foster, M D'Arcy. "*Monitoring and Discovery in a Web Services Framework: Functionality and Performance of Globus Toolkit MDS4*", under review at IEEE HPDC 2006.

[12] B Allcock, J Bresnahan, R Kettimuthu, M Link, C Dumitrescu, I Raicu, I Foster. "*The Globus Striped GridFTP Framework and Server*", IEEE/ACM SC 2005.

[13] SDSS Data Release 4 (DR4), http://www.sdss.org/dr4/

[14] Sloan Digital Sky Survey / SkyServer, http://cas.sdss.org/astro/en/

[15] C Dumitrescu, I Raicu, M Ripeanu, I Foster. "*DiPerF: an automated DIstributed PERformance testing Framework*", IEEE/ACM GRID2004, Pittsburgh, PA, November 2004, pp 289 - 296

[16] I Raicu. *"A Performance Study of the Globus Toolkit® and Grid Services via DiPerF, an automated DIstributed PERformance testing Framework*", University of Chicago, Computer Science Department, MS Thesis, May 2005, Chicago, Illinois.

[17] I Raicu, C Dumitrescu, M Ripeanu, I Foster. "*The Design, Performance, and Use of DiPerF: An automated DIstributed PERformance testing Framework*", to appear in the Journal of Grid Computing, Special Issue on Global and Peer-to-Peer Computing 2006.

[18] B Chun, D Culler, T Roscoe, A Bavier, L Peterson, M Wawrzoniak, and M Bowman, "*PlanetLab: An Overlay Testbed for Broad-Coverage Services,*" ACM Computer Communications Review, vol. 33, no. 3, July 2003.

[19] C Dumitrescu, I Raicu, I Foster. "*DI-GRUBER: A Distributed Approach for Grid Resource Brokering*", IEEE/ACM Super Computing 2005

[20] C Dumitrescu, I Raicu, I Foster. "*Extending a Distributed Usage SLA Resource Broker to Support Dynamic Grid Environments*", under review at EuroPar 2006.

[21] US National Virtual Observatory (NVO), http://www.us-vo.org/index.cfm

[22] GGF14 - The Fourteenth Global Grid Forum, Science Gateways: Common Community Interfaces to Grid Resources. http://www.gridforum.org/GGF14/ggf_events_next_schedule_Gateways.htm

[23] SDSC Resource Broker, http://www.sdsc.edu/srb/

[24] JAVA FITS library: http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v0.9/

[25] I. Foster, V. Nefedova, L. Liming, R. Ananthakrishnan, R. Madduri, L. Pearlman, O. Mulmo, M. Ahsant. "Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service." Workshop on Grid Applications: from Early Adopters to Mainstream Users 2005.

[26] Hanisch, R. J.; Farris, A.; Greisen, E. W.; Pence, W. D.; Schlesinger, B. M.; Teuben, P. J.; Thompson, R. W.; Warnock, A., III. Definition of the Flexible Image Transport System (FITS) Astronomy and Astrophysics, v.376, p.359-380, September 2001.