

GridOS: Operating System Services for Grid Architectures

Pradeep Padala and Joseph N Wilson

Department of Computer and Information Science and Engineering

University of Florida

Gainesville, Florida 32611-6120

Email: {ppadala, jnw}@cise.ufl.edu

In the proceedings of the International Conference on High Performance Computing (HiPC'03)

Abstract—In this work, we demonstrate the power of providing a common set of operating system services to Grid Architectures, including high-performance I/O, communication, resource management, and process management. A Grid enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, and specialized devices owned by different organizations administered with different policies. In the last few years, a number of exciting projects like Globus, Legion, and UNICORE developed the software infrastructure needed for grid computing. However, operating system support for grid computing is minimal or non-existent. Tool writers are forced to re-invent the wheel by implementing from scratch. This is error prone and often results in sub-optimal solutions. To address these problems, we are developing GridOS, a set of operating system services that facilitate grid computing. The services are designed to make writing middleware easier and make a normal commodity operating system like Linux highly suitable for grid computing. The modules are designed to be policy neutral, exploit commonality in various grid infrastructures and provide high-performance. Experiments with GridOS verify that there is dramatic improvement in performance when compared to the existing grid file transfer protocols like GridFTP. Our proof-of-concept middleware shows that writing middleware is easy using GridOS.

Index Terms—Grid Computing, Distributed Systems, Operating Systems, OS Services, High-Performance Computing, Linux, Kernel, Modules

I. INTRODUCTION

A Grid[1] enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources and specialized devices owned by different organizations administered with different policies. Grids are typically used for solving large-scale resource and computing intensive problems in science, engineering, and commerce. In the last few years, a number of exciting projects like Globus[2], Legion[3] and UNICORE[4], developed the software infrastructure needed for grid computing. Various distributed computing problems have been solved using these tools and libraries. However, operating system support for grid computing is minimal or non-existent. Though these tools have been developed with different goals, they use a common set of services provided by the existing operating system to achieve different abstractions.

GridOS provides operating system services that support grid computing. It makes writing middleware easier and provides

services that make a normal commodity operating system like Linux more suitable for grid computing. The services are designed as a set of kernel modules that can be inserted and removed with ease. The modules provide mechanisms for high performance I/O (`gridos_io`), communication (`gridos_comm`), resource management (`gridos_rm`), and process management (`gridos_pm`). These modules are designed to be policy neutral, easy to use, consistent and clean. We have also developed modules on top of the above mentioned primary modules that provide high data transfer rates similar to GridFTP[5].

In this paper, we first review the existing toolkits, their common mechanisms to facilitate grid computing and Podos[6] a work similar to ours. Then, we describe the modular architecture of GridOS. Next, we explain the current state of implementation. We conclude by showing the performance results obtained using GridOS modules.

II. PREVIOUS WORK

Primary motivation for this work comes from chapter twenty of the book “*The Anatomy of the Grid: Enabling Scalable Virtual Organizations*”, edited by Ian Foster et al[1], in which the authors discuss the challenges in the operating system interfaces for grid architectures. The book discusses various principles but stops short of implementation details.

While there has been little work on Operating System interfaces, there has been tremendous development in grid middleware. Projects like Globus and Legion provide elaborate software infrastructure for writing grid applications. These tools and libraries have to cope with the existing operating system services that are not designed for high-performance computing. As a result, they are forced to implement some commonly used high-performance optimizations like multiple TCP streams and TCP buffer size negotiation that more suitably should be implemented in the operating system’s kernel. These tools, though quite different, often use the same set of low-level services like resource management, process management, and high-performance I/O. For example, both Globus and Legion have to query the operating system for resources and maintain information for effective resource management. As there is no operating system support for these low-level services, middleware developers must implement them from scratch. This is error prone and often results in sub-optimal solutions.

There have been some attempts to add operating system services that perform high performance computing. WebOS[7] provides operating system services for wide area applications. PODOS[6], a performance oriented distributed operating system, is similar and provides high performance through optimized services in the kernel. This work succeeds in providing high performance. But due to the extensive changes made to the kernel, it is difficult to port this work to newer kernels. It is also difficult to extend the services due to its monolithic structure. GridOS addresses these problems by providing a modular architecture that requires minimal changes to the kernel and makes it easy to extend the services.

Apart from the above mentioned work, there has been great amount of research done in distributed operating systems. Systems like Amoeba[8] and Sprite[9] had great impact on the distributed programming paradigm. We have incorporated some of the principles used in designing these distributed operating systems in GridOS.

III. GRIDOS DESIGN

The following principles drive the design of GridOS. These principles derive from the fact that the toolkits like Globus require a common set of services from the underlying operating system.

- *Modularity.* The key principle in GridOS is to provide modularity. The Linux kernel module architecture is exploited to provide a clean modular functionality.
- *Policy Neutrality.* GridOS follows one of the guiding principles of design of operating systems: policy free mechanisms. Instead of providing a “black box” implementation that takes care of all possibilities, the modules provide a policy-free API which can be used to develop high level services like GridFTP.
- *Universality of Infrastructure.* GridOS provides a basic set of services that are common to prevalent grid software infrastructures. It should be easy to build radically different toolkits like Globus (a set of independent services) and Legion (an object-based meta-systems infrastructure).
- *Minimal Core Operating System Changes.* We do not want to make extensive modifications to the core operating system as that would make it difficult to keep up with new OS versions.

These guiding principles led us to develop a layered architecture (Figure 1). The lowest layer contains the primary GridOS modules that provide high-performance grid computing facilities. These modules are orthogonal and provide basic mechanisms. They do not mandate any policies. The upper layers provide specific services similar to GridFTP, GASS[10].

This approach is motivated by the observation that the toolkits usually make policy decisions on behalf of the grid applications. The toolkit, knowing the application requirements and having domain knowledge, can make a better decision about what policy to employ. This also encourages a wide variety of toolkits to be developed on top of GridOS.

The GridOS core services are provided by an `ioctl` interface. The library `libgridos` provides a C wrapper

to the low-level `ioctl` interface. Toolkit developers should normally be able to use the `libgridos` API for accessing GridOS services.

IV. MODULE DESIGN

In the following sections we describe GridOS modules. We explain the implementation details in a later section.

A. Core Modules

1) *gridos_io: High-Performance I/O Module:* This module provides High-Performance network I/O capabilities for GridOS. In an increasing number of scientific disciplines, large amounts of data (sometimes of the order of peta bytes) are accessed and analyzed regularly. For transporting these large amounts of data, high-speed WANs are used. These networks have high bandwidth and large round trip times (RTT), sometimes referred to as “Long Fat Networks” (LFNs, pronounced “elefan(t)s”) [11].

In order to take full advantage of these high speed networks, various operating system parameters must be tuned and optimized. Several techniques for achieving high-performance are outlined below. It is important to realize that some of these techniques are not additive, but rather complementary. Applying a single technique may have little or no effect, because the absence of any one of the techniques can create a bottleneck.

a) *No User-Space Copying:* A user-space FTP server requires the kernel to copy the network buffers to user-space buffers to retrieve data from the client. It then requires the kernel to copy the user-space buffer to file system buffers to write data to a file. This incurs a large overhead due to time spent in copying.

Because `gridos_io` and `gridos_ftp` are kernel modules that handle both network and file system I/O, thus double copying can be avoided.

b) *TCP WAN Throughput:* TCP (Transmissions Control Protocol)[12] is the standard transport layer used over IP networks. TCP uses the congestion window (CWND), to determine how many packets can be sent before waiting for an acknowledgment. On wide area networks, if CWND is too small, long network delays will cause decreased throughput. This follows directly from Little’s Law[13] that

$$WindowSize = Bandwidth * Delay$$

The Bandwidth is the bandwidth of the slowest link in the path. This can be measured with tools like `pipechar` and `pchar`. The delay can be measured by calculating the round trip time (RTT) using `ping` or `traceroute`.

The TCP “slow start” and “congestion avoidance” algorithms determine the size of the congestion window. The kernel buffers are allocated depending on the maximum size of the congestion window.

To get maximal throughput it is essential to use optimal TCP send and receive kernel buffer sizes for the WAN link we are using. If the buffers are too small, the TCP congestion window will never fully open up. If the buffers are too large, a fast sender can overrun a slow receiver, and the TCP window will

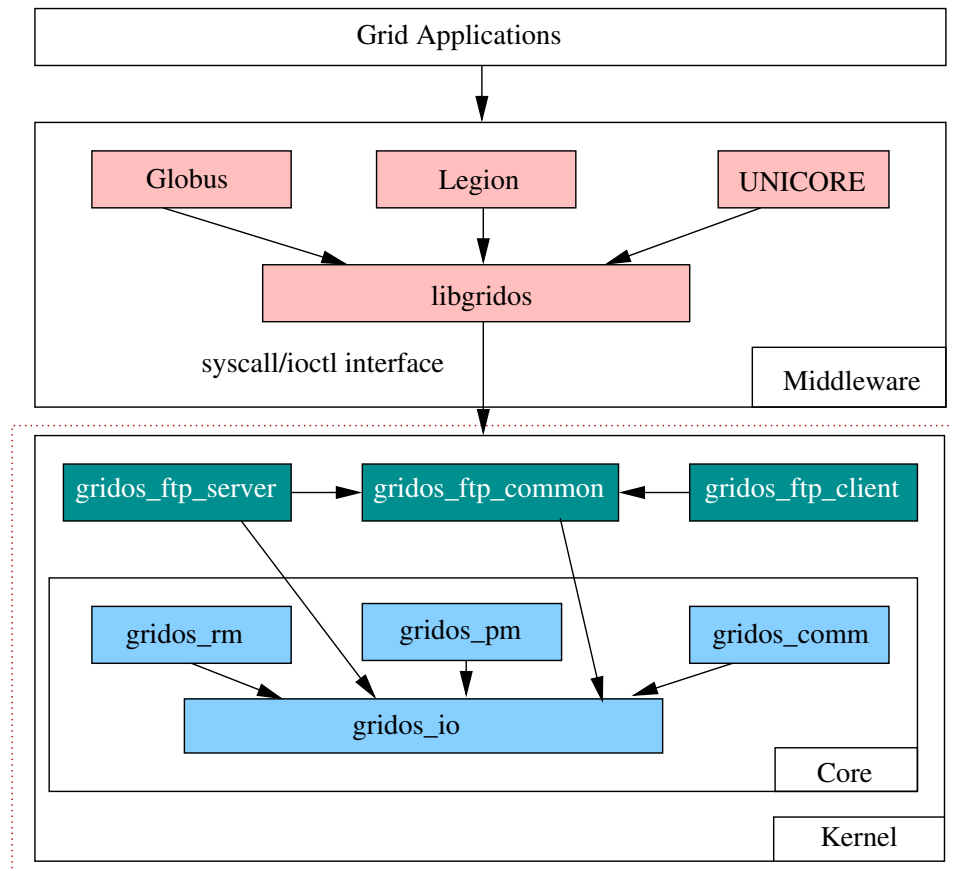


Fig. 1. Major Modules and Structure of GridOS

shut down. The optimal buffer size for a link is bandwidth * RTT[14] [15]

Apart from the **default** send and receive buffer sizes, **maximum** send and receive buffer sizes must be tuned. The default maximum buffer size for Linux is only 64KB which is quite low for high-speed WANs.

The `gridos_io` module provides various ways of controlling these buffer sizes. Using the `gridos` system call, default values for the buffers can be specified. This may be done by the system administrator who has knowledge about the networks. `gridos_io` measures the throughput from time to time and updates the buffer sizes accordingly.

2) *gridos_comm: Communication Module:* This module provides multiple communication methods that support both automatic and programmer-assisted method selection. Grid applications are heterogeneous not only in their computational requirements, but also in types of communication. Different communication methods differ in usage of network interfaces, low-level protocols and data encodings and may have different quality of service requirements. This module allows communication operations to be specified independent of methods used to implement them.

The module also provided multi-threaded communication which is used in implementing the FTP module. Using the library various high-level mechanisms like MPI (Message Passing Interface)[16] can be implemented.

3) *gridos_rm: Resource Management Module:* Grid systems allow applications to assemble and use collections of resources on an as-needed basis, without regard to physical location. Grid middleware and other software architecture that manage resources have to locate and allocate resources according to application requirements. They also have to manage other activities like authentication and process creation that are required to prepare a resource to use. See [17] for details on various issues involved in resource management.

Interestingly, these software infrastructure use a common set of services. `Gridos_rm` provides these facilities so that software infrastructure can be developed to address higher-level issues like co-allocation, online-control etc.

The module provides facilities to query and use resources. The module maintains information about current resource usage of processes started by local operating system and GridOS modules. It also provides facilities to reserve resources. These facilities can be used to develop systems like condor[18], which discovers idle resources on a network.

4) *gridos_pm: Process Management Module:* This module allows creation and management of processes in GridOS. It provides a global PID (GPID) for every process in GridOS and provides communication primitives which can be used on top of `gridos_comm` for processes to communicate among themselves[19].

The module also provides services for process accounting. This feature is important in accounting for the jobs which are

transported to GridOS.

B. Additional Modules

1) *gridos_ftp_common*: This module provides facilities common to any FTP client and server. This includes parsing of FTP commands, handling of FTP responses etc.

2) *gridos_ftp_server*: This module implements a simple FTP server in kernel space. Due to its design, copying of buffers between user and kernel space is avoided. The server does not start a new process for each client as is usually done in typical FTP servers. This incurs low overhead and high-performance. The server also uses *gridos_io* facilities to monitor bandwidth and adjust the file system buffer sizes. The file system buffers are changed depending on the file size to get maximum overlap between network and disk I/O.

The module is designed with security in mind. Even while operating in the kernel mode it drops all privileges and switches to an unprivileged user-id and group-id. It also chroots to FTP top level directory *docroot* which can be configured dynamically.

3) *gridos_ftp_client*: This module implements a simple FTP client in kernel. The main purpose of this module is to decrease the overhead of writing or reading file on the client side. Our experiments indicate that primary overhead on the client side is the time spent in reading and writing files. By carefully optimizing the file system buffers to achieve maximum overlap between network and disk I/O, high-performance is achieved.

V. IMPLEMENTATION

We have implemented a subset of GridOS on Linux using the stable 2.4.20 kernel. The Linux kernel module architecture is exploited to provide ease of installation for users and ease of modification for developers. The code is divided into a small patch to the core kernel and a set of kernel modules. The patch is designed to be minimal and adds the basic *ioctl* and *syscall* interfaces to GridOS.

The modules are inserted and removed using *insmod* and *rmmod* utilities. The dependencies between the modules are resolved using *modules.dep*. As a result, if *gridos_comm* is inserted without inserting *gridos_io* the kernel would find that there is a dependency and insert *gridos_io* automatically.

Currently, we have implemented *gridos_io*, *gridos_ftp_server*, *gridos_ftp_client* and the library wrapper *libgridos*. We have also implemented a simple middleware called *gridos-middleware* as a proof-of-concept showing the ease with which middleware can be developed. The following sections explain the internals of the modules.

A. IO Module

The globus IO module implementation is divided into two APIs, one each for the network and the file system. The IO module is designed to minimize copy operations and let the data flow remain within the kernel.

The network API includes functions to read and write data from a *gridos* managed socket. Both blocking and non-blocking read calls are provided. *Gridos* FTP modules make extensive use of the non-blocking read for high-performance. It also has functions to set various buffer management options.

- *gridos_io_sync_read*: This function is used to read data from a *gridos* managed socket in blocking mode.
- *gridos_io_async_read*: This function is used to read data from a *gridos* managed socket in non-blocking mode
- *gridos_io_write*: This function writes data to the *gridos* managed socket
- *gridos_io_buffer_setopt*: This function sets options for buffer management. The options include setting of TCP send and receive buffer sizes, maximum TCP buffer size etc.
- *gridos_io_buffer_getopt*: This function returns the current buffer management options.

The file system API has similar functions for reading and writing data to files. These functions call the Linux Kernel's VFS (Virtual File System) functions to achieve this. Here is some sample code showing implementation details (simplified for clarity)

The file system API also has a higher-level function *gridos_file_copy* which can be used to copy a file locally. This can be used for high-performance local copying of files.

gridos_io also has facilities to compressed *gzip* data stream. This can be configured by setting the *compression* option.

B. FTP Client and Server Modules

The FTP server API allows the user to create an FTP server on a specified port. Various configuration options like *docroot* (top level directory for anonymous FTP) *threads* (number of simultaneous threads) etc can be set to control the behaviors of FTP module. Dynamic configuration can be done via Linux's *sysctl* mechanism. This provides access to FTP module features through */proc* interface which can be used to read and modify various variables in the kernel address space.

a) *Threading Model*: For high performance kernel threads are used to satisfy simultaneous connections. The thread model has been adapted from TUX, a kernel web server. The model is similar to the FLASH web server[20] where requests are handled simultaneously while managing any disk I/O asynchronously in separate threads.

There are two thread pools. The first pool of threads is I/O or *cache-miss* thread pool. These threads populate the buffers asynchronously at the request of listener threads. The number of I/O threads can be changed by setting the *num_threads* configuration option. All the threads are started at the start of *gridos_ftp_server* module.

The second pool of threads is *fast* or *listener* threads. These threads handle requests and send responses to clients.

C. Library Wrapper

There are three ways of controlling *gridos* behavior from user-space.

```

int gridos_io_file_write(const char *buf, const char *dst, int size)
{
    struct file *f = NULL;
    int flags, len;
    mm_segment_t oldmm;
    int mode = 0600;

    flags = O_WRONLY;
    if(!dst) {
        printk(KERN_ERR "Destination file name is NULL\n");
        return -1;
    }
    f = filp_open(dst, flags, mode);
    if (!f || !f->f_op || !f->f_op->write) {
        printk(KERN_ERR "File (write) object is NULL \n");
        return -1;
    }
    f->f_pos = 0;

    oldmm = get_fs(); set_fs(KERNEL_DS);
    len = f->f_op->write(f, buf, size, &f->f_pos);
    set_fs(oldmm);
    if (f->f_op && f->f_op->flush) {
        lock_kernel();
        f->f_op->flush(f);
        unlock_kernel();
    }
    fput(f);
    printk(KERN_INFO "Wrote %d bytes\n", len);
    return len;
}

```

Fig. 2. File Write Function Listing

- Through system call `sys_gridos`
- Using `ioctl` on `gridos` device
- Using `/proc` interface

`libgridos` provides an easy-to-use interface to these three low level interfaces. Currently, C function wrappers are provided to

- Read and write from/to the network
- Read and write from/to files
- Set configuration options
- Start and Stop FTP server
- Use the FTP client

D. GridOS Middleware

One of our design goals is to ease the development of middleware like `Globus`. We have developed a small middleware which provides uniform access to files whether they are located remotely or locally. The operations are similar to the operations provided by `Globus GASS`. We developed this library in just a few days and it shows how easy it is to develop middleware using `GridOS`.

E. Scenario #1: Transporting a file

In this section, we show module interaction while transporting a file. The file transporting is initiated by a user-space application like `gridos-middleware` that uses `sys_gridos`. Depending on the parameters various `GridOS` modules do the required work. In this case `gridos_ftp_client` calls `gridos_io_file_read`

which reads the file into file system buffers. The buffers are used directly to initiate the network IO. On the server side the network buffers are handed over to `gridos_ftp_server` which can initiate a file writing with `gridos_io_file_write`. Figure 3 shows the scenario in detail. The process of starting the ftp server is not shown in the figure.

F. Scenario #2: Reading and Writing to a file locally

In this scenario, we show the usage of `gridos_file_copy` that can be used to copy a file locally. As there is no copying between user and kernel mode, the performance is improved. Figure 4 shows the scenario in detail.

VI. PERFORMANCE

We have conducted various experiments to measure the performance of `GridOS` comparing it to standard OS services and `Globus` services. The use of `GridOS` services results in a dramatic increase in throughput in our experiments. First, we compare ftp performance using `GridOS` and the standard `proftpd` shipped with `Mandrake Linux`, which showcases the advantages of zero-copy I/O. Then, we compare performance of transporting a file using `GridFTP` server and `GridOS` ftp using `globus-url-copy` as the client. This reveals an important bottleneck in high-performance I/O: file writing overhead. Finally, we compare performance of file transfer using `GridOS` ftp client and server and `globus-url-copy` and `GridFTP` server.

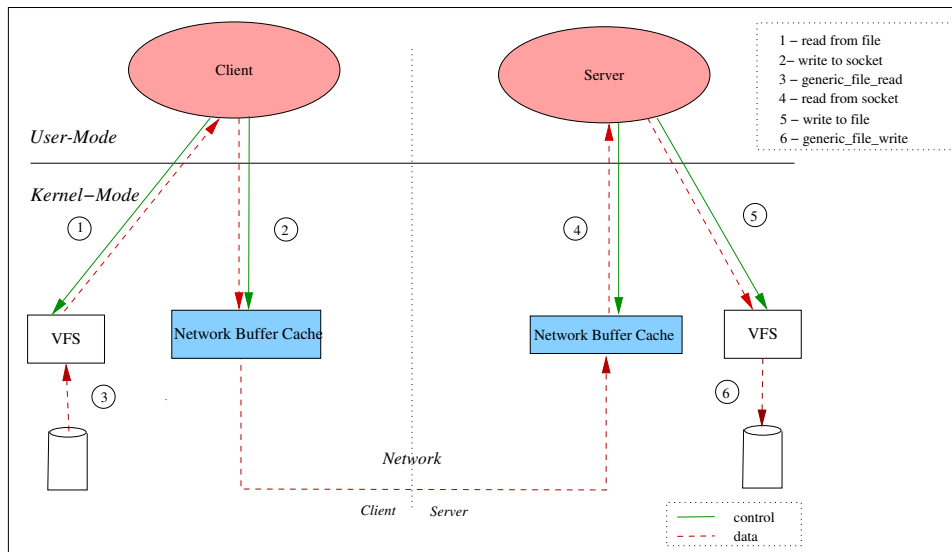


Fig. 3. Transporting a File

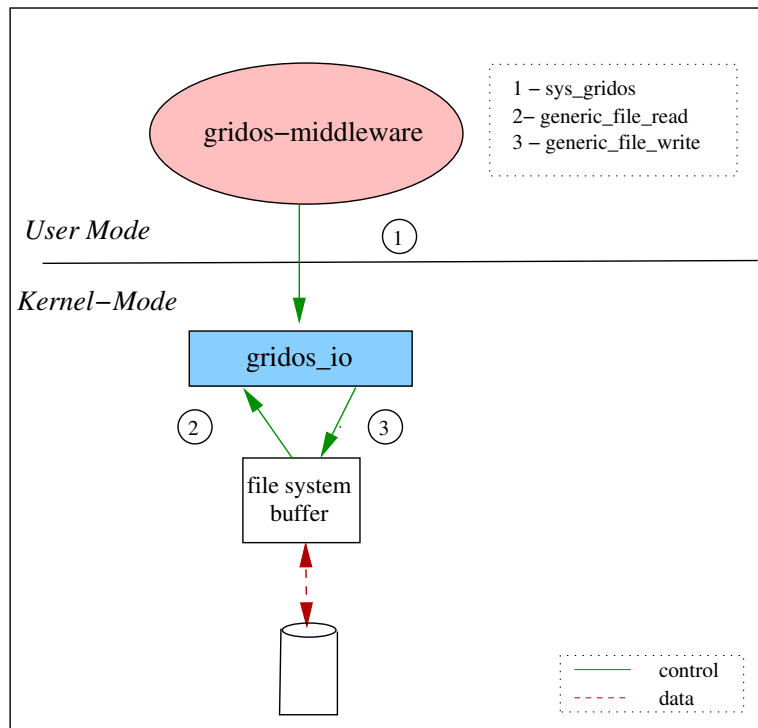


Fig. 4. Reading and Writing to a file

The experiments confirm that on average GridOS is 4 times faster than ftp and 1.5 times faster than GridFTP.

A. Test Environment and Methodology

The tests are conducted in an experimental grid setup in the self-managed network of CISE at UFL. Globus toolkit version 2.2 is used for setting up the grid. The two testing machines were connected over 100Mbps Ethernet network. The machines are kept idle while running the tests so as not to allow other processes running on these machines affect the performance numbers.

In each experiment files of sizes varying from 1KB to 512MB are used and the total time taken to transport each of these files is calculated. For each transfer the average of 5 runs for each file is taken. To avoid the affects of serving from the cache, testing machine is rebooted after each run.

B. GridOS vs Proftpd

In our first experiment, we compared the performance of the proftpd and GridOS ftp servers using the standard ftp client shipped with Mandrake Linux as the client. Results are shown in figure 5. This experiment show-cases the advantages

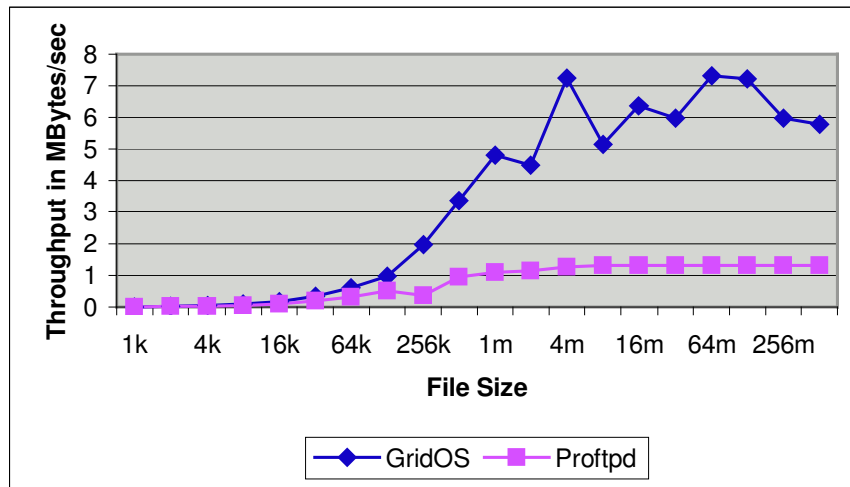


Fig. 5. GridOS vs Proftpd using standard ftp client

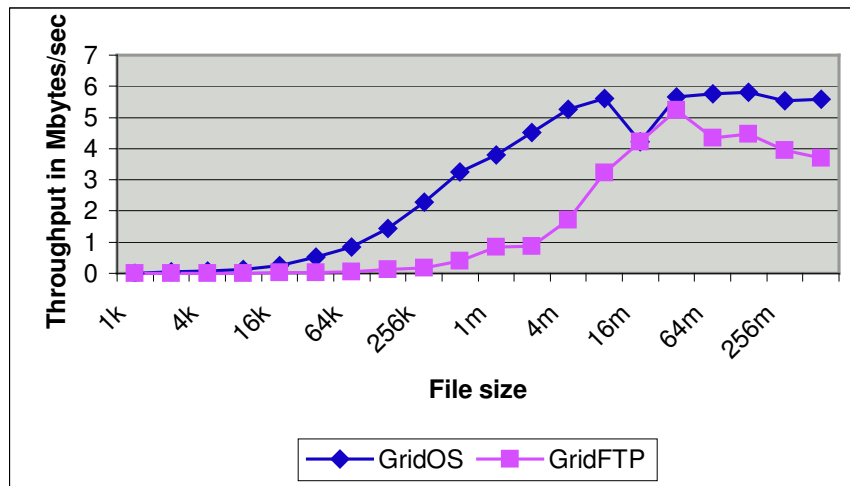


Fig. 6. GridOS ftp vs GridFTP using globus-url-copy as client

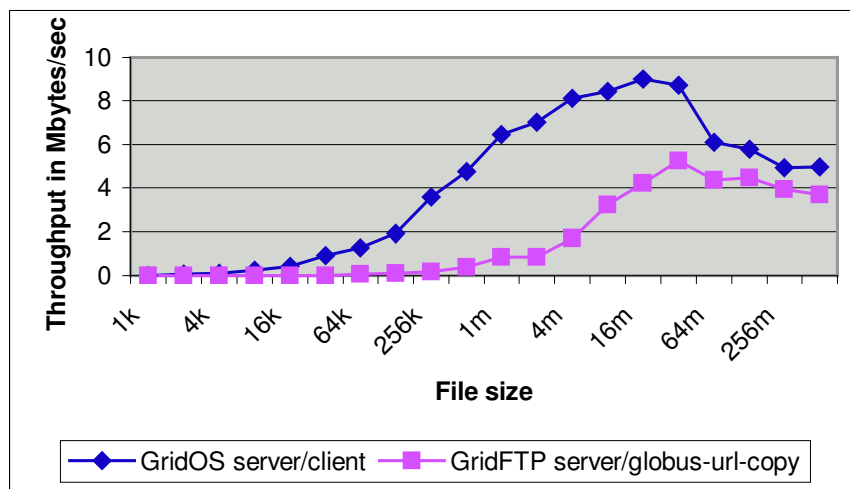


Fig. 7. GridOS ftp server/client vs GridFTP server/globus-url-copy

of serving the files from within the kernel. GridOS consistently performs better than Proftpd for all file sizes.

C. *GridFTP vs GridOS ftp server using globus-url-copy client*

This experiment is done using the `globus-url-copy` client for a fair comparison. Results are shown in figure 6.

GridFTP performs poorly for small files due to the overhead incurred in the performing security mechanisms. GridOS ftp employs the standard ftp password authentication for security.

We also conducted experiments using standard ftp client instead of globus-url-copy for GridOS. Performance was poor compared to the performance obtained using globus-url-copy. Globus-url-copy is specifically designed for high-performance computing and has better buffer management. This led us to develop an in-kernel ftp client.

D. GridFTP server/client vs GridOS ftp server/client

In this experiment, globus-url-copy and gridos-ftp-client are used as clients for GridFTP and GridOS ftp server respectively. GridOS ftp client makes effective buffer management and is designed on the same lines as globus-url-copy. Results are shown in figure 7. Based on our experiments we observe that the performance drop for larger files is due to the overhead involved in file writing.

VII. CONCLUSIONS AND FUTURE WORK

We have described a set of operating system services for grid architectures. These services make a commodity operating system like Linux highly suitable for high-performance computing. We have identified a common set of services that use grid software infrastructures like Globus, Legion, Condor etc. The services are developed as a set of modules for ease of use and installation. Our performance experiments show that high-performance can be achieved with GridOS modules. We have also described a proof-of-concept middleware that uses GridOS facilities.

The work presented here is a first step towards a grid operating system which provides extensive, flexible services for grid architectures. Our next step is to implement other GridOS modules. We have plans to deploy it in a wide area computing testbed. Our first target is the grid used by GriPhyN (Grid Physics Network)[21] at the University of Florida. This environment will enable us to evaluate GridOS modules more realistically. We also plan to port Globus libraries to GridOS thus providing a complete software infrastructure for grid architectures.

ACKNOWLEDGMENTS

The authors would like to thank Ravi Parimi and Shashank Shetty for their valuable comments.

REFERENCES

- [1] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
- [2] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.
- [3] A. S. Grimshaw, W. A. Wulf, and the Legion team, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, no. 1, pp. 39–45, Jan. 1997.
- [4] V. Huber, "UNICORE: A Grid computing environment for distributed and parallel computing," *Lecture Notes in Computer Science*, vol. 2127, pp. 258–266, 2001.
- [5] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, no. 5, pp. 749–771, May 2002.
- [6] S. Vazhkudai, J. Syed, and T. Maginnis, "PODOS — the design and implementation of a performance oriented Linux cluster," *Future Generation Computer Systems*, vol. 18, no. 3, pp. 335–352, Jan. 2002.
- [7] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa, "WebOS: Operating system services for wide area applications," in *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, July 1999.
- [8] A. S. Tanenbaum and S. Mullender, "An overview of the Amoeba distributed operating system," *Operating Systems Review*, vol. 15, no. 3, pp. 51–64, July 1981.
- [9] J. K. Ousterhout, A. R. Cherenon, F. Douglass, M. N. Nelson, and B. B. Welch, "The Sprite network operating system," *Computer*, vol. 21, no. 2, pp. 23–36, Feb. 1988.
- [10] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: A data movement and access service for wide area computing systems," in *Proc. IOPADS'99*. ACM Press, 1999.
- [11] W. R. Stevens, *TCP/IP Illustrated, Volume 1; The Protocols*. Addison Wesley, 1995.
- [12] J. Postel, "RFC 793: Transmission control protocol," Sept. 1981.
- [13] L. Kleinrock, *Queueing Systems: Theory*. John Wiley and Sons, 1975, vol. 1.
- [14] J. Semke, M. Mathis, and J. Mahdavi, "Automatic TCP buffer tuning," *SIGCOMM 98*, 1998.
- [15] B. L. Tierney, J. Lee, B. Crowley, M. Holding, J. Hylton, and F. L. Drake, Jr., "A network-aware distributed storage cache for data-intensive environments," in *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*. Redondo Beach, CA: IEEE Computer Society Press, Aug. 1999, pp. 185–193.
- [16] M. P. I. Forum, "MPI: A message-passing interface standard, Tech. Rep. UT-CS-94-230, 1994. [Online]. Available: citeseer.nj.nec.com/article/forum94mpi.html
- [17] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag LNCS 1459, 1998, pp. 62–82.
- [18] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor : A hunter of idle workstations," in *8th International Conference on Distributed Computing Systems*. Washington, D.C., USA: IEEE Computer Society Press, June 1988, pp. 104–111.
- [19] P. T. Maginnis, "Design considerations for the transformation of MINIX into a distributed operating system," in *Proceedings, focus on software / 1988 ACM Sixteenth Annual Computer Science Conference, February 23–25, the Westin, Peachtree Plaza, Atlanta, Georgia*. ACM, Ed. New York, NY 10036, USA: ACM Press, 1988, pp. 608–615.
- [20] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An efficient and portable web server," in *Proceedings of the 1999 USENIX Annual Technical Conference (USENIX-99)*. Berkeley, CA: USENIX Association, June 6–11 1999, pp. 199–212.
- [21] P. Avery and I. Foster, "The griphyn project: Towards petascale virtual data grids," 2001. [Online]. Available: <http://www.griphyn.org>