WILLIAM ALLCOCK AND JOHN BRESNAHAN

Maximizing Your Globus Toolkit™ GridFTP Server

GridFTP is a protocol that provides support for the secure, fast, efficient, and robust transport of data. The protocol currently holds the status of a Global Grid Forum recommendation (GFD.020). One of the most common implementations of the GridFTP protocol is provided in the Globus Toolkit™, available from the Globus Alliance, an open source project with a very liberal license. The most current version of the toolkit is V3.2.

The GridFTP protocol builds on three IETF RFCs: RFC 959 provides the base FTP protocol, RFC 2228 provides a mechanism for adding security via the GSS-API, and RFC 2389 provides a feature negotiation mechanism and an OPTS command to set options for other commands. GridFTP also builds on an IETF draft from the FTP working group that provides functions to get the file size and modification time, restart capabilities to stream mode transfers, and structured directory listings. To these IETF features GridFTP adds support for a new mode (extended block mode, or MODE E), which allows out-of-order data reception, thereby enabling multiple data paths for speed and efficiency. The SPOR and SPAS (striped PORT and striped PASV) commands allow for the return of multiple IP addresses. These, in conjunction with MODE E, enable multiple hosts to participate in the transfer of a single file, so that performance greater than a single machine can achieve is possible. Restart markers for robust error recovery in MODE E are provided, as are commands to allow the data channel to be secured. Other new commands allow for processing of the data at the server (we implement partial file transfers with this mechanism). Moreover, and of particular relevance to the subject at hand, GridFTP includes a mechanism for setting the TCP buffer sizes, both manually and automatically.

Despite the numerous features GridFTP provides, a common attitude is: "Yes, security is important, and it's cool that I don't have to start from scratch if the transfer fails, and the other features are great, but how do I make it go FAST?" We address this question here.

How Fast Is Fast Enough?

Whether troubleshooting or trying to define features and specifications for future versions of GridFTP, we find it frustrating to hear, "I want to go as fast as possible." Fine; then buy a dedicated 10 gigabit Ethernet (GigE) link from your desk to the destination you want to reach; install a fast storage attached network (SAN); get the biggest machine possible; buy a 10 GigE network interface card (NIC); rack up a debt larger than many small nations; and you can go as fast as possible. On the other hand, if you say, "My application needs to move X amount of data in Y amount of time," then you can calculate your bandwidth requirements. Yes, X and Y may be arbitrary, but at least they are based on some data. Or you may simply want to run at whatever is considered "normal" or "reasonable" performance, getting kind of bandwidth other sites similar to yours are achieving. If you want to get 500 Mbs (Megabits per second) disk to disk, over a GigE link, that's possible with the right hardware. If you want 750 Mbs, you are going to have to work at it. If you want 900+ Mbs, it is not likely to happen over your LAN, let alone in the wide area (see the discussion on TCP for an explanation). The bottom line is: Know what you are shooting for, and have a reason for it. It will help you make decisions later on.

The Weakest Link

In a BBC TV game show, people must answer tricky questions; at the end of each round, one of the contestants is voted off, and a woman with a nasal voice proclaims, "You are the weakest link ... GOODBYE." Well, a large part of making GridFTP go fast is basically engineering to say "Goodbye" to the weakest link in your hardware chain. Let us state the painfully obvious: Your performance is only as good as the slowest component. Unfortunately, it is not always obvious which component that is.

In the next section, we review the various hardware subsystems and look at the impact each has on the performance of a GridFTP server. While this article is specifically about the Globus Toolkit V3.2 GridFTP server, the hardware discussion should apply equally well to any bulk data transport application.

The Disk Subsystem

Disk is one of the most commonly overlooked culprits of poor performance in a GridFTP server. We regularly get mail complaining about poor performance, and we've noticed that one magic number pops up a lot: 160 Mbs. Our normal response in such cases is, "You wouldn't happen to be running a single IDE disk on the machine, would you?" You see, you can have screaming CPUs, a fast network, and all that other stuff, but performance is about moving data from a file on one disk to a file on another disk. If your typical IDE

disk subsystem is limited to about 20 MB/s (MegaBytes per second), then that is the best performance you are going to get. So, what to do?

The answer is RAID. For those not familiar with the term, it stands for Redundant Array of Inexpensive Disk. The Web is replete with references on RAID, and there are numerous books on the subject, but the basic concept is straightforward. When you write to a disk, you write a series of "blocks" of data. In RAID, you take multiples of these disks; make them look like one big disk; and then, when you write this series of blocks, you tell the first drive to write the first block, the second drive to write the second block, and so on. The idea is that if you have enough drives, by the time you get back around to the first drive, it should be done writing its block and be waiting for the next one. This strategy allows much higher disk I/O rates than a single drive can achieve. Technically, this is called RAID 0, or striping. There are other levels of RAID, the most common being RAID 5, which adds redundancy, so that one of the drives can fail without losing data. How that works is beyond the scope of this article. Simply be aware that redundancy is gained at the cost of speed and additional disks. As someone once said, "Price, speed, robustness: pick any two."

Various RAID solutions are available. Linux includes something called software RAID, in which basically all of the RAID functionality is executed on the host CPU in software. This approach has the advantage of being cheap (the costs are disks and your time), but it can be quite CPU-intensive. One of the hosts on which we work has four (expensive) 15K RPM SCSI disks with Linux software RAID, and we achieve about 125 MB/s sequential read Performance — at the cost of 100 percent CPU utilization on a 1 GHz Pentium III. The alternative to software RAID is ... you guessed it, hardware RAID. In this case, all of the work of controlling the disks is handled by specialized hardware installed in the host. Some hosts actually include HW RAID support on board. For example, we have a set of clusters based on Compaq DL380G2s that have on board HW RAID. Each node contains six 36 GB 10K RPM SCSI disks, and we achieve about 138 MB/s sequential read performance with only 20 percent CPU utilization on a 1.13 GHz Pentium III. The early RAID solutions all involved SCSI disks. Recently, however, controllers for non-SCSI drives have been making inroads into the traditionally SCSI-dominated world. Controllers are also available for IDE disks. These have the advantage of using the cheapest disks, but they tend to have the lowest performance. Serial ATA, or SATA RAID, is rapidly gaining in popularity, and SATA RAID controllers are available from a number of different vendors. They provide a nice compromise between cost and performance.

But what if you don't have space or money for more drives? How can you improve the performance of your current disk drive? The following tips and tricks are taken primarily from the system tuning guides found at Red Hat and Linux.com. URLs are provided in the resources sidebar.

Tune the file system by adjusting the bdflush parameters:

```
echo 100 5000 640 2560 150 30000 \
  5000 1884 2 > /proc/sys/vm/ \
    bdflush
```

The cited parameters are recommended for file servers. However, you should check the parameter description and run tests to see what parameters work best for you. Also, disabling the access time updates on the filesystem will speed disk accesses. This setting is less of an issue if you move a few large files, but if the server has other applications that do many disk access, this can have an effect. Simply add the *noatime* option to `/etc/fstab`:

```
/dev/rd/c0d0p3  /test  \
  ext2  noatime  1 2
```

Linux 2.4 also allows you to trade disk throughput for latency, and vice versa, by adjusting the disk I/O elevators. Larger numbers generally mean more throughput but higher latency. You make these adjustments with the elvtune command:

```
/sbin/elvtune -r <sectors> \
  -w <sectors> /dev/hda1
```

You can determine the current settings via:

```
/sbin/elvtune /dev/hda1
```

Again, you need to experiment to determine which parameters are best for your site. Finally, if you are using only one partition on your disk drive, most modern disks can access some sectors faster than others. Usually, this is the lower numbered sectors — namely, the first partition defined, but not always. The bonnie++ disk performance benchmark has a test to determine which sectors are fastest. Picking the fastest partition for your data can help improve your server performance.

In summary, if disk is your bottleneck, a few tweaks to Linux might gain you some performance on GridFTP (at the potential cost of hurting latency-critical applications). But for the most part, it is a question of money — money for more disks and, potentially, money for more processing power, in the form of faster CPUs for software RAID or custom ASICs on a hardware RAID controller. On the other hand, a few hundred dollars of disks

and software RAID should give you a substantial performance boost, and a few thousand dollars should provide you 1TB+ of disk space on a hardware RAID controller that is fast enough to keep up with the performance you can achieve on a gigabit Ethernet link. We note, however, one important caution with HW RAID. The quality of the controllers varies widely, and the performance can depend on the combination of controller and motherboard. The performance numbers that you are quoted will tend to be the best possible under the best possible circumstance, and you are not likely to match them, so doing some testing with a loaner is a good idea.

Other, more expensive solutions to this problem do exist. Direct attached storage (DAS) boxes, for example, use fiber channel to connect large numbers of RAIDed disks (16 is common). These external DAS boxes typically cost $5,000 to $10,000, depending on the number of disks. There are also SAN solutions (a SAN looks like a block device but is network attached), and Network Attached Storage (NAS) solutions (a NAS is a file serving device). However, installing these devices is usually a significant engineering task, often done for a site or entire department; it costs tens or even hundreds of thousands of dollars and thus seems a little extreme just to make your GridFTP server run faster.

CPU Speed

Current dual-processor, multi-GHz CPUs should have plenty of power to drive a GridFTP server. Older machines, however, may end up being CPU bound, particularly if they are running SW RAID. As a rule of thumb, a GridFTP server will consume the equivalent of a dual processor 1 GHz Pentium III machine if you are running SW RAID. Basically, one CPU drives the disks, and one drives the GigE NIC. It is critical to remember that the CPU is servicing a lot of interrupts when you are trying to move data that fast. We have not yet tried running GridFTP with a 10 GigE NIC, but it is almost a guarantee that either the disk or the CPU will be the bottleneck. Note, too, that no cycles are left for other processing. In general, if you really want good GridFTP performance, the box should be dedicated to GridFTP. Besides avoiding context switches, it allows you to tune certain system parameters that might have a negative impact on other applications (such as MPI-based programs).

The Network

If you have a 100 Mbs NIC in your host, the best you will get is 100 Mbs. That seems Obvious, doesn't it? Gigabit Ethernet NICs are readily available, and many (most?) machines now come standard with them. If you are using an add-in card, however, you should be aware that because of HW timing-related issues, certain NICs work better with some motherboards than with others, so testing an evaluation unit is worthwhile, or simply swapping out your current controller for a new one might buy you some performance. Though today it's less of a problem than it used to be, some NICs do not auto-sense correctly and need to be forced into full duplex mode. How to do this varies, and you will need to check the documentation for your NIC. Also, keep in mind that processing an interrupt every time a packet arrives on a GigE NIC can consume a lot of CPU and hurt your performance. Interrupt coalescing, that is, having the NIC wait some period of time before issuing the interrupt to see whether another packet arrives, can have a significant impact on bulk data transport. It can also have a significant negative impact on applications such as MPI that are latency critical. Again, you need to read the documentation to see how to adjust this (if it can be adjusted at all) and test with your system to see what the right balance is.

Unfortunately, you can have a fast machine, fast disk, lots of RAM, GigE NIC, everything tweaked, connected to a fast network — and still not achieve decent performance. The reason is an insidious problem, often hidden in the network, called the "last mile" problem. This situation is another example of our weakest link metaphor. Specifically, the ultra-fast network to which you are attached is not plugged directly into your GigE NIC. There is your internal corporate or campus infrastructure to think about. Many networking people either don't know about this or are so hurried they don't take the time to understand your problem and give you an accurate explanation. For instance, if you ask your network folks, "What is the slowest link in the path from my desktop to host x.someotherdomain.edu?" the answer you may get back is "minimum GigE all the way." Technically, that may be right. However, if that GigE path happens to include a GigE hub (multiple GigE connections in, but only one GigE out), or if the entire campus traffic is sharing the same GigE backbone, you are not going to get gigabit speeds.

If you are interested in the performance of your GridFTP transfers, then your network connection is a critical component. It is worth the time to sit down with your networking people, explain what you are doing, and get a detailed explanation of every piece of hardware you pass through from your NIC to the exit router. (Note that if you run to multiple places, you could take multiple different paths, even within your facility.) What is its capacity? How much is it shared? What is the typical utilization, peak utilization, and so on? These are questions to seed the conversation. Unless you are in-

timately involved with networking, you probably don't know all the right questions to ask. Get your networking people involved. Get them to understand what you need to do, with realistic expectations (see "How Fast Is Fast Enough," above). Then, not only can they tell you what you have today, but they may be able to simply switch some things around in the wiring closet and help you out. If not, they can help you plan for the infrastructure changes necessary to meet your goals.

Firewalls

Firewalls are not evil. People are sometimes evil, and we use firewalls to keep such people from intruding on our peace and harmony. That being said, when it comes to network performance, firewalls are ugly. Because they must inspect every packet, they tend to slow the traffic. Limits of a few hundred megabits are not uncommon, although modern firewalls are purported to be able to keep up with GigE flows (we have not tested this for ourselves). Find out whether you have a firewall. If so, check to see whether there is a way to get TCP streams through without being processed by the firewall. If not, you are pretty much stuck. Note that since the Globus Toolkit implementation of GridFTP defaults to data channel authentication, and you can control the range of ports used via the *GLOBUS_TCP_PORT_RANGE* environment variable, opening holes in the firewall is less of a security risk than it might be for other applications.

TCP Buffer Size

TCP guarantees the application reliable, in-order reception of the data. When an application calls `write()` on a socket, and it returns, this indicates that the data has been transferred to a buffer in the kernel, not that it has been received at the other end. The kernel has to hold a copy of all the data that is sent out over the network until it receives an acknowledgment from the other side that it has received that data. This behavior is because the network may drop that packet, which then may need to be retransmitted. The size of the buffer that the data is held in pending acknowledgment controls the maximum bandwidth achievable. A finite amount of time is required for a packet to travel to its destination and its acknowledgment to return. This increment is called the round-trip time (RTT). The RTT in your LAN is probably less than 1 ms. The RTT from Chicago to the West Coast is on the order of 50-60 ms, and Chicago to Amsterdam is around 100 ms. To make the math easy, assume you can transmit 1 packet per ms. In your LAN, you would need room for only one packet because the acknowledgment would have arrived before you transmitted the next packet. For a West Coast transfer, you would need space for 50 or 60 packets; for Amsterdam, approximately 100. If you had space for only 50 packets and were trying to send to Amsterdam, you could transmit 50 packets but then would have to wait for an acknowledgment of the first packet before you could send the 51st, and so on. So, in order to be sure you can achieve your desired bandwidth, you need to make sure the TCP buffer is big enough. To calculate the size of the buffer, you need to multiply the desired bandwidth times the RTT (with appropriate units conversions). This is called the bandwidth delay product. A simple equation is

```
Buffer size (KB) =
  Bandwidth (Mbs) *
    RTT (ms) / 8
```

This equation assumes that kilo and mega are based on 1000, rather than 1024. In `globus-url-copy`, the TCP buffer size is set via the `-tcp-bs` command line option. Note that in the discussion of parallel streams below, we adjust this calculation to account for the fact that the bandwidth is divided across multiple streams.

You also need to make sure that Linux is configured to allow the buffer size you request. If you request a buffer size larger than is available, it is not an error; you simply get the largest allowed. To increase the absolute buffer limits, add the following to `/etc/sysctl.conf`:

```
net.core.rmem_max =
  <max read buffer size (bytes)>
net.core.wmem_max =
  <max write buffer size (bytes)>
net.core.rmem_default =
  <default read buffer size
    (bytes)>
net.core.wmem_default =
  <default write buffer size
    (bytes)>
```

To increase the auto-tuning limits, add:

```
net.ipv4.tcp_rmem = <min>
  <default> <max>
net.ipv4.tcp_wmem = <min>
  <default> <max>
net.ipv4.tcp_mem = <min>
  <default> <max>
```

Note that the first two are in bytes but the last is in pages. For more detail, see the Berkeley TCP tuning guide listed in the resources.

Parallelism

Parallelism means that there are multiple TCP connections, rather than one. Data is sent as fast as possible down each stream, with the fastest streams getting the most data. This strategy works well in light to moderate congestion over high-latency networks. How many streams to use is environment dependent, but four seems to be a good rule of thumb. Parallelism won't

help on a LAN (low latency); and if you are lucky enough to be on a WAN that is not dropping any packets (not likely), it won't help you either. Since you are using multiple channels, your bandwidth is divided across them, so each stream only needs a buffer big enough for the bandwidth it will carry. As a rule of thumb, I suggest you calculate the buffer size as above and then divide by n-1, where n is the number of streams. The "-1" accounts for the fact that some streams may be faster than others, and this will keep the buffer size from limiting the bandwidth. The reason for not using the full buffer size is that if you have large buffers (12 MB is necessary for GigE from Chicago to Amsterdam) and many streams, you can run the system out of memory. For instance, in the 12 MB example, 10 streams would use nearly half the memory on a host with 256 MB of RAM.

Why does this work in such a way? In brief, when there is too much traffic on the net, the routers drop packets. When TCP sees that it has lost a packet (i.e., it believes there is congestion), it responds by cutting its bandwidth in half. By dividing the bandwidth across multiple streams, you lessen the impact of a lost packet. For instance, if you were running at a full gigabit, 1000 Mbs, a lost packet would slow you down 500 Mbs. However, if you had that 1000 Mbs divided across 4 streams (250 Mbs each) and if one of those streams lost a packet, you would slow down only 125 Mbs. Note that if you are in a heavily congested network, and you start dropping packets on many or all of your streams, you may actually get lower performance because of the overhead of managing multiple streams. For more details, check one of the references.

Send Stalls

Dropped packets are not the only way bandwidth can be cut drastically. The network interface card (NIC) has an input queue. If it becomes full, Linux treats this as a congestion event (equivalent to a dropped packet), and you lose half your bandwidth. The conventional explanation is that if that the network is slow, but today's fast CPUs can overflow the default queue length quite easily. To avoid this problem, increase the size of your queue by issuing the following command:

```
ifconfig eth<N> txqueuelen
  <new queue size>
```

What to use for your queue size depends on your system, but try 2500 and work up from there.

Route Caching

Let's say someone down the hall was transferring a big file to a remote site with which you also work. While doing so, a packet was dropped. Shortly thereafter, you decide to move a big file to that same remote domain. Guess what? You don't even get the privilege of trying to drop a packet. Linux remembers that there was congestion on that link and puts you into congestion avoidance right from the start. Never mind that network traffic is dynamic, so what happened two minutes ago is absolutely no prediction about what is happening now. Never mind that you may not be taking the same route to that destination. So what can you do? If you can become root, you can:

```
echo 1 > /proc/sys/net/ipv4/
  route/flush
```

Channel Caching

Channel caching and its impact on performance will be mentioned only briefly here. In a subsequent article on developing with the Globus GridFTP client library, we will discuss this in more detail and show how to implement channel caching. By default, each call to the Globus GridFTP client library is completely self-contained. It establishes a control channel connection, sends the necessary commands, and then closes the session. If you are going to be repeatedly moving files from the same location this process gets expensive, since every connection involves a delegation (public/private key pair generation, a computationally expensive operation). The alternative is to set an attribute so that the channels are left open. If the next command uses the same source, destination, and credentials, then the existing (cached) channel is used. This attribute can make a dramatic difference when transferring many, particularly small, files between the same source and destination. Note that in the Globus Toolkit V3.2, the command line client provided, called `globus-url-copy`, now supports moving multiple files with a single invocation via file globbing (`*.dat`) and directory moves. The `globus-url-copy` client automatically employs channel caching where it makes sense.

File Size

File size can have a huge impact on the performance of GridFTP. Consider two scenarios. Both involve moving 1 TB of data. However, one scenario involves a single 1 TB file, whereas the other scenario has the same 1 TB in 1,000,000 files each 1MB in length. Clearly, the one large file scenario will be faster than the 1,000,000-file scenario. The obvious factor is that there is only a single window open and you send data until the file is done. In the many-file scenario, a number of things hurt performance. First, you break the "pipeline" each time a file ends, since you have to wait for the first command to complete before you can initiate the next file transfer. Second, depending on the TCP implementation and

configuration, after a single round-trip time with no data moving, TCP may close the congestion window, forcing you into slow start again and hurting the bandwidth. If channel caching (see above) is not employed, the performance hit is disastrous because, in our scenario, there would be 1,000,000 delegations performed. If you have control over the file size, remember: For data transport, bigger is better. If you don't have control over it, there is very little you can do other than ensure channel caching is used.

Summary

The key to getting good performance out of GridFTP involves two basic components. First, you have to do basic system engineering. Any good engineering starts with a set of specs or, in other words, a design target. Usually, this will be a sustained bandwidth you want to achieve between two sites. Once you have that, you have to have the right hardware in place to achieve your goals. To achieve decent performance on a gigabit Ethernet link, you need a minimum dual-processor 1 GHz Pentium III or the equivalent (rarely a problem these days), RAIDed disks, and a GigE NIC. Some hardware works better together than others, so test before you buy. Also, sit down and have a heart-to-heart talk with your networking folks. Make sure they understand what you are trying to accomplish, and get them to explain every single piece of hardware between you and the exit router. Look for links that are simply too slow (your GigE connection gets routed over a 100 Mbs segment at some point) or that are shared (your department has a 16 port GigE hub: that is, 16 GigE connections go in, but only one comes out).

Once you have the right system in place, you have to make sure it is configured correctly so you don't prevent it from doing its job. You will need to balance the constraints of the various applications, but if this is a dedicated GridFTP host, tuned for maximum throughput, latency is not an issue. Tweak the file system and TCP system settings, and then make sure your client takes advantage of it, particularly the TCP buffer settings. If you want high performance, never use the default buffer sizes: set them yourself. The TCP auto buffer tuning will get you much better performance than simply using the default, but hand-tuned buffers will generally outperform auto-tuned buffers.

If you do all those things, you should be able to get decent performance. What's "decent performance"? Remember in school how nice it was to have the answers to your homework so you could check to see whether you had done it right? System checking is a lot harder, but if you want ballpark numbers to compare against, try this: On a 100 Mbs link, you should be able to get between 80 and 95 Mbs on a lightly loaded link. Getting gigabit speeds is harder, but as a rule of thumb, 500 Mbs should be pretty easy to get, and 700 or 800 Mbs is achievable. Again, let use stress that as the load on the network (more packet loss) and the distance you cover (latency) increase, either your bandwidth tends to decrease, or the effort required to achieve a given bandwidth increases.

Good luck, and have at it!

Resources

Linux RAID How-To
- www.tldp.org/HOWTO/HOWTO-INDEX/os.html#OSRAID

Linux System Tuning Guide
- people.redhat.com/alikins/system_tuning.html

Linux.com Performance Guide
- howtos.linux.com/guides/solrhe/Securing-Optimizing-Linux-RH-Edition- v1.3/chap6sec68.shtml

LBNL TCP Tuning Guide
- www-didc.lbl.gov/TCP-tuning/buffers.html

Globus Web Site
- www.globus.org

GridFTP Protocol Specification
- www.ggf.org/documents/GWD-R/GFD-R.020.pdf

Bonnie++ Disk Performance Benchmark
- www.coker.com.au/bonnie++

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38 and under Contract DE-AC03-76SF0098 with the University of California; by the National Science Foundation; by the NASA Information Power Grid program; and by IBM.

Bill Allcock is the Technology Coordinator for GridFTP at Argonne National Laboratory and is a member of the Globus Alliance. His work centers on data-intensive applications, particularly transport and management of large datasets.

John Bresnahan is a senior scientific programmer with the Mathematics and Computer Science Division at Argonne National Laboratory, where he works with the Globus Alliance as a member of the GridFTP team, designing and implementing data transfer protocols for the Grid.