Storage Matters: Evaluating the Impact of Big Data Transfer Techniques on Storage Performance

(Submitted to the 2013 International Conference on Internet Computing and Big Data, ICOMP'13)

Adam H. Villa
Providence College
Providence, Rhode Island 02918
Email: avilla@providence.edu

***Abstract*—When it comes to transferring Big Data, there are two main areas of concern: network performance and storage performance. The primary focus of recent work has been devoted to the problems of network connectivity and bandwidth. Different transfer techniques have been proposed to quickly move massive amounts of data between computers. The goal of these techniques is to maximize bandwidth consumption by any means necessary. The network performance of these techniques has been analyzed; however their impact on storage performance is not thoroughly investigated. In this study, Big Data transfers are evaluated from the storage viewpoint. Particular attention is focused on the granularity of request sizes issued to a storage node. This paper illustrates that there is a significant impact on performance when small portions of a data set are requested in place of a single large request.**

I. Introduction

The amount of data being produced every day is growing at a tremendous rate. Both academic and corporate entities are creating massive data sets and sharing these files with users around the world. The pinnacle of scientific data creation is CERN's large hadron collider experiments, which have generated thousands of terabytes of data [5]. CERN uses a well-defined data grid to distribute the data around the world [1]. Users are able to connect to one or more servers to retrieve the desired data sets. The user is tasked with creating, monitoring, managing and maintaining the data transfer [12], [13]. Other scientific fields, such as DNA sequencing, have also created massive data sets and share them using clouds or other distributed systems [2].

One of the major concerns for users is the ability to download these data sets for personal use. When it comes to file transfers there are two key concepts that should be of concern to users: networking and storage. Due to limited bandwidth and utilization of public networks, the main focus is typically network based. Storage is often not examined since it is assumed that it will be able to provide adequate performance. Since the focus of performance improvement studies have been isolated to networking, many of the techniques developed for Big Data transfers are engineered to maximize throughput of network connections. Studies examine how these techniques impact network performance on a large scale [11], however the impact of these techniques on storage systems is not fully understood. This paper attempts to examine their impact on simple storage systems to by simulating shared storage systems with a varying number of concurrent users.

When users retrieve these massive data sets, they are typically moving them across shared networks and the Internet. Due to the intense demand for available bandwidth, users on shared connections must compete for bandwidth in order to retrieve the desired data sets as quickly as possible. On residential and campus networks, users will find themselves competing with predominately streaming multimedia. A study of network traffic on a campus network illustrated the demands placed on shared Internet connections for thousands of users and identified that Netflix and other streaming video services consumed large portions of available bandwidth throughout a typical day [15]. Users must compete with this demand during peak periods in order to successfully transfer their desired data sets [14].

It is because of this intense competition for shared network resources that researchers have developed various transfer techniques to greedily consume as much bandwidth as possible for Big Data transfers. Recent studies examine various techniques for maximizing the throughput of a user's internet connection in order to reduce transfer times. There are many different techniques, but the main idea behind all of these techniques is to open multiple, simultaneous connections to one or more data servers. The user would then download portions of the data sets using multiple data streams that are optimized to maximize bandwidth usage. Section 2 presents a summary of the key works in this area.

Network performance is an important and crucial component of data transfer performance, however storage performance is equally important. Storage systems are closely monitored and engineered for high availability and low latency by system administrators. Users, however, do not immediately consider how their requests are impacted by the storage system. As most storage systems still use magnetic, hard disk drives for primary storage, the physical access times for disk requests are critical components of a user response times. Traditional hard disk drives are orders of magnitude slower than other devices and therefore they must be properly utilized in order to reduce physical access times. There have been many studies that examine the performance of disks and storage systems [8]. This paper does not examine or evaluate a specific storage device or system. The focus of this paper is to evaluate the impact that data transfer techniques have on storage system performance and consequentially on the performance of user requests. The rest of the paper is organized as follows. The next section details some of the techniques that have been designed for Big Data transfers. Evaluations of storage performance under varying user workloads is presented in Section 3. Section 4 summarizes the findings of this study and provides motivation for future work.

II. Related Work

This section presents several Big Data transfer techniques developed specifically for retrieving extremely large files over shared network connections and the Internet (highlighted in boldface). Many large data sets are often replicated amongst multiple servers distributed around the world. Users can then retrieve data from any of these replica servers. The transfer techniques are grouped based on how they retrieve data from the servers that store the data. These studies evaluate their various transfer techniques solely from a network perspective. Their impact on storage performance is evaluated using a simulated environment in Section 3.

A. Basic Technique

The basic, **brute-force**, parallel download technique [9] issues a request for equal sized portions of the file from all available servers. Every replica that contains the file is utilized and each is responsible for servicing an equal amount of data. There is no consideration given to the performance of servers or network conditions. Many studies include this technique as a baseline for comparison with other co-allocation strategies.

B. Predictive Techniques - History Based

In the brute-force technique, the performance of each transfer is not analyzed. Depending on network and server workload, each transfer will have varying performance. The following algorithms take into account the performance metrics of each server interaction when dividing the workload amongst all replicas in order to minimize the transfer completion time. Vazhkudai presents a **history-based data co-allocation technique** [9], [10]. This technique adjusts the amount of data retrieved from each replica by predicting the expected transfer rate for each replica. Zhou et al. also develop a history-based data co-allocation technique. They develop **Replica Convoy (ReCon)** [21], a tool for retrieving data from multiple replicas simultaneously where replicas that are predicted to deliver data faster are assigned a larger portion to service.

C. Predictive Techniques - Network Probes

Feng and Humphrey develop data retrieval techniques that utilize Network Weather Service (NWS) [17] predictions to specify the amount of data to be requested from replica servers [7]. The NWS is a distributed system that detects the network status at periodical intervals. Other mechanisms can be used to determine the status of connections between users and servers. Zhou et al. present a **probe-based data retrieval technique** [21], where a fixed sized pinging mechanism is used to probe network connections and determine network output. Based on the data returned by the probes, varying amounts of data are assigned to each replica.

D. Dynamic Techniques - Equal Request Sizes

The following retrieval techniques dynamically adapt to changing network conditions by requesting small, equally sized, portions of a file from multiple replicas. Vazhkudai develops a **conservative load balancing technique** that dynamically adapts to changing network and system conditions [9], [10]. The amount of data requested for a given server is decided dynamically instead of being based on previous history. The desired data file is divided into equal sized, disjoint blocks. Each available server is initially assigned one block to service in parallel. Once a server delivers the block, another block is assigned until the entire file is retrieved. Faster servers will transfer larger portions of the file. Feng and Humphrey also develop a similar dynamic data co-allocation algorithm called, **NoObserve** [7].

E. Dynamic Techniques - Varying Request Sizes

The techniques described in the previous section divide the desired data file into equal sized disjoint blocks. Other data retrieval techniques try to improve performance by varying the size of the blocks based on the performance of the replica servers. Faster servers are assigned larger blocks.

Vazhkudai develops an **aggressive load-balancing technique** [9], [10], which is a modified version his conservative load-balancing technique that was discussed in the previous section. Instead of requesting a single block from each replica, the amount of data requested from faster servers is progressively increased. The amount of data requested from slower servers is decreased or stopped completely.

The **recursively-adjusting co-allocation technique** [18]–[20] developed by Yang et al. is a combination of dynamic and predictive techniques, since it utilizes Network Weather Service forecasts. This technique works by continually adjusting the amount of data requested from each replica server to correspond to its real-time bandwidth during file transfers. The technique begins by dividing the desired data file into several sections. Each of these sections is then sub-divided into varying sized blocks that are individually assigned to all replicas. The number and size of the larger sections is variable and can be adjusted by the user. The size of each section is a percentage of the remaining file size to be retrieved. Each section size will therefore be progressively smaller than previous sections. The user can select the smallest section size that is used.

Another dynamic data retrieval technique, similar to the previous recursive mechanism, which varies the amount of data requested from each server while still dividing the data file into blocks is the **MSDT algorithm** [16] developed by Wang et al. The MSDT algorithm is a combination of dynamic and predictive techniques, as it utilizes the past transfer histories for predictions. The algorithm uses the overhead and bandwidth of previous segment transfers to predict the future performance of a replica. The amount of segments that are assigned varies depending on the transfer history for the particular replica.

F. Dynamic Techniques - Preemptive Measures

The dynamic techniques in the two previous sections retrieve portions of the data file from multiple replica servers. The amount of data retrieved may vary depending on the algorithm, however there is the possibility that a client will end up waiting for slower servers to deliver portions of the file. The previous techniques do not preempt transfers or redistribute the workload to other servers when replicas become unresponsive.

The ReCon data retrieval service [21] designed by Zhou et al. offers a **Greedy retrieval algorithm** where the desired

data file is divided into equal sized segments. Each replica is initially assigned one segment. As servers complete their segments, they are assigned additional segments to service. A recursive scheduling mechanism handles any errors that occur. If a transfer fails, the mechanism automatically reschedules the failed data request to another replica that is currently transferring data.

Bhuvan et al. develop a different preemptive data co-allocation mechanism, the **Dynamic Co-allocation Scheme with Duplicate Assignments (DCDA)** [3]. This technique is used to cope with highly inconsistent network performance of replica servers. In their algorithm, the desired data file is divided into disjoint blocks of equal size. Each server is initially assigned one block to service. When a server completes a request, it is assigned another outstanding block. The algorithm continues until all blocks have been assigned. If a server delivers a block and there are no blocks remaining that have not been initially assigned, the server will be given an outstanding block request that has not been completed. There will now be several servers working on the same request. When a server delivers a request, all other servers are notified to stop serving this request.

Chang et al. [6] develop an advanced preemptive technique, **Multiple Parallel Downloads with Bandwidth Considerations technique**, that considers both server output throughput and client input bandwidth when assigning workloads to the replica servers. This paper is the first to discuss their technique in terms of multiple users. They realize that when everyone uses parallel downloads, they will compete for system resources that causes a degradation of system efficiency and unfairness for the users. They also determine that a server should not outdo its capacity by serving too many clients and a client should not download from too many servers with its limited incoming bandwidth. The authors discuss a multiple user environment and provide an example of how their technique would work with six users accessing a small number of files. Their experiments however, do not show the performance of their algorithm when many users are simultaneously utilizing their technique.

III. Evaluations

In order to evaluate the impact of Big Data transfer techniques on storage performance, the DiskSim storage simulator is utilized to emulate real workloads. DiskSim has been proven to correctly emulate real storage devices [4]. The hard disk drive model selected for the DiskSim simulations is the Seagate Cheetah 9LP, which provides fast disk service times due to its high-end configuration. The most recent iteration of the Cheetah drive similarly provides high performance and the results of the simulations are scaled to match the most recent drive specification.

Metric	Cheetah9LP	Cheetah15K.7
Capacity	9.10 GB	600 GB
Avg. Seek Time	5.4 msec	3.4 msec
Transfer Rate	23.95 MByte/sec	204 MByte/sec
Rotation Speed	10,025 RPM	15,000 RPM
Rotate Latency	2.99 ms	2.0 ms

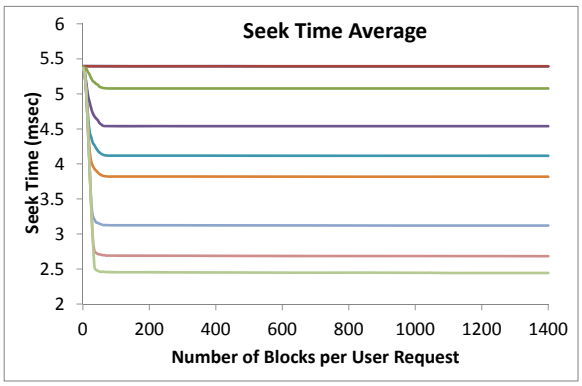

Fig. 1. Changes in the average seek time as the number of blocks per request increases. The top line represents 1 user and the bottom line represents 20 users.

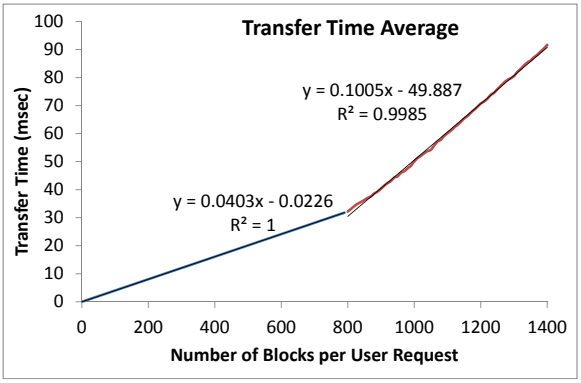

Fig. 2. Changes in the average transfer time as the number of blocks per request increases.

A. Initial Evaluations

In the initial evaluations, a varying number of users are configured in the simulator for a restricted number of blocks per disk request. Experiments are run for the following number of concurrent storage users: 1, 2, 3, 5, 6, 10, 15, and 20. The size of each request is fixed for each experiment and the number blocks requested is varied from 1 to 1400 blocks throughout the series of experiments. Several performance metrics are monitored and evaluated for each experiment. Metrics for the physical disk performance, as well as all users requests, are carefully monitored.

The physical disk metrics are examined for all of the experiments. Figure 1 shows the changes in the average seek time as the number of blocks per request increases. The average seek time actually reduces as the number of users in the system increases. This is due to the larger number of requests being serviced and likelihood that the next request to be serviced is close to the current location of the read/write head of the disk. The changes in transfer time are demonstrated in Figure 2. The average transfer time was almost identical for any number of concurrent users. The determining factor for the transfer time is solely the number of blocks being requested. The shape of the graph is interesting in that the transfer time increases at a slower rate until 800 blocks and after that point there is a marked increase in the transfer rate for larger request

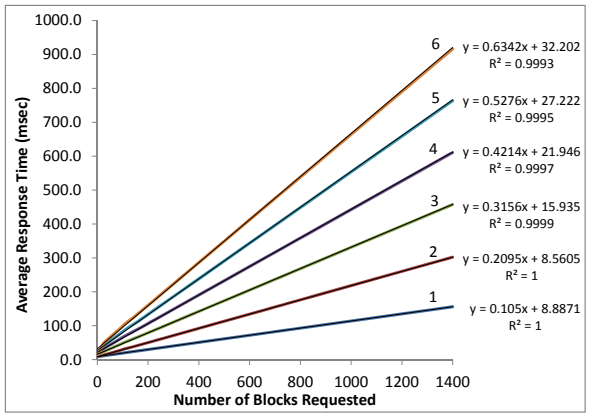

Fig. 3. Changes in the average system response time as the number of blocks per request increases. The bottom line represents 1 user and the top line represents 6 users.

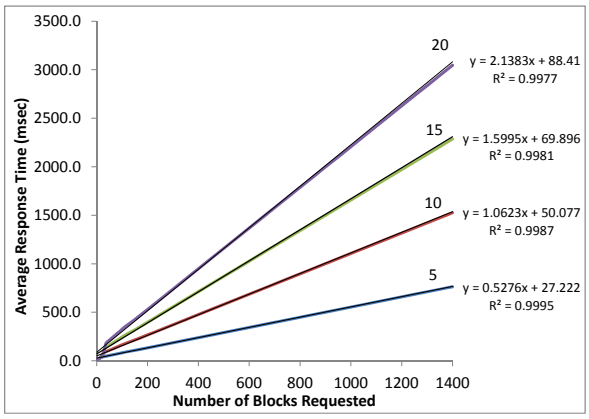

Fig. 4. Changes in the average system response time as the number of blocks per request increases. The bottom line represents 5 users and the top line represents 20 users.

sizes. This is confirmed by the larger coefficient on the trend lines for each section of the graph.

After examining the physical disk access metrics, the user request response times are evaluated. Figure 3 illustrates the system response time for the various sized requests when there are between 1 and 6 concurrent users utilizing the disk. The system response time is the total time that it takes to service a user's request and includes both disk access time and queue time. The graph shows that the average system response time increases as the number of blocks being requested increases. As one would expect, larger requests have larger response times. The increase in response time is linear and can be approximated using trend lines. The graph also indicates well-fitting trend lines, r squared values close to 1.

Figure 4 illustrates the system response time for the various sized requests when there are 5, 10, 15, and 20 concurrent users utilizing the disk. Similar to the previous graph, the increase in response time is linear. As the number of concurrent users grows, the average system response time also increases at a linear rate.

B. Big Data Transfers

Using the extremely well fitting trend lines that were calculated in the initial evaluations, the simulations can be scaled to emulate Big Data transfers that represent terabytes of data being requested from a storage system. In the next group of experiments, the size of the total data set being requested from the storage system is varied from 1 TB to 100 TB using the scaled DiskSim results in a custom designed simulation system.

When a user requests multiple terabytes of data, this type of request requires reading from multiple disks. In these simulations, a user's request is serviced in a serial, one disk at a time fashion. The request could be serviced by multiple hard disks in parallel, but it is unlikely that the user would have the bandwidth required to support the read rate supplied by simultaneous disk transfers. It is because of this rationale that the users' requests are serviced in a serial fashion and can be viewed as one request to a very large, logical disk.

The individual request sizes sent by the users are also varied in the evaluations from 1MB to 10GB. As described in the related work section, Big Data transfer techniques often divide the entire data set into chunks that are systematically retrieved from multiple replicas. The size of these chunks can vary depending on the algorithm and user configuration. These evaluations represent a set of possible file sizes that could be utilized by these Big Data transfer techniques.

In the following results, the total response time for retrieving a Big Data set is evaluated by comparing two approaches to Big Data transfers: single request and multiple requests. The single request approach issues only one request to the storage system for the entire data set. The multiple request approach issues multiple, smaller requests for the entire data set. In all of the evaluations, it was found that issuing a single request never incurred a penalty and frequently provided marked performance improvements over multiple, smaller requests. In order to better illustrate the performance differences between these two approaches, the **extra time** required when using the multiple request approach is graphed. The following graphs show how much longer a user would have to wait to retrieve the entire data set from the disk when using multiple, smaller requests instead of a single, large request.

Figure 5 demonstrates the significant performance impact of using a very small request size (1 MB) to retrieve a large data set. The lines on the graph represent a varying number of users in the system. The bottom line represents 1 user and the top line represents 20 users. As the size of the total data set increases and the number of users sharing the disk increases, the amount of additional time required retrieve the entire data set greatly increases. In the most extreme case, a user could potentially spend an additional 2500 hours (104 days) trying to retrieve a 100TB data set using 1 MB request sizes with 19 other users in the system performing the same task. For a single user, the impact is not as severe, especially for smaller total data set sizes. A 10TB data set would take a day longer using only 1MB requests, however retrieving a 100TB using the same technique would take almost 11 days longer than issuing a single request.

In reality, most users would not attempt to retrieve a large data set 1MB at a time; however some of the algorithms

specified in the related work section in theory could be configured to issue requests of this granularity. The next group of evaluations have the user request sizes fixed at 10MB and Figure 6 illustrates the time increase when issuing these 10MB requests for the large data sets. The overall time increases for retrieving the entire data set are less than the 1MB file requests, but the increases are still significant. With 10 users in the system, a 50TB data set would take an additional 73 hours to transfer the data from the storage system using 10MB sized requests.

The next two groups of evaluations examine the extra time that it would take to retrieve the entire data set when the file request sizes are raised. Figure 7 shows the time increases for 100MB file requests. A 100TB data set would take 26 hours longer using the 100MB file request size with 19 other users accessing the system. The time increase is reduced to 3 hours when the user is the sole workload in the system. Figure 8 shows the time increases for 500MB file requests. With the larger request size, the 100TB data set would only take an hour longer than retrieving the entire data set in a single request with only 1 user in the system.

The final three groups of evaluations raise the individual file requests to 1GB, 5GB and 10GB. Figure 9 shows the time increases for 1GB file requests and Figure 10 shows the time increases for 5GB file requests. As the user issues requests for larger portions of the data set, the extra time required is decreased. With 5GB file requests, the extra time for issuing multiple requests is reduced to minutes instead of hours. A final group of evaluations are run with 10GB file requests and the increase in total time is further reduced to less than 15 minutes even with 20 concurrent user requests in the system.

IV. Summary and Future Work

From the evaluations, it is evident that user request sizes have an enormous impact on storage performance when retrieving Big Data sets. When the number of concurrent users sharing the storage system increases, this impact only grows more important. As the file size of the individual storage requests increased, the extra time required for multiple request based Big Data transfer techniques decreased. Requests that represent a larger percentage of the total data set performed significantly better than smaller percentage requests. This is a clear indication that users should always attempt to retrieve the largest possible subset of data from each server involved in a transfer. Another aspect to storage, which is not specifically examined in any of the graphs, is prefetching. Storage systems will often pre-fetch and cache data when sequential data streams are detected. By issuing larger requests, the system can potentially decrease the physical access penalties of the storage system for the user.

Big Data transfers are becoming more commonplace amongst users around the world. As more Big Data sets are created, the need for efficient transfers of these massive file sets will grow. Research studies must consider both network and storage impacts when evaluating transfer performance. For future work, an examination of Big Data transfers workloads from a live storage system should be conducted in order to assess typical user request sizes. Big Data transfer techniques that consider both network and storage performance should also be developed and investigated further.

References

[1] W. 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 Journal*, vol. 28, no. 5, pp. 749–771, May 2002. [Online]. Available: citeseer.ist.psu.edu/article/allcock01data.html

[2] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[3] R. S. Bhuvaneswaran and et al., "Redundant parallel data transfer schemes for the grid environment," in *ACSW Frontiers*, 2006.

[4] J. Bucy and G. Granger, "The disksim simulation environment, version 3.0," Reference Manual. Technical report, School of Computer Science, Carnegie Mellon University, 2003.

[5] CERN, "Lhc physics data taking gets underway at new record collision energy of 8tev," *http://press.web.cern.ch*, 2012.

[6] R.-S. Chang, M.-H. Guo, and H.-C. Lin, "A multiple parallel download scheme with server throughput and client bandwidth considerations for data grids," *Future Generation Computer Systems*, vol. 24, no. 8, pp. 798–805, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/B6V06-4S9P5JN-1/2/837940f81cceafd5ec0530511a40834b

[7] J. Feng and M. Humphrey, "Eliminating replica selection - using multiple replicas to accelerate data transfer on grids," in *ICPADS*, 2004, p. 359.

[8] A. Riska and E. Riedel, "It's not fair - evaluating efficient disk scheduling," in *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, 2003, pp. 288 – 295.

[9] S. Vazhkudai, "Enabling the co-allocation of grid data transfers," in *GRID*, 2003, p. 44.

[10] ——, "Distributed downloads of bulk, replicated grid data," in *Journal of Grid Computing*, vol. 2, no. 1, March 2004, pp. 31–42.

[11] A. H. Villa and E. Varki, "Co-allocation in data grids: A global, multi-user perspective," *Advances in Grid and Pervasive Computing*, pp. 152–165, 2008.

[12] ——, "It takes know-how to retrieve large files over public networks," in *1st International Conference on Advanced Computing and Communications (ACC)*, 2010.

[13] ——, "The feasibility of moving terabyte files between campus and cloud," in *23rd IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, December 2011.

[14] ——, "Automating large file transfers," in *13th International Conference on Internet Computing (ICOMP'12)*, July 2012.

[15] ——, "Characterization of a campus internet workload," in *27th International Conference on Computers and their Applications (CATA)*, March 2012.

[16] C.-M. Wang, C.-C. Hsu, H.-M. Chen, and J.-J. Wu, "Efficient multi-source data transfer in data grids," in *CCGRID*, 2006, pp. 421–424.

[17] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Gener. Comput. Syst.*, vol. 15, no. 5-6, pp. 757–768, 1999.

[18] C.-T. Yang, Y.-C. Chi, and C.-P. Fu, "Redundant parallel file transfer with anticipative adjustment mechanism in data grids," in *Journal of Information Technology and Applications*, vol. Vol. 1, no. No. 4, March 2007, pp. 305–313.

[19] C.-T. Yang, I.-H. Yang, C.-H. Chen, and S.-Y. Wang, "Implementation of a dynamic adjustment mechanism with efficient replica selection in data grid environments," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 797–804.

[20] C.-T. Yang, I.-H. Yang, K.-C. Li, and S.-Y. Wang, "Improvements on dynamic adjustment mechanism in co-allocation data grid environments," *The Journal of Supercomputing*, vol. 40, no. 3, pp. 269–280, 2007.

[21] X. Zhou, E. Kim, J. W. Kim, and H. Y. Yeom, "Recon: A fast and reliable replica retrieval service for the data grid," in *CCGRID*, 2006, pp. 446–453.

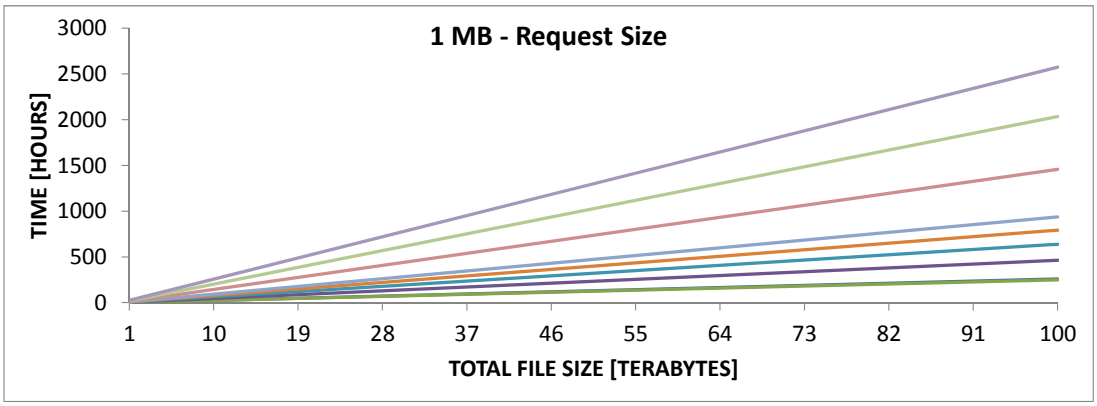

Fig. 5. This graph illustrates the extra time required (in hours) that would be required when issuing multiple smaller requests (1MB in size) instead of issuing a single request for the entire data set, as the size of the data set increases. The bottom line represents 1 user and the top line represents 20 concurrent users.

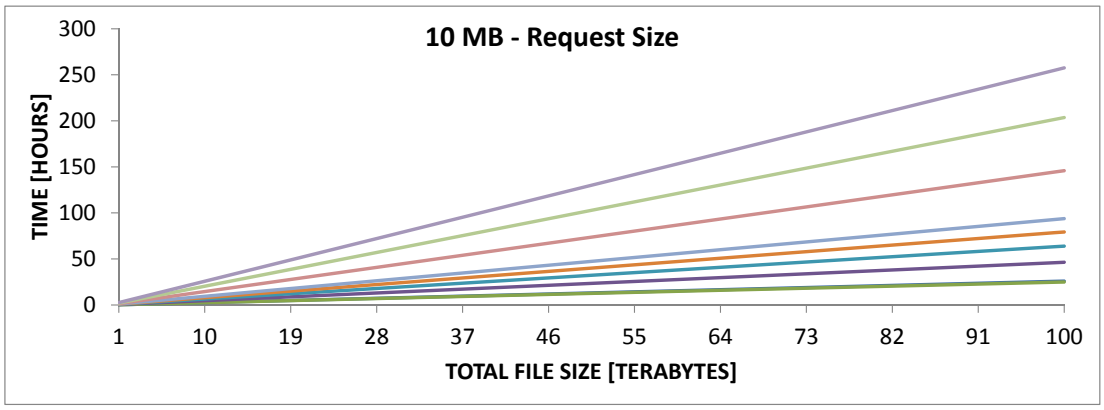

Fig. 6. This graph illustrates the extra time required (in hours) that would be required when issuing multiple smaller requests (10MB in size) instead of issuing a single request for the entire data set, as the size of the data set increases. The bottom line represents 1 user and the top line represents 20 concurrent users.

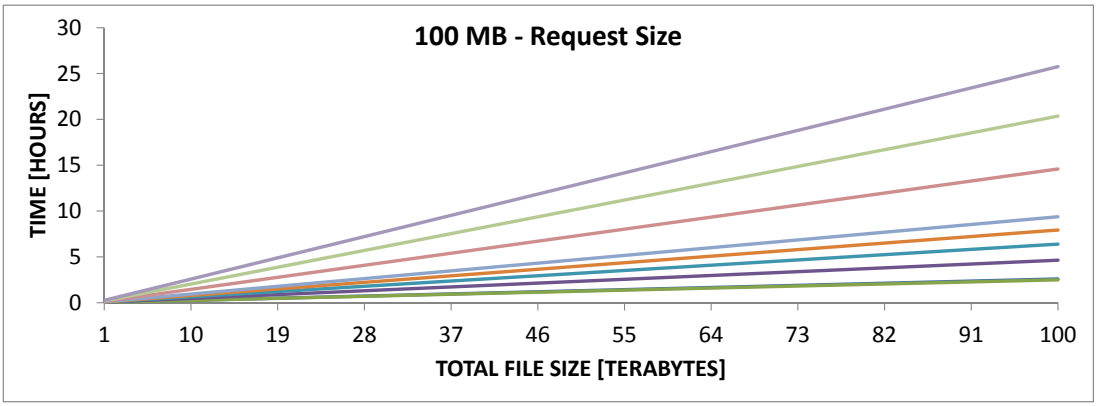

Fig. 7. This graph illustrates the extra time required (in hours) that would be required when issuing multiple smaller requests (100MB in size) instead of issuing a single request for the entire data set, as the size of the data set increases. The bottom line represents 1 user and the top line represents 20 concurrent users.

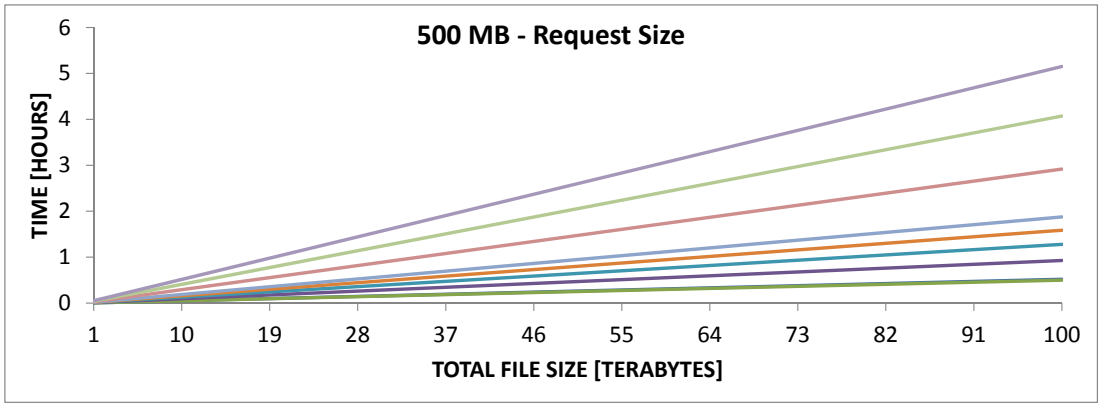

Fig. 8. This graph illustrates the extra time required (in hours) that would be required when issuing multiple smaller requests (500MB in size) instead of issuing a single request for the entire data set, as the size of the data set increases. The bottom line represents 1 user and the top line represents 20 concurrent users.

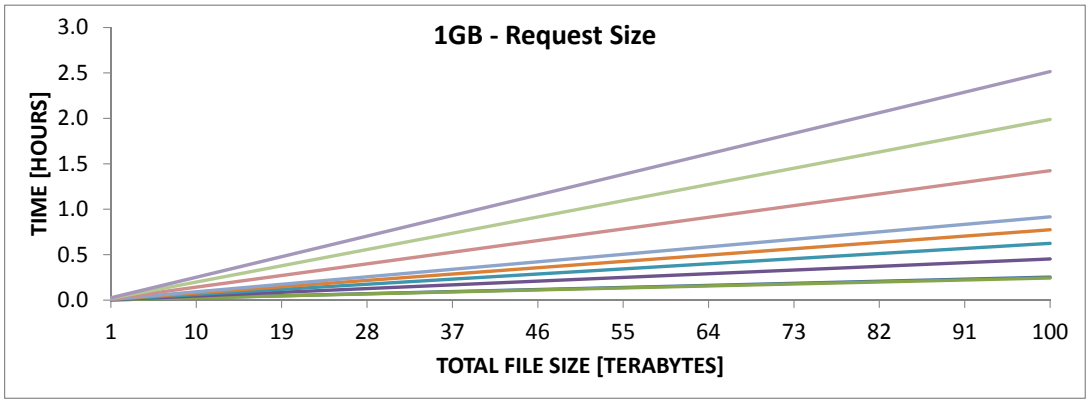

Fig. 9. This graph illustrates the extra time required (in hours) that would be required when issuing multiple smaller requests (1GB in size) instead of issuing a single request for the entire data set, as the size of the data set increases. The bottom line represents 1 user and the top line represents 20 concurrent users.

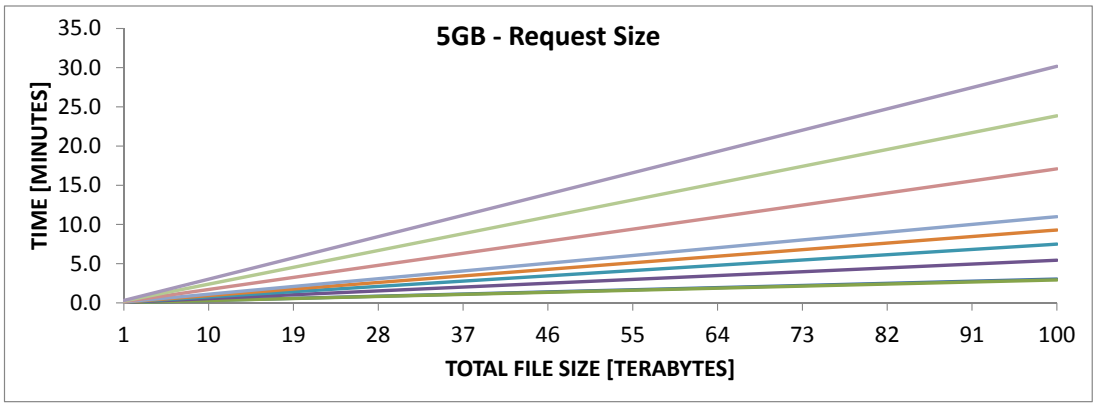

Fig. 10. This graph illustrates the extra time required (in hours) that would be required when issuing multiple smaller requests (5GB in size) instead of issuing a single request for the entire data set, as the size of the data set increases. The bottom line represents 1 user and the top line represents 20 concurrent users.