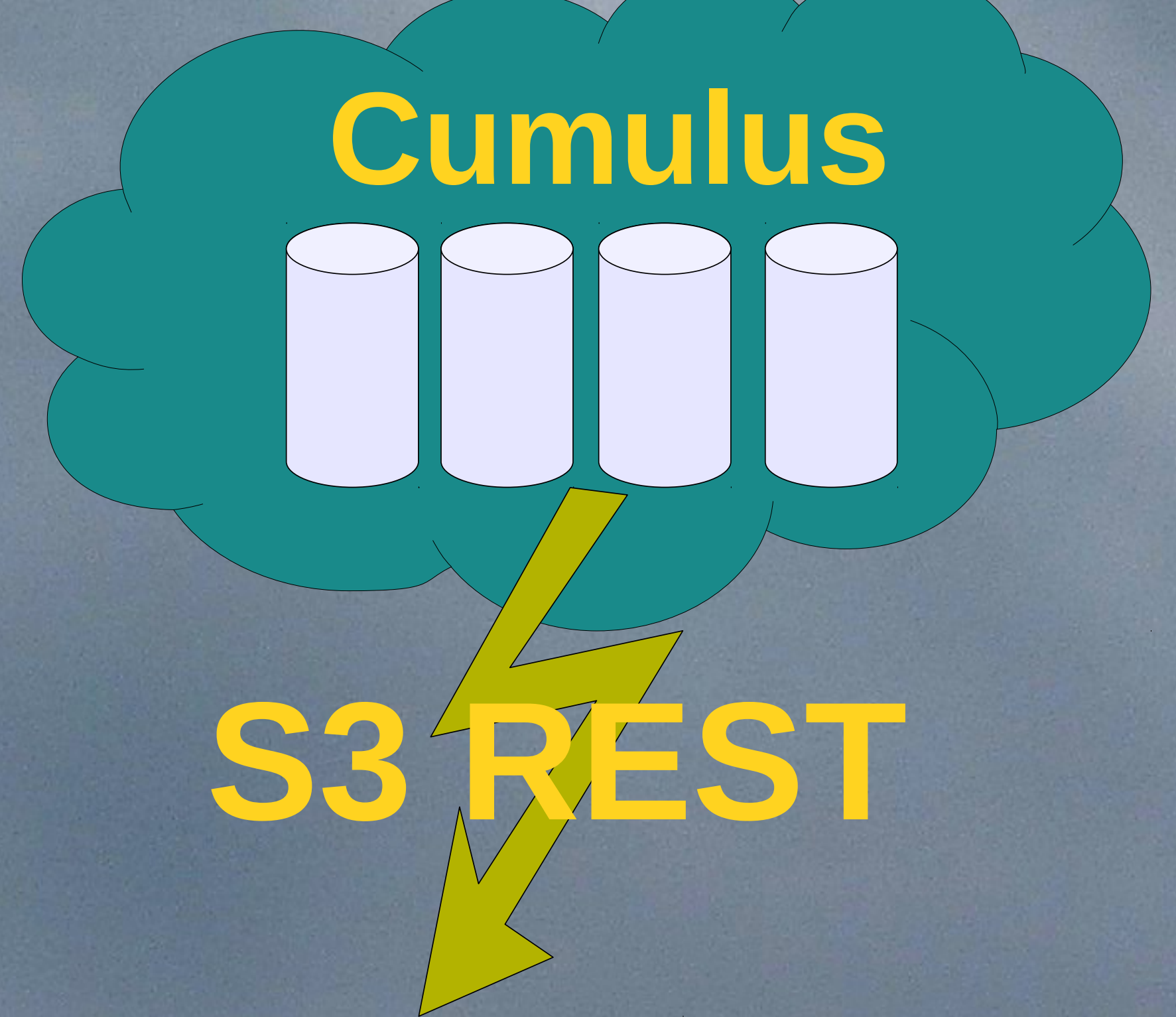
Cumulus

Open Source Storage Cloud for Science

John Bresnahan, Tim Freeman, David LaBissoniere, Kate Keahey



Amazon's S3 protocol has emerged as the de-facto interface for storage in the commercial data cloud. However, it is closed source and unavailable to the numerous data centers actively used for science. Just as Amazon's S3 provides reliable data cloud access to commercial users, scientific data centers must provide their users with a similar level of service. Ideally scientific data centers could allow the use of the same clients and protocols that have proven effective to Amazon's users, but can the S3 interface compare to the data cloud transfer services used in today's computational centers? Does it have the feature set needed to support the scientific community, and if not can it be extended to include them without loss of compatibility? Can it scale and distribute resources equally when presented with common scientific usage patterns?

We address these questions by experimenting with Cumulus: an open source implementation of the Amazon S3 REST API. It is packaged with the Nimbus IaaS toolkit and provides scalable and reliable access to scientific data. We have compared its performance to that of GridFTP and SCP and we have added features necessary to support the cost metrics important to the scientific community.

Features for Science

S3 REST API Compatible

Works with popular clients (s3cmd, boto, JetS3t, etc).

Disk Usages Quotas

In the scientific computing world users do not typically pay for storage with dollars. Yet storage is still a valuable commodity that must be rationed to users based on other metrics. Cumulus allows administrators to easily set per user quotas allowing resources to be provisioned appropriately. This extension is backward compatible with the existing S3 REST API. Quotas are set by administrators with easy to use tools and when a client attempts to exceed a quota the S3 *AccountProblem* is returned.

Easy to use

Cumulus can be installed with a single command, and run with another command. A rich set of user management tools are included as well, thus valuable system administrator time is not wasted dealing with a complex system.

Customizable backend storage system

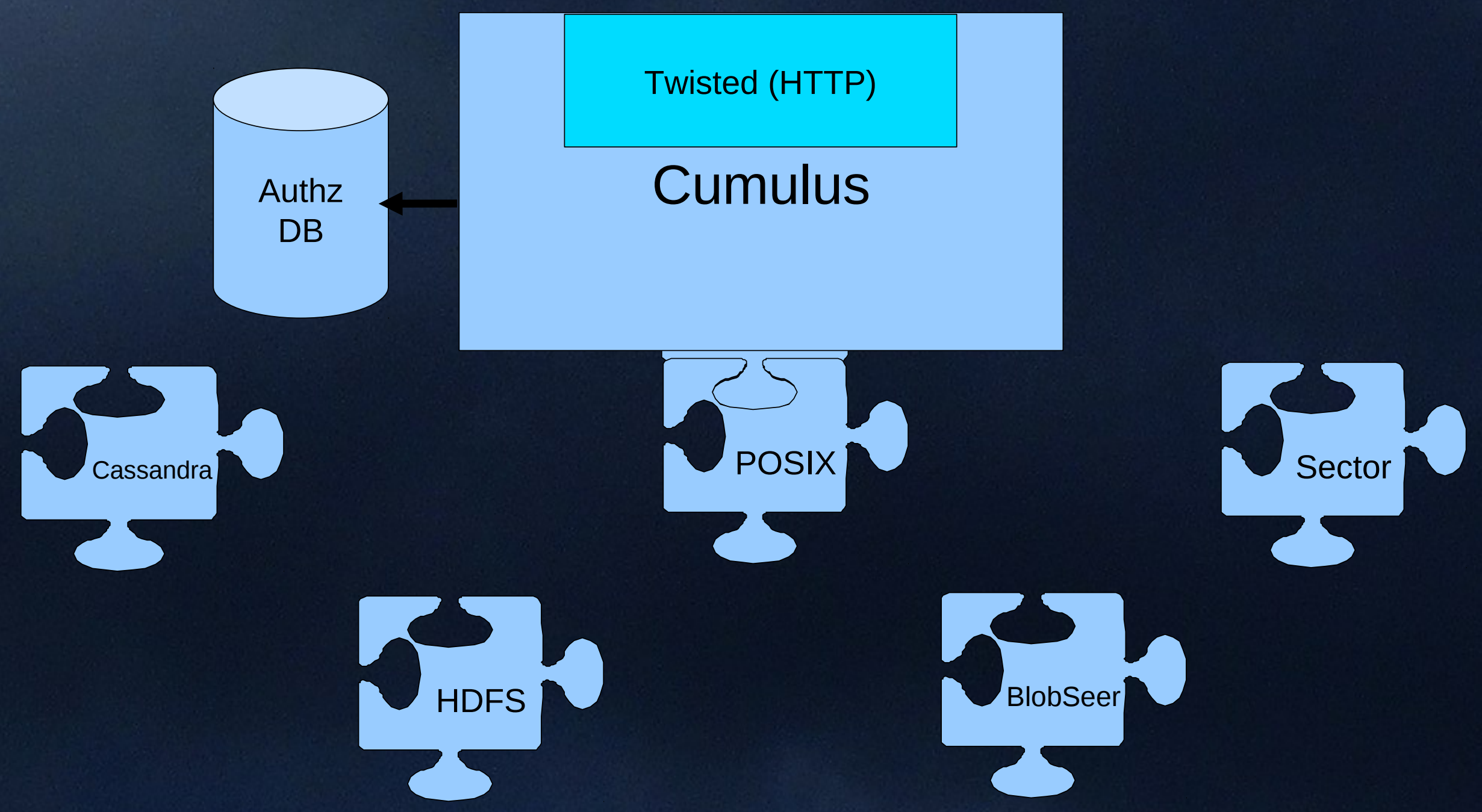
Different data centers have varying degrees of hardware resources. Some have the resources to provide extremely high levels of redundancy and reliability, while others have simpler needs. Cumulus can be customized to support any storage backend and thus a data center can make its own decisions about cost versus reliability.

Architecture

Cumulus provides a plugable abstraction to the storage system. The abstraction is clean and simple, thus it allows for the creation of any number of custom storage drivers suitable to the needs of various data centers.

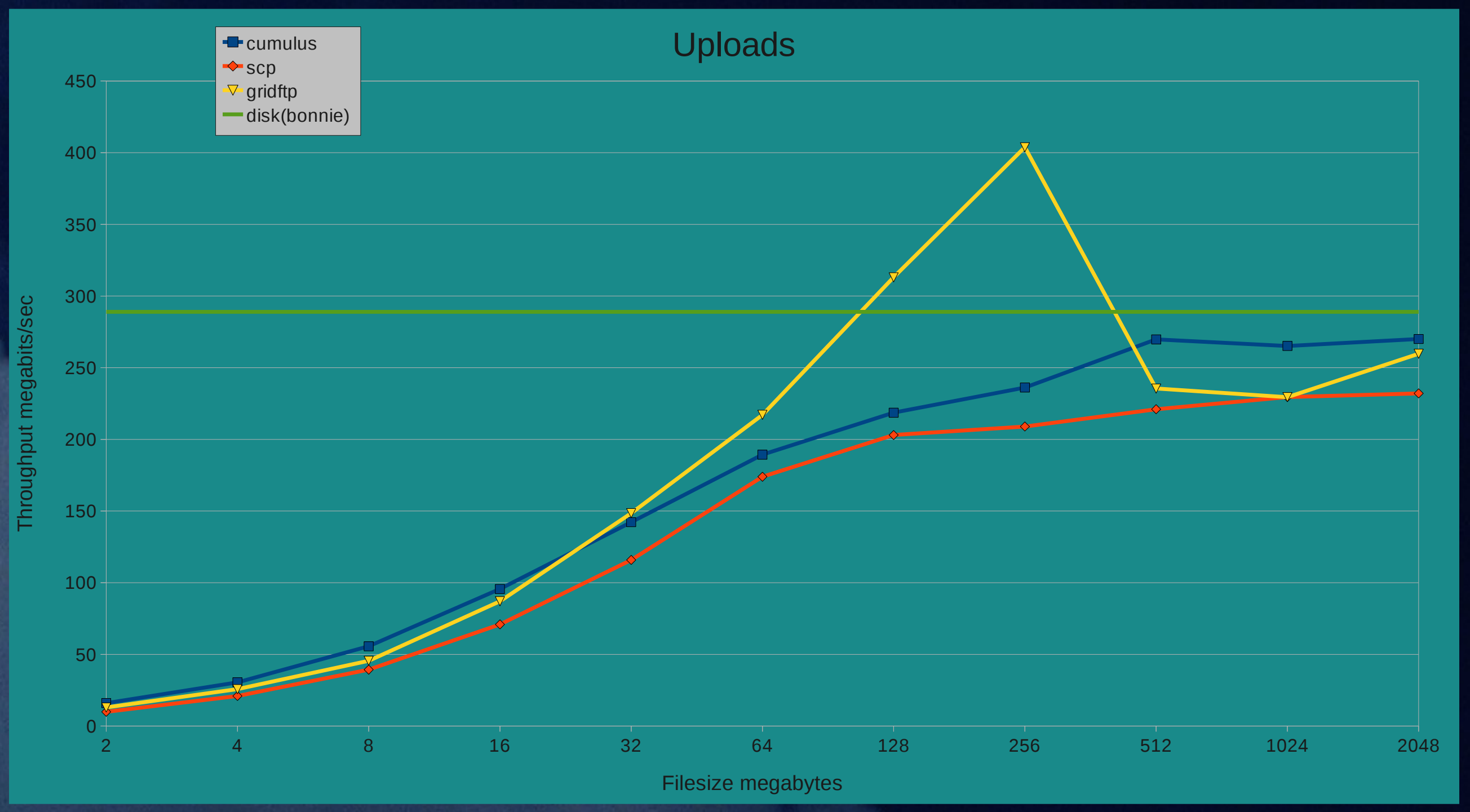
System administrators can choose which storage system meets their needs while at the same time providing the same known network API to clients. If desired, a custom backend can be created against our documented interface to meet the specific needs of special data centers.

The POSIX backend has been included in the first release of Cumulus. This allows for access to many complex storage systems via the filesystem, however future releases will include direct access to HDFS, Sector, and Cassandra.

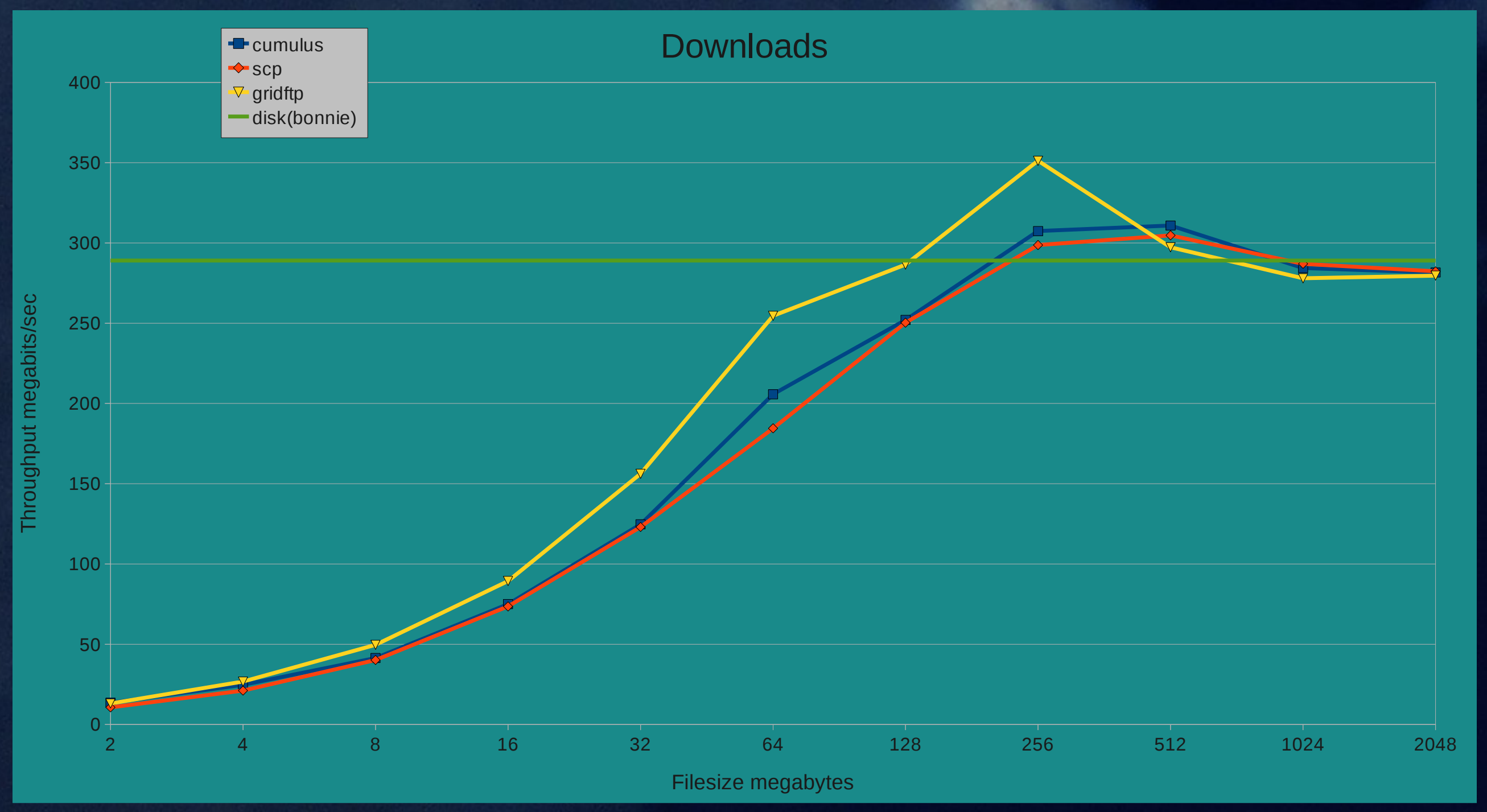


Acknowledgements and Logos

Performance

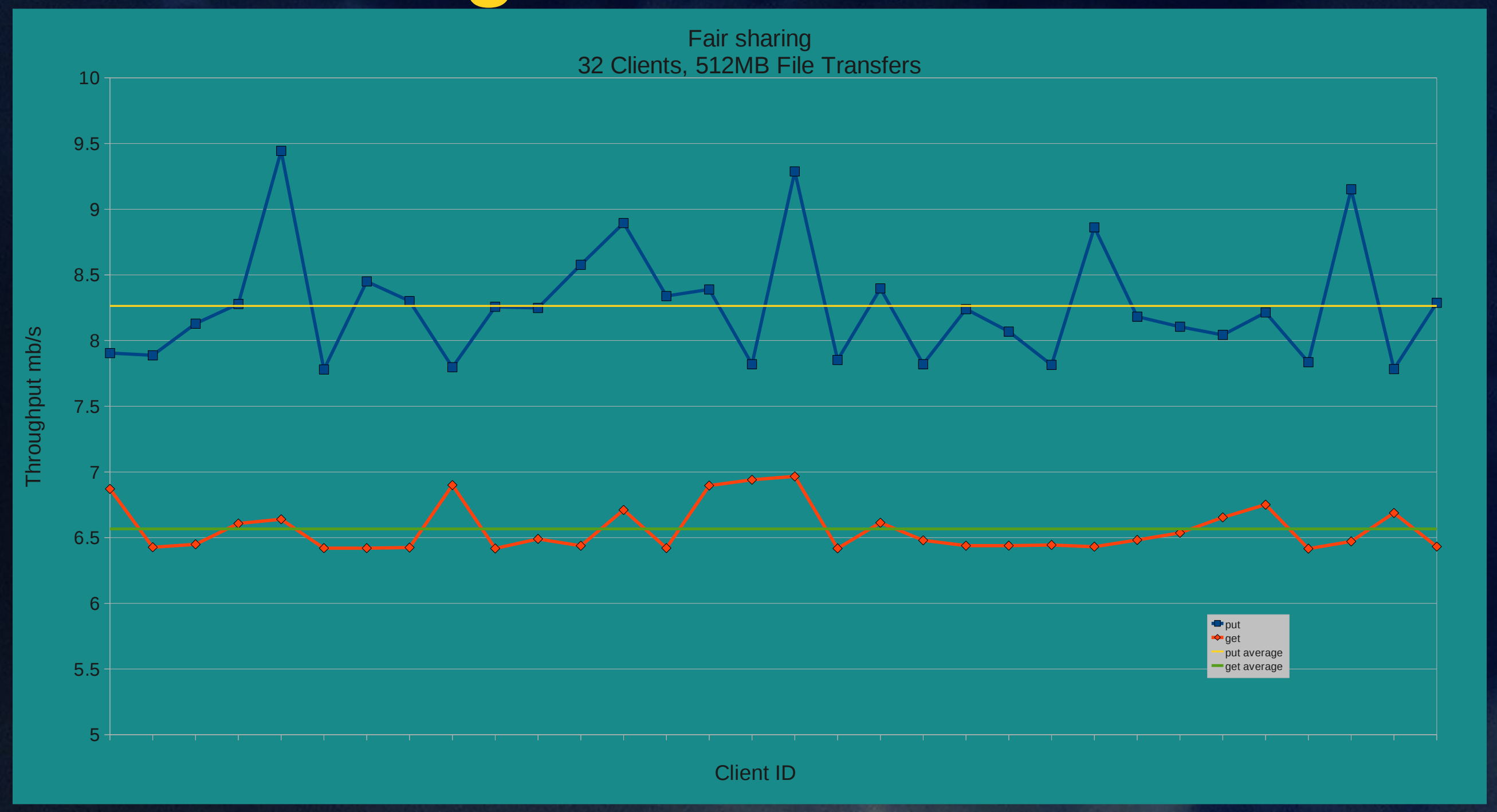


The performance of Cumulus is on par with other popular storage system transfer protocols. GridFTP has set the standard for data transfer performance and Cumulus displays similar (and in some cases better) performance characteristics. The graphs here shows single file upload (top) and download (bottom) throughput of increasingly larger files on a GigE connected LAN. The machines used in all experiments had 512MB of RAM and the disk throughout was measured by bonnie++.



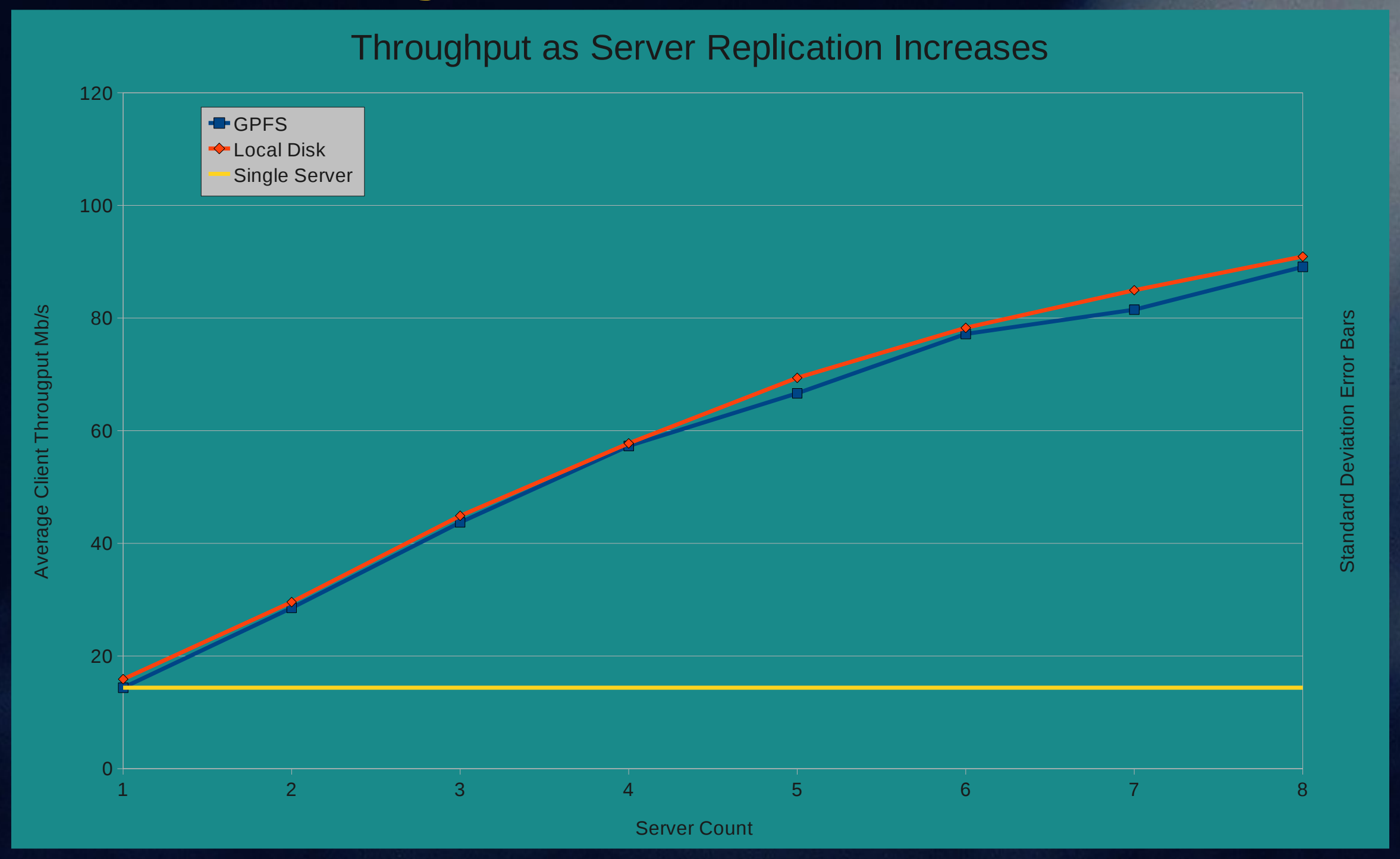
The spike in GridFTP's performance is related to the amount of RAM on the system. When the memory cache for disk is full GridFTP experiences significant performance degradation while Cumulus continues to steadily increase.

Fair Sharing



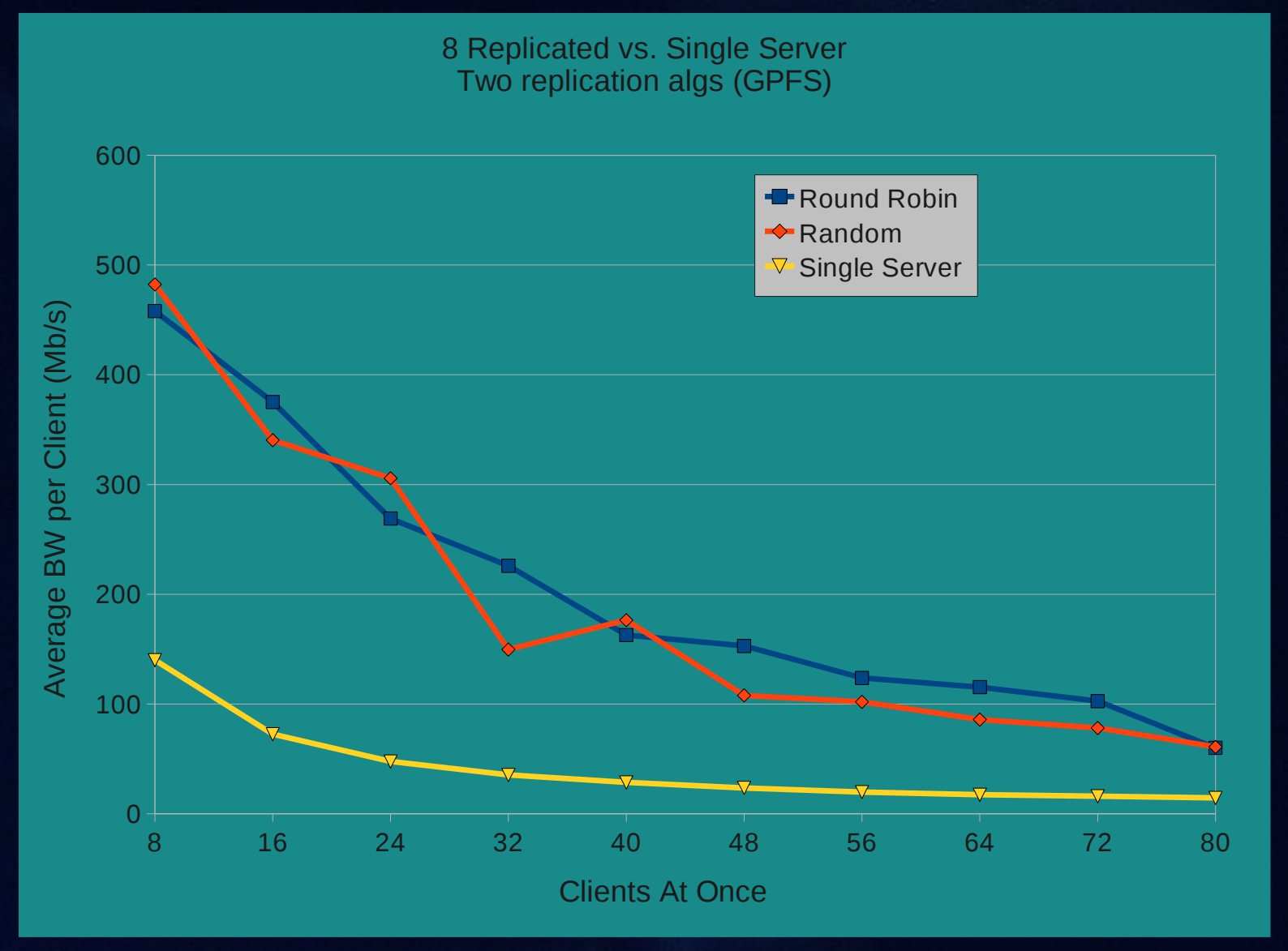
The above graph shows how Cumulus distributes bandwidth to many simultaneous clients. 32 clients were run at the same time all uploading and then downloading a 512MB file from the same Cumulus server. The average achieved BW is shown to indicate how far each client deviated. The results indicate less than two megabit/second difference in the worst case. The collective BW for puts was 264mbs and 210mbs for gets.

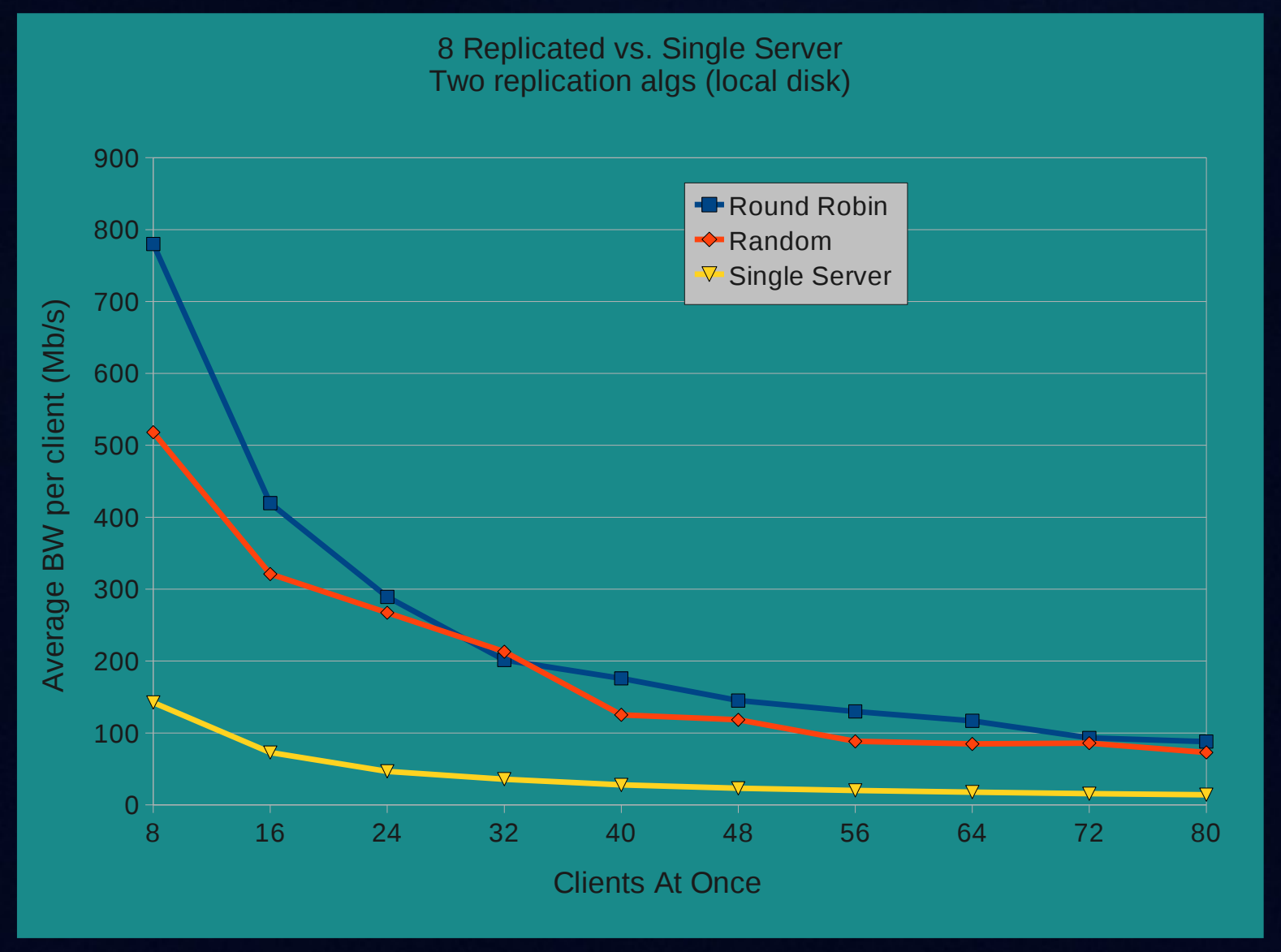
Scalability



By leveraging the S3 protocol's redirect feature, Cumulus can be configured to run as a set of replicated hosts. To display this feature we ran an experiment where 80 clients, run on 8 separate machines (10 on each) downloaded a 500MB file at the same time. We steadily increased the number of replicated servers from 1 to 8 and we used the round robin algorithm to determine the redirection target.

The average of 10 trials were recorded. The graph above shows the results of this experiment. Two storage systems were used, *local disk* which is mirrored on all of the 8 servers, and GPFS. As shown in the graph above average throughput steadily increases as more servers are added.

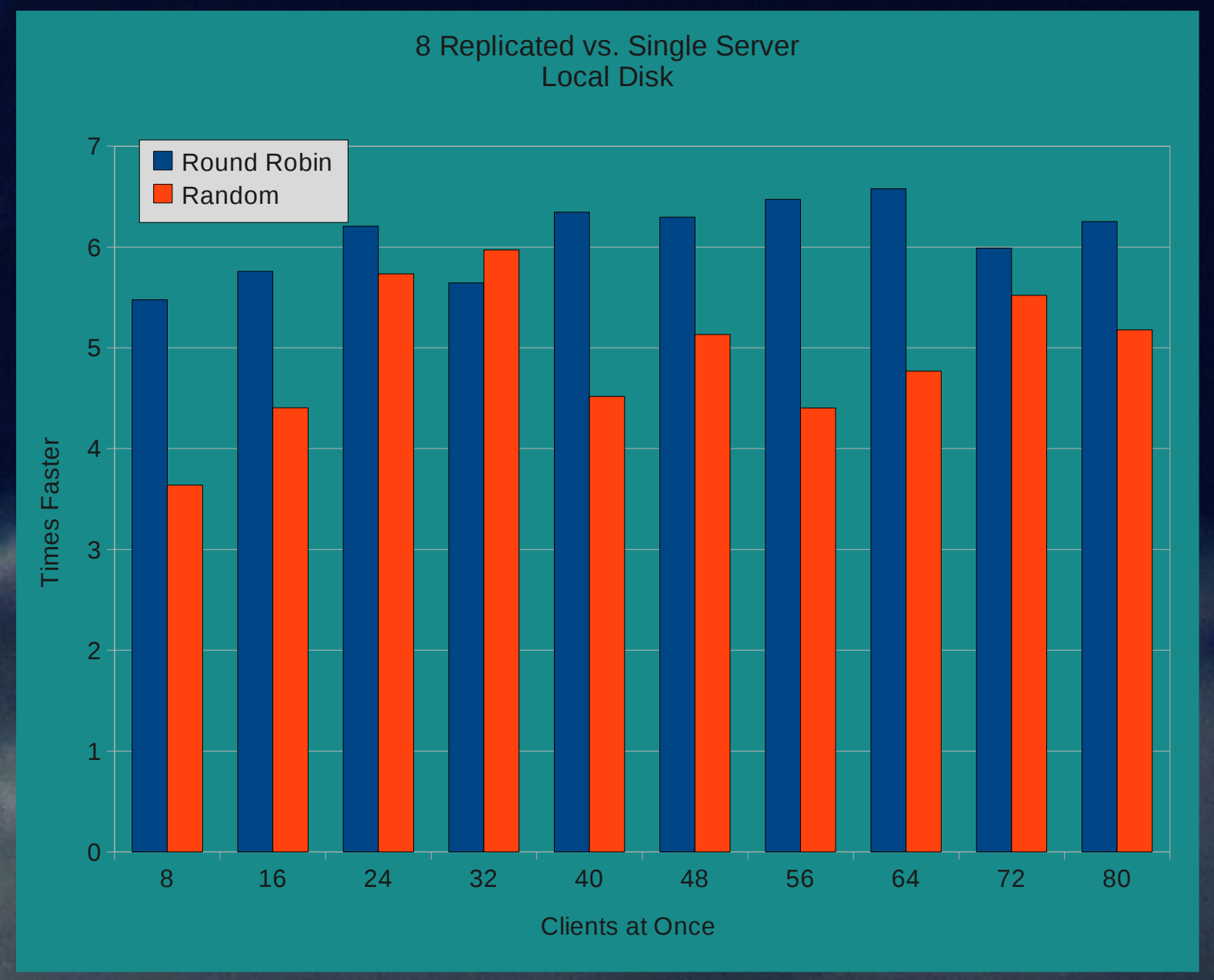


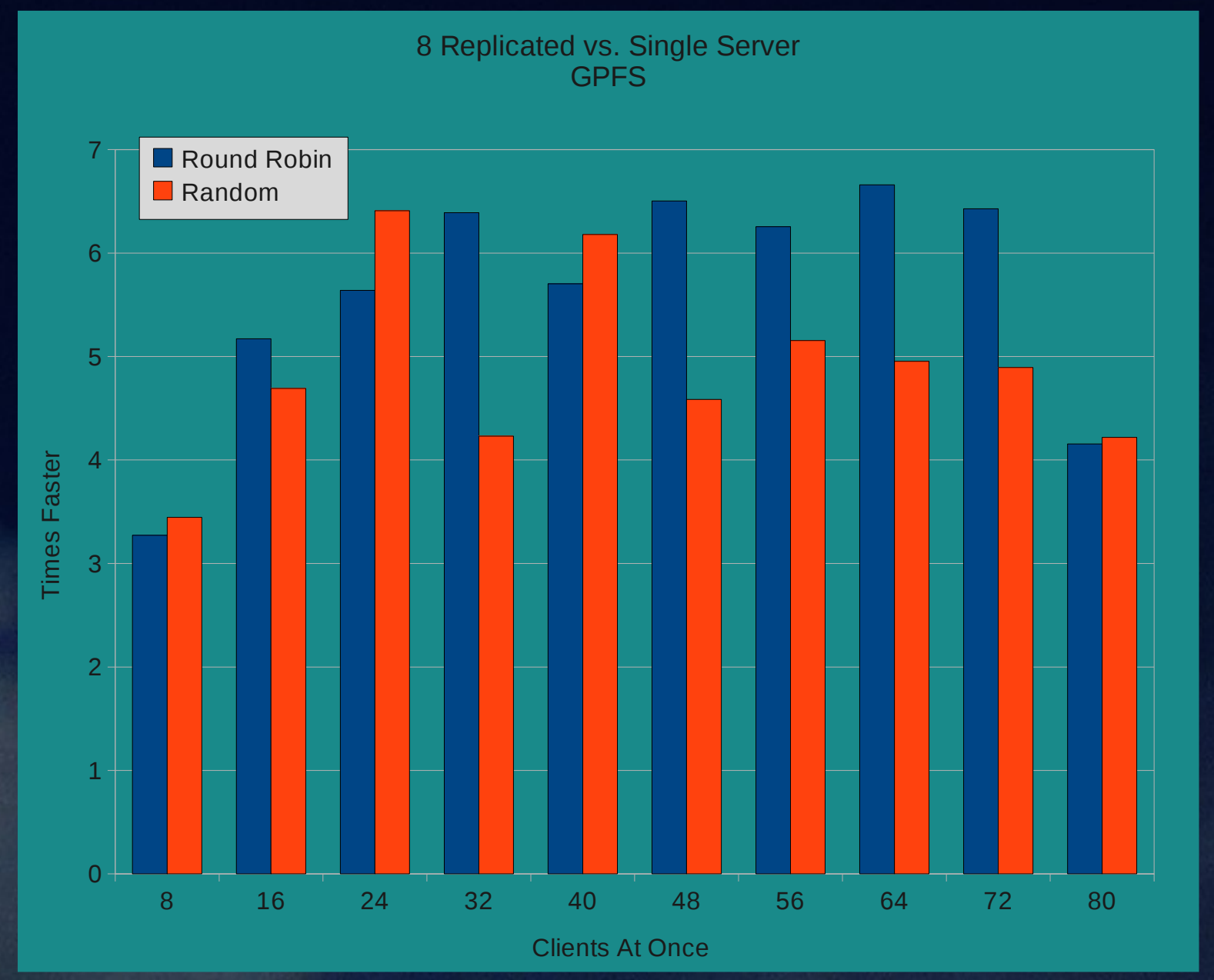


Above we compare the effects of an increasingly heavy client load. Eight client machines are used to simultaneously download a 500 MB file. The number of clients downloading on each client machine ranges from 1 to 10. Three different server configurations are used:

- •Single Server: A single non/replicated Cumulus server.
- •Round Robin: An eight node replicated Cumulus server using a redirection scheme that cycles through the list of nodes.
- •Random: An eight node replicated Cumulus server using a redirection scheme that selects the target from the list at random.

Since all of the files are the same size, and they start at the same time *Round Robin* line shows the best results. Random show results more true to the a real world workload. In this case the ideal server may not be chosen each time. In the above results we see that Random and *Round Robin* are both significantly better than *Single*. And that Random only pays significant penalty when using local disk. When using the more likely setup of a network file system (GPFS) the benefits of the ideal choice are already normalized by network traffic contention.





The above bar graphs present the same data as show in the line graphs but here we focus on how much faster the replicated server is than the non-replicated server. In all cases we see a 3x increase in performance with an average of ~5x. Ideally we would see a performance increase of 8x across the board, however the latency of processing the redirection and the network competition (especially when using the network file system GPFS) introduce overhead.