SQUID, GridFTP and SRM tests on ANI Testbed

By Anupam Rajendran, Dave Dykstra, Parag Mhashilkar, Gabriele Garzoglio
Last Updated 05 Sep 2012

1. Introduction

This report gives details about the tests performed on Advanced Networking Initiative testbed[1] with the purpose of middleware evaluation and its tuning to get the best performance. This helps to test the scalability of the protocols by understanding their implementation and overheads. All tests involved transferring data on a Cross country 100 Gbps Testbed[2] placed between DOE Supercomputer centers in Argonne National Lab (ANL) (near Chicago, IL) and NERSC (in Oakland, CA).
The rest of the report is as follows. Section 2 describes the test hardware and network. Section 3, 4 and 5 discuss the testing and analysis of Squid, GridFTP and Storage Resource Management. We conclude in Section 6.

2. Test Hardware and Network

The ANI 100 Gbps testbed has two sets of performance machines. The machines at NERSC are disk performance servers and machines at ANL are memory performance servers. We had access to three machines at NERSC (nersc-diskpt-1, nersc-diskpt-2 and nersc-diskpt-3) and three machines at ANL (anl-mempt-1, anl-mempt-2 and anl-mempt-3). Each machine had four 10 Gbps interfaces through which it connected to 100 Gbps cross country link.
All the ANL machines had two AMD 6140 Opteron Processor with 16 cores(8 cores each) at 2.6 GHz and 64GB memory.
Nersc machines 1 and 2 had Intel Xeon X5650 with 12 cores at 2.67 GHz and 48 GB of memory while Nersc 3 had Intel Xeon E5530 with 8 cores at 2.4 GHz and 24 GB of memory.
The round trip time between NERSC and ANL machines is measured to be 53 ms.
This testbed is not directly accessible to the internet and is available through Virtual Private Network using a Virtual Machine at Fermilab.

3. Squid

a. Tests and Data Set

The testing of SQUID is done by fetching an 8MB file stored in a CERN database using the wget command and using the SQUID server as proxy. The first time when the request goes to a SQUID proxy, the proxy fetches the file from the CERN database and forwards the file to its client while simultaneously caching it. All future requests for the same file are completed using the local copy. So the tests involved repeatedly fetching the same file to get maximum utilization of SQUID servers. SQUID stores the 8MB file on disk, but it is always in the file system buffers so disk access is not necessary.
The parameter space in the test involved:

1. Transferring data in both directions
   a. Server at ANL, Client at NERSC
   b. Server at NERSC, Client at ANL

2. Client to Server connection
    a. One to One transfer - nersc-diskpt-1 client requesting 8MB file from anl-mempt-1 server, nersc-diskpt-2 from anl-mempt-2 and nersc-diskpt-3 from anl-mempt-3
    b. All to All transfer - each diskpt client requesting 8MB file to all mempt servers
3. Number of SQUID processes running in each host. We used squid2 which is single threaded, but ran multiple squid server processes listening on the same port for connections coming in to all 4 NICs.
4. With and without Core Affinity for SQUID instances

In all-to-all transfers, for each NIC on the server side, there are 250 parallel processes running on the client side. There are 12 NICs in total on the server side. So there are 250 x 12 = 3000 clients per machine. And there are three machines on the client side, so in total there are 9000 clients. Each client repeatedly does wgets to fetch the 8MB file using the SQUID server as http proxy.

In one-to-one transfers, each machine on the client end connects to only one machine on the server end. The number of NICs contacted is reduced from 12 to 4. So we increase the number of parallel processes 250 to 750 per NIC to maintain the same total number of clients.

The tests were also repeated for half the number of clients for some cases.

b. Results

Throughput in Gbps

ANL to NERSC:

No. of SQUID servers per ANL host	Core Affinity Disabled			Core Affinity Enabled		
	One to One	All to All		One to One	All to All	
	3000 clients per machine	3000 clients per machine	1500 clients per machine	3000 clients per machine	3000 clients per machine	1500 clients per machine
10	52	50	55	55	50	54
12	62	60	68	62	58	64
14	66	88	85	78	88	84
16	77	**96**	**90**	83	**100**	**90**

NERSC to ANL:

No. of SQUID servers on each of 3 hosts at NERSC	Core Affinity Disabled				Core Affinity Enabled			
	One to One		All to All		One to One		All to All	
	3000 clients per machine	1500 clients per machine	3000 clients per machine	1500 clients per machine	3000 clients per machine	1500 clients per machine	3000 clients per machine	1500 clients per machine
8, 8, 8	70	83	70	80	75	83	76	83
10, 10, 8	76	**90**	75	**90**	**99**	**95**	**97**	**96**
12, 12, 8	80	**91**	79	**91**	**98**	**96**	**98**	**98**

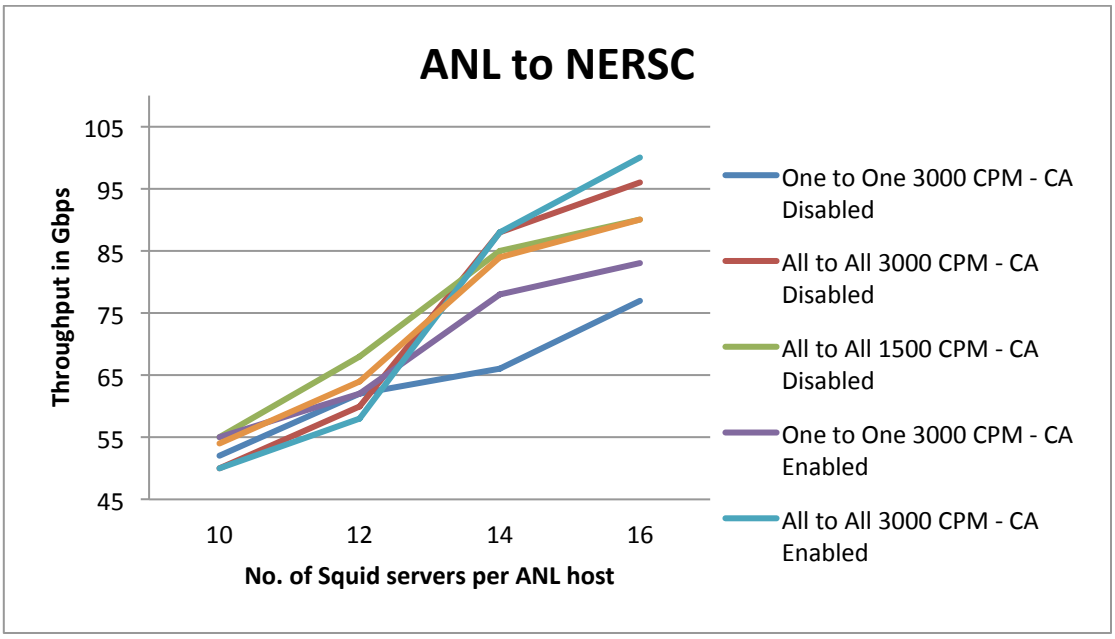

Figure 1 ANL to NERSC

CPM – Clients per machine, CA – Core Affinity

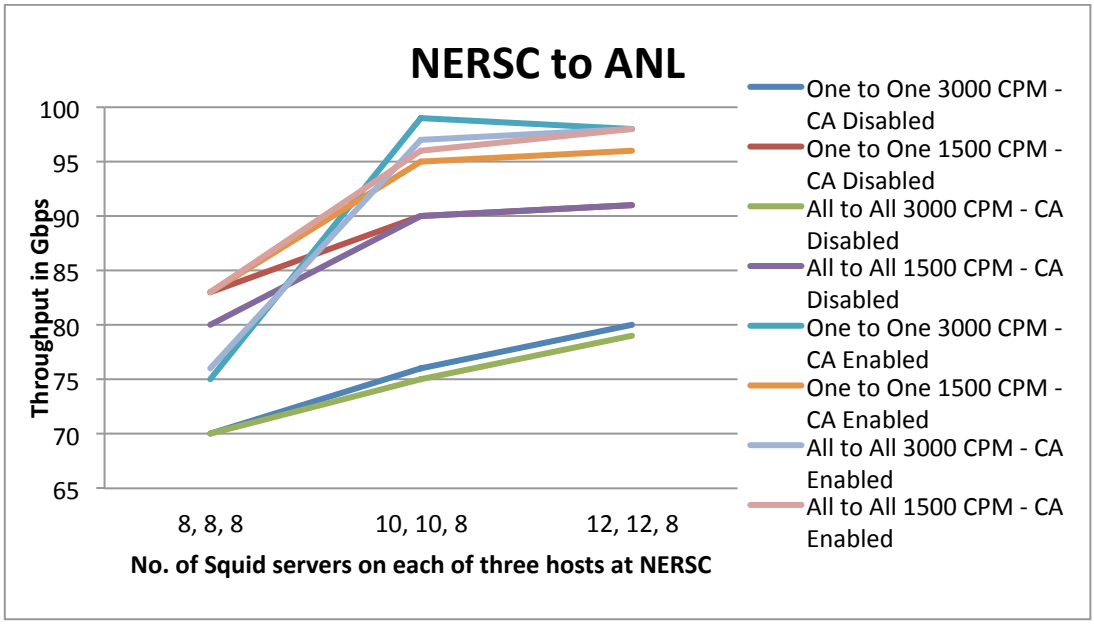

Figure 2 NERSC to ANL

CPM – Clients per machine, CA – Core Affinity

c. Observations

ANL to NERSC:

1. When there are more servers running (14 and 16), it means more capacity. So in All-to-All case more clients are required to increase utilization and hence increase throughput.
2. When there is lesser capacity at the server end (10 and 12), more clients overload the server thus bringing down the throughput. So fewer clients must be there to get maximum utilization of that capacity.

3. In general one-to-one gave lower performance than all-to-all. One possible reason could be that nersc-diskpt-3 is a slower machine and had one 10 Gbps NIC(eth2) that frequently went bad. This causes decreased utilization when there is more client load. For the same reason one-to-one tests were not repeated for 1500 clients per machine as it was not going to improve any further.
4. Finally, core affinity did improve performance slightly, especially when there were higher numbers of SQUID servers per host.

NERSC to ANL:

1. As seen before, more capacity on the server end required more clients to get better performance.
2. One-to-one performance was almost same as all-to-all.
3. When the servers were running on NERSC side, even the one bad NIC did not cause any issue because the other three NICs were compensating and sending more data than the bad interface.
4. 1500 clients had improved performance only when there was less server side capacity, because too many clients overload the servers.

It is seen that in both directions, as we increase the number of SQUID servers per host, the throughput increases. When the number of servers was low, it became the bottleneck and when the number of servers was high, number of client became the bottleneck.
Also enabling core affinity pushed the throughput higher.

4. GridFTP

GridFTP is a high performance, secure, reliable data transfer protocol optimized for high bandwidth, wide-area networks.

a. Data Set:

There were three categories based on the size of file being transferred.

1. Small - 8KB to 4MB
2. Medium - 8MB to 1GB
3. Large - 2GB to 8GB

File size increased in powers of 2.

b. Tests:

As GridFTP tests involved extensive disk reading, NERSC side was chosen to be the source server since they had high performance disks.
The GridFTP client used was globus-url-copy. Performance options like parallelism, concurrency and pipelining were tried on all categories although analysis shows that pipelining feature did not work as expected. Parallelism was limited only to large files while concurrency was used for both large and medium files.
The local server-server tests were initiated at NERSC using the globus-url-copy command and files were transferred from every 10 Gbps interface on the NERSC hosts to every 10 Gbps interface on the ANL hosts where it was written to /dev/null to avoid any disk latency.
As the file size decreased, the number of files being transmitted was increased to have constant amount of data being transferred.

c. Results:

These values have been referred from the report [3] of the GridFTP tests.

Type	Local: Server-Server	Globus Online
Large Files	92.74	62.9
Medium Files	90.94	28.49
Small Files	2.51	2.3

d. Analysis of Results:

GridFTP performance was affected by Lots Of Small Files(LOSF) problem. Although the idea of pipelining mentioned in the paper **GridFTP Pipelining**[4] aimed to solve this problem, it did not work as explained.

A simple test was done by transferring five small files of sizes 8KB and 2MB. The TCPDump of this transfer was analyzed.

For a local server-server transfer, there were two control and one data channel formed as shown below.

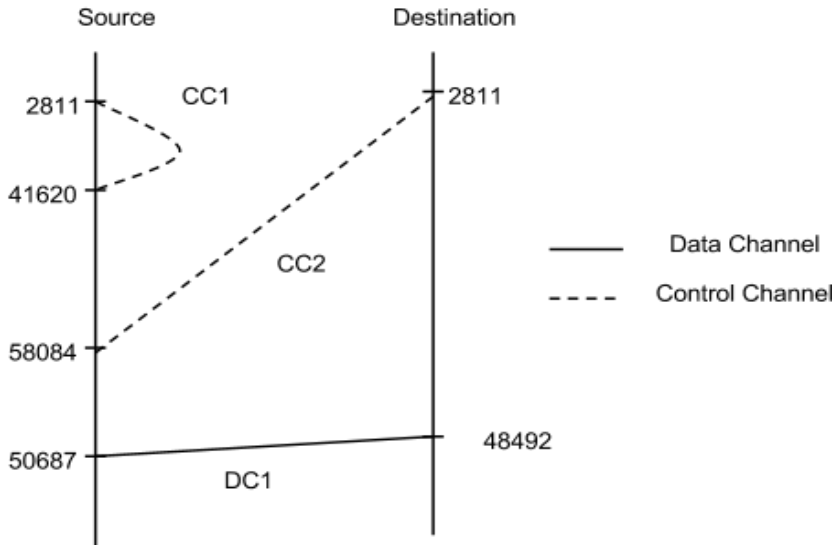

Figure 3 Control and Data Channels in Local Server-Server Transfer

The series of commands that flow on control channel can be summarized as:

1. Before the transfer of first file, GSSAPI authentication takes place on both control channels.
2. Client sends STOR command on control channel 2 to destination.
3. Client sends a RETR command to source server on control channel 1 .Source server sends back Beginning transfer.
4. Destination server sends Beginning transfer to the client and then sends the first performance marker.
5. Then data starts flowing on the data channel.
6. If the file is big enough, the destination keeps sending performance and range markers every 5 seconds on the control channel 2.
7. During the transfer, when the source server has done sending data, it sends Transfer Complete to client on control channel 1.
8. After receiving all data on data channel, the destination sends the final performance marker, range marker and Transfer Complete status on the control channel 2.
9. If there are more files to send, then the process repeats from step 2.

This information is also logged by both servers.

Note: 1. Perf Marker is an instantaneous state of transfer at a given timestamp. It tells how many bytes have been transferred on a stripe till that time. This is there in extended block mode (MODE E) of GridFTP to monitor the performance.

2. Range Marker is present to provide back compatibility with BLOCK mode. It is a concatenation of all byte received. These are essentially restart markers where a client can request to restart transfer a particular range of data using REST command.

The above analysis was done with the options: **-pp -p 1 -cc 1 -fast -nodcau**

If the pipelining option is removed, then the client sends SIZE command to source server and ALLO command to the destination server before sending the STOR command on the control channel. The source server replies with the size of the file. The destination server after receiving ALLO command sends back an "ALLO command successful" to the client. Then the same steps are followed from step 2.

The complete flow diagram of above steps can be visualized as:

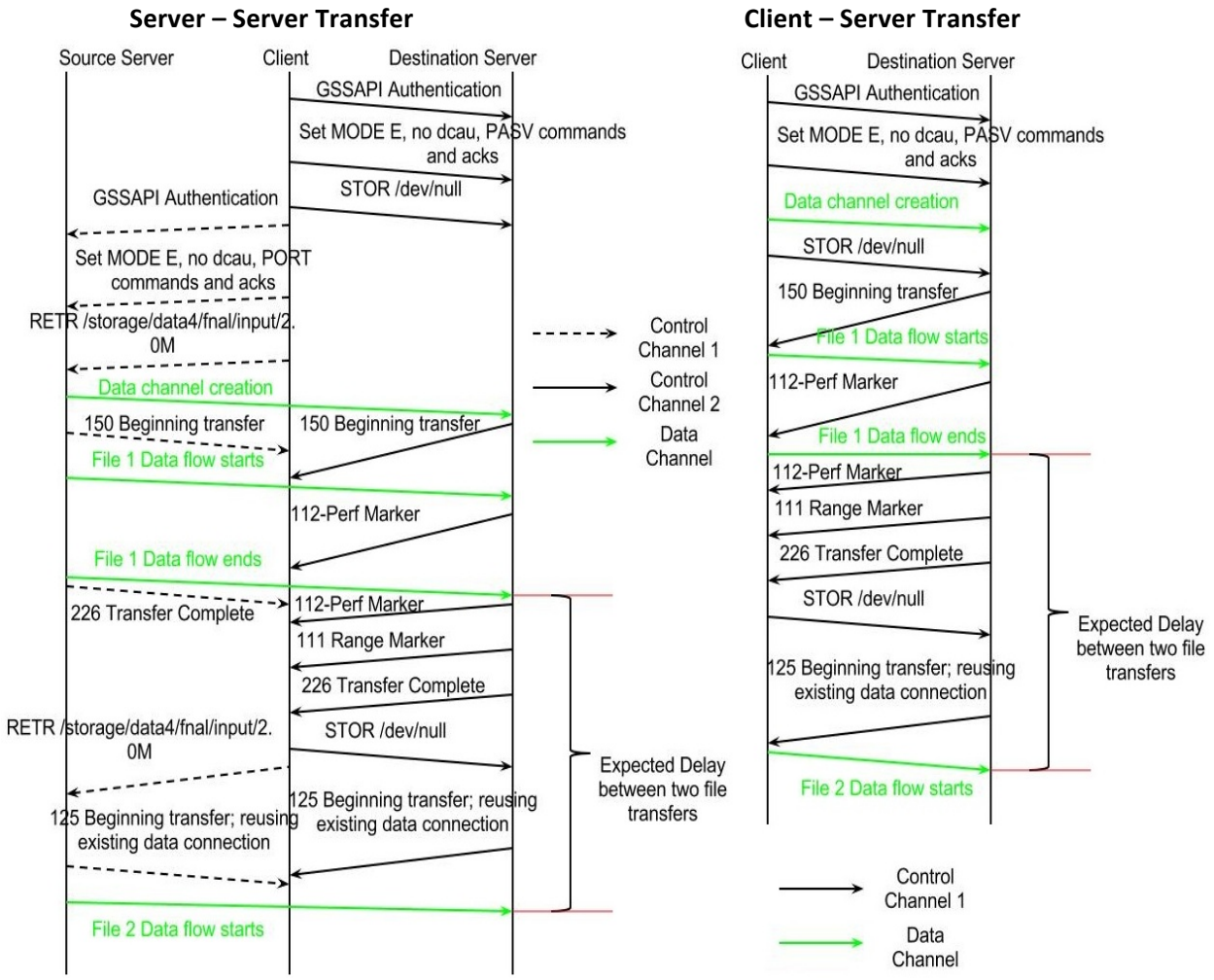

Figure 4 Command Flow Sequence in GridFTP Server-Server and Client-Server Transfer

e. Observations

The throughput is limited by the STORE command that needs to be sent for every file only after receiving the Transfer Complete response for previous file sent on the same data channel. This causes a wait time of one RTT between every file transfers. For small files, this wait time is larger than the actual time required for data transfer. And when there are too many small files, this overhead affects the throughput badly. Thus we see a low throughput of 2.5 Gbps for small files.

One good thing is that it has data channel caching enabled which means the same data channel can be reused if the set of files are given in a transfer list. This avoids the overhead of establishing the data channel for transferring every file.

5. SRM

SRM is a web service protocol operating over http and is the most common protocol for interfacing storage. Its main purpose include

1. Metadata operations
2. Data movement between storage elements
3. Generic management of backend storage

SRM is not designed for high throughput transfers, so it simply redirects the client to transfer protocol like GridFTP. This can also effectively load balance transfers over multiple nodes and thus showing good scalability.

a. Data Set:

The Data set is same as that of GridFTP comprising small (8KB to 4MB), medium (8MB to 1GB) and large files (2GB to 8GB) increasing in powers of 2.

b. Tests:

The local server-server tests were run in a similar manner to GridFTP i.e. files were transferred from every 10 Gbps interface on the NERSC hosts to every 10 Gbps interface on the ANL hosts. The tests used the BeStMan server which is a full implementation of SRM v2.2.

There were three client tools available for SRM:

1. BeStMan client (srm-copy)
2. dcache client (srmcp)
3. LCG Utilities (lcg-cp)

The first two clients enabled giving an input file containing list of files to be transferred as a command line option. The BeStMan client required the file to be in XML format while the dcache client just needed a text file as in globus-url-copy. BeStMan client also had support for parallelism and concurrency. LCG and dcache client had option for multi-stream transfer.

c. Results:

The tests were run three times using LCG Utilities with four parallel streams for large and small files and two parallel streams for medium files. The average values for three trials are presented here.

File Size	No. of parallel streams	Simultaneous lcg-cp	Total Data Transferred (MB)	Average Transfer Time (sec)	Throughput (Gbps)
Large: diskpt-1	4	48	1,376,256	378.33	29.10
Large: diskpt-2	4	48	1,376,256	379.00	29.05
Large: diskpt-3	4	48	1,376,256	377.00	29.20
Large: Total			**4,128,768**	**378.11**	**87.36**
Medium: diskpt-1	2	48	489,600	149.67	26.17
Medium: diskpt-2	2	48	489,600	154.00	25.43
Medium: diskpt-3	2	48	489,600	153.33	25.54
Medium: Total			**4,128,768**	**152.33**	**77.15**
Small: diskpt-1	4	500	5,994	289.00	0.17
Small: diskpt-2	4	500	5,994	290.33	0.17
Small: diskpt-3	4	500	5,994	291.67	0.16
Small: Total			**17,982**	**290.33**	**0.50**

Large files gave a throughput of 87.36 Gbps, medium files gave 77.15 Gbps and small files gave 0.50 Gbps.

d. Analysis of Results

SRM tests performed ran on top of GridFTP for transferring data. So throughput of SRM is bounded by that of GridFTP. Moreover SRM has its own overhead like converting URL from srm:// to gsiftp://. Further, it has no data channel caching, meaning it closes data channel after every file transfer and opens a new data channel for next file transfer. This affects the throughput badly. In case lots of small files, data channel is established for every file transferred.

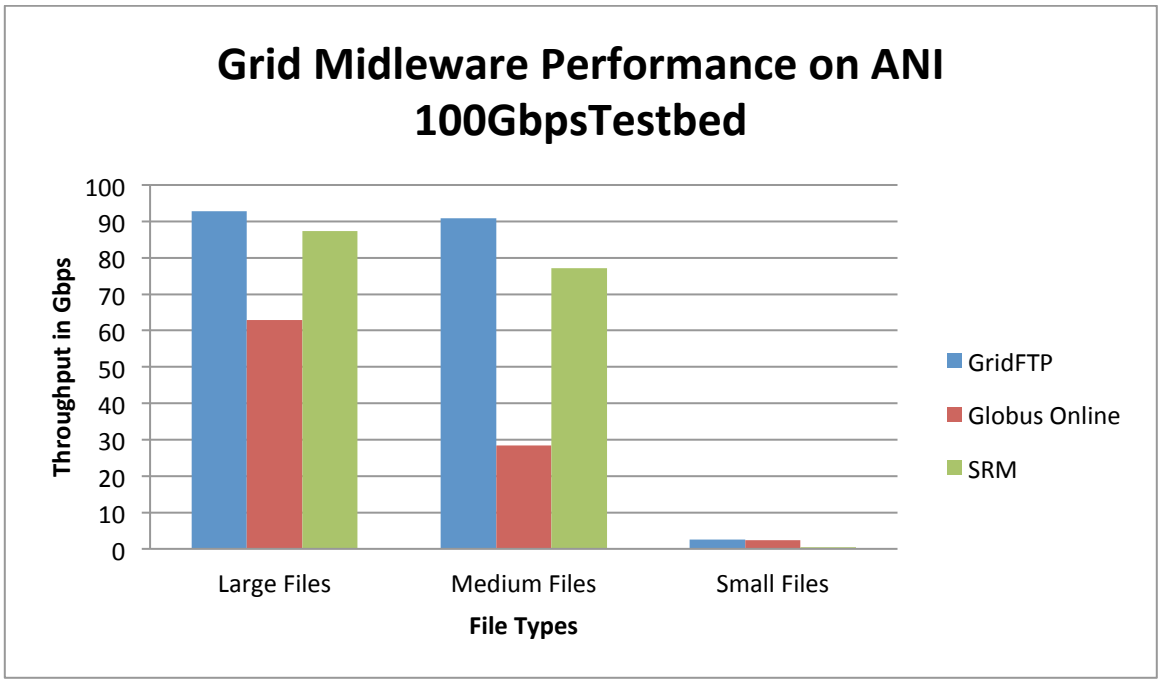

Figure 5 Performance comparison of Local Server-Server transfers using GridFTP, Globus Online and SRM

6. Conclusion

The analysis of results indicates the potential that the different middleware technologies could be scalable up to 100 Gbps in certain cases. It also reveals the cases where the middleware performs poorly and thus the features that need to be improved for better performance. For example, in GridFTP, a proper implementation of pipelining feature is required to get better performance when there are lots of files to be transferred.

7. Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1007115, ExTENCI: Extending Science Through Enhanced National Cyberinfrastructure.

8. References

[1] ANI Testbed Overview - https://sites.google.com/a/lbl.gov/ani-testbed/
[2] Testbed Description - https://sites.google.com/a/lbl.gov/ani-testbed/testbed-description

[3] Identifying Gaps in Grid Middleware on Fast Network with the Advanced Networking Initiative, Dave Dykstra, Gabriele Garzoglio, Hyunwoo Kim, Parag Mhashilkar, Scientific Computing Division, Fermi National Accelerator Laboratory
[4] GridFTP Pipelining - http://www.globus.org/alliance/publications/papers/Pipelining.pdf