2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

Experimental Analysis of File Transfer Rates Over Wide-Area Dedicated Connections

Nageswara S. V. Rao, Qiang Liu, Satyabrata Sen, Greg Hinkel, and Neena Imam
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831
{raons,liuq1,sens,hinkelgc,imamn}@ornl.gov

Ian Foster and Rajkumar Kettimuthu
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{foster,kettimut}@anl.gov

Bradley W. Settlemyer
Systems Integration Group
Los Alamos National Laboratory
Los Alamos, NM 87545
bws@lanl.gov

Chase Q. Wu and Daqing Yun
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102
{chase.wu,dy83}@njit.edu

***Abstract*—File transfers over dedicated connections, supported by large parallel filesystems, have become increasingly important in high-performance computing and big data workflows. It remains a challenge to achieve peak rates for such transfers due to the complexities of file I/O, host, and network transport subsystems, and equally importantly, their interactions. We present extensive measurements of disk-to-disk file transfers using Lustre and XFS filesystems mounted on multi-core servers over a suite of 10 Gbps emulated connections with 0–366 ms round trip times. Our results indicate that large buffer sizes and many parallel flows do not always guarantee high transfer rates. Furthermore, large variations in the measured rates necessitate repeated measurements to ensure confidence in inferences based on them. We propose a new method to efficiently identify the optimal joint file I/O and network transport parameters using a small number of measurements. We show that for XFS and Lustre with direct I/O, this method identifies configurations achieving 97% of the peak transfer rate while probing only 12% of the parameter space.**

***Index Terms*—Wide area transport, dedicated connections, TCP, RTT, throughput, file I/O read and write, profiling, profiling overhead.**

I. Introduction

In recent years, there has been an increasing demand for wide-area data transfers in a number of scenarios involving high-performance computing (HPC) work flows, cloud computing server complexes, and big data computing facilities. These transfers often involve disk-to-disk file transfers between remote sites, for example, between a supercomputing facility and a remote storage site. To support these transfers in HPC scenarios, the underlying infrastructures are being enhanced: (a) networks, such as the Department of Energy's (DOE) ESnet, provide on-demand, dedicated connections [1]; (b) high-performance filesystems, such as Lustre [18], are deployed with large collections of disk drives to provide site-wide access; and (c) dedicated hosts, such as Data Transfer Nodes (DTNs) [7], are deployed that employ specialized transfer protocols such as GridFTP [2] and transfer management software such as Globus [3].

High-performance disk-to-disk transfers require the composition of complex file I/O and network subsystems, and host orchestration. For example, the Lustre filesystem employs multiple Object Storage Targets (OSTs) to manage collections of disks, multiple Object Storage Servers (OSSes) to stripe file contents, and distributed MetaData Servers (MDSes) to provide site-wide file naming and access. Such complex filesystem must be effectively coupled with DTNs and wide-area networks to achieve peak file transfer rates. But, sustaining high file transfer rates requires *joint* optimization of subsystem parameters to account for the impedance mismatches among them [22]. For Lustre filesystems, for example, important parameters include the stripe size and number of stripes for the files, typically specified at the creation time, and the number of parallel I/O threads for read/write operations, specified at the transfer time. Typically, I/O buffer size and the number of parallel threads are chosen to be sufficiently large to sustain high throughput, but as we will see, this simple heuristic is not always optimal. Key parameters for TCP network transport include the choice of congestion control module, number of parallel streams, and various buffer sizes [8]. We focus here on choosing the number of parallel flows for file I/O and network transport, two critical factors in determining file transfer rates.

Throughput profiles for network transport and file I/O are typically generated by sweeping the values for chosen parameters, using, for example, *iperf* and *xddprof*, respectively [23]. Given those profiles, we might hope to determine an optimal profile for disk-to-disk file transfers via a simple composition of individual network transport and file I/O profiles: for example, by using the minimum of subsystem throughput maxima as the transfer rate for a particular parameter combination. However, increasingly complex file I/O and network transport subsystems and inter-subsystem interactions mean that such simple compositions can overestimate transfer rates. For example, for wide-area file transfers over 10 Gbps networks with files stored in Lustre filesystems and striped across 8 storage servers, accessed with 8 MB buffers, and with 8 I/O

978-1-5090-4297-5/16 $31.00 © 2016 IEEE
DOI 10.1109/HPCC-SmartCity-DSS.2016.101

and TCP threads, we observe disk-to-disk transfer rates of only 1.5 Gbps [20]. Yet the peak throughputs of the file and network subsystems are each close to 10 Gbps.

To study these issues in more depth, we measured file I/O and network throughput and file transfer rates over Lustre and XFS filesystems for a suite of seven emulated connections in the 0–366ms RTT range. We learned that: (a) conventional practices of large buffers and higher parallelism do not always translate into higher transfer rates; (b) direct I/O methods that avoid file buffers at the hosts provide higher wide-area transfer rates, and (c) significant statistical variations in measurements, due to complex interactions of non-linear TCP dynamics with parallel file I/O streams, necessitate repeated measurements to ensure confidence in inferences based on them.

These results suggest a need for complete joint file transfer rate profiles. However, building such joint profiles via brute-force parameter sweeps is not feasible in operational environments, as it would take weeks to months, during which the system is unavailable to users. Thus we propose a new $d-w$ (for *depth-width*) method that we show can identify close-to-optimal joint parameters with significantly fewer measurements. By exploiting the overall *unimodality*, namely, having a single maximum, of profiles, this method implements a stochastic gradient method using d repeated measurements over w-sized windows. It also uses domain knowledge extracted from measurements to identify the starting parameters for the search. We evaluate its performance using Lustre and XFS measurements, and show that it reduces measurement times to hours and days on those two systems, respectively. More specifically, for both filesystems, the computed parameters achieved 97% of peak transfer rate while probing only 12% of the parameter space.

The rest of the paper is as follows. We summarize related work in Section II; describe network transport and file I/O subsystems, and XDD file transfers, in Section III; present measurements of file transfer rates over emulated connections in Section IV; and describe our $d-w$ method and its performance analysis in Section V. We conclude in Section VI.

II. Related Work

Tools for choosing parameter values to maximize file transfer performance have been a frequent area of interest. For TCP, dynamic right-sizing of buffers has been used to improve file transfer throughput [9], [11]. The GridFTP-APT project [14] develops models that identify TCP buffer sizes and number of TCP flows for improved transfer performance [12], and builds tools for dynamically changing the number of connections during a file transfer [13]. There is also previous work on estimating optimal parallel storage system parameters [4], [6], [17]; however, that work does not consider wide-area networking models. Our approach is unique in simultaneously optimizing both network and storage system parameters, and using measurements rather than coarse-grain TCP behavior to estimate optimal parameters.

Our work is also distinguished by its efficient method for arriving at the selected parameters without exhaustive search. The $d-w$ scheme described in this paper is similar to existing statistical blocking techniques, such as Latin Squares [27] and Latin Hypercubes [19]; however, it exploits the overall unimodality of the response regression surfaces to converge more quickly on the optimal parameter set. In this respect, it is similar to the more computationally intensive machine learning techniques, such as stochastic gradient descent [5], but unlike those methods it does not require significant amounts of training data for each source-destination pair. Our method can be viewed as a version of the stochastic approximation method [16], [25], but with a derivative computation customized to the observed profiles. As a result, it provides confidence estimates based on finite samples rather than traditional asymptotic convergence results.

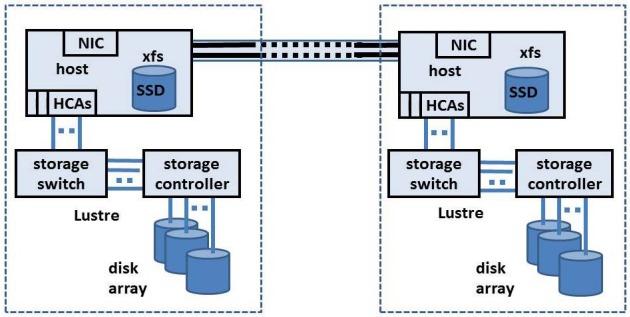


Fig. 1: File transfers over long-haul connections

III. Wide-Area File Transfers

A wide-area disk-to-disk file transfer encompasses storage devices, data transfer hosts, and local- and wide-area connections: see Fig. 1. Major sites often use dedicated data transfer hosts, such as DTNs, with high performance Network Interface Cards (NICs) to access network connections and Host Channel Adapters (HCAs) to access network storage systems. Transfers also involve a range of software components, including filesystem I/O modules for disk access and the TCP/IP stack for network transport, which are used by the file transfer software such as GridFTP [2] and XDD [28] running on the hosts.

A. Experimental Setup

We measured file transfer rates for both Lustre and XFS filesystems using XDD (described in next section) between two dedicated 48-core Linux servers over emulated 10 Gbps connections for RTT, τ = 0.4, 11.6, 22.6. 45.6, 93.6, 183, and 366 ms. The 0.4, 11.6, 22.6, 45.6 ms RTTs correspond to cross-country connections, the 93.6 and 183 ms RTTs to intercontinental connections, and the 366 ms RTT to a connection spanning the globe. The connections are emulated in hardware using ANUE-ixia devices to which host 10GigE interfaces are directly connected: see Fig. 2. The network packets are sent to the emulator, which delays them based on the specified RTT, a process that closely matches the transport over a physical connection: in particular, TCP dynamics and file transfer rates are more closely matched than when using simulators such as ns-3 and OPNET. Fig. 1 shows the filesystem configurations

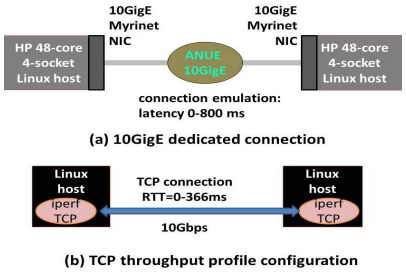


Fig. 2: Testbed configurations of emulated long-haul connections

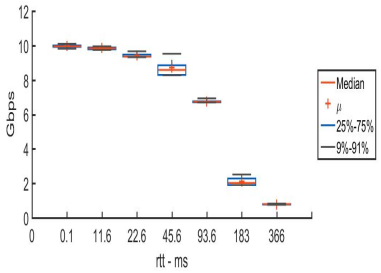


Fig. 3: Throughput profile of TCP CUBIC as a function of RTT with 8 streams

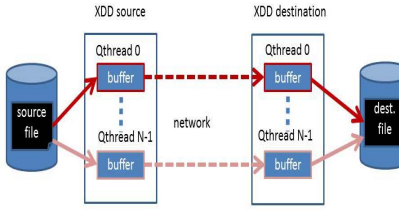


Fig. 4: Qthreads implement parallel XDD disk-to-disk flows

used: Lustre is mounted over a local InfiniBand network and XFS is mounted locally on each host over SSDs connected to its PCI bus. The XFS and Lustre filesystems provide peak file I/O throughput greater than 10 Gbps, and thus it is the network link rather than the filesystems that are a limiting factor for these file transfer rates.

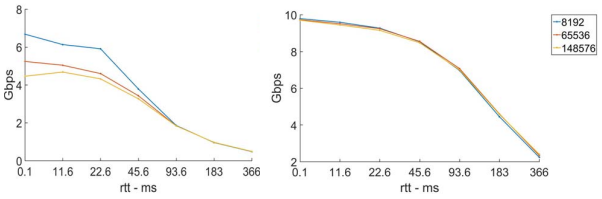
B. XDD File Transfers

A single XDD file transfer process spawns a set of threads to open a file and perform data transfers between either storage and memory or memory and network. To initiate a file transfer, it creates a set of source and destination XDD processes that are paired as shown in Figure 4. A source XDD process creates a *TargetThread* that opens the file, initiates a connection with a destination XDD process, and subsequently creates a number of *QThreads* that issue read commands to fill a thread-local buffer. Once a thread's buffer is filled, that thread transmits the data over the network to a destination XDD process; the size of the buffer is referred to as the *request size*. Similarly, the destination XDD process creates a TargetThread that listens for a connection from a source XDD process and then creates QThreads that receive data from the network and write the data into the storage system. The number of source and destination QThread pairs is equal to the number of TCP parallel streams, and hence we refer to each source-destination QThread connection as a *flow*. XDD reports *read* transfer rate at the sender and *write* transfer rate at the receiver for each file transfer by aggregating across all flows.

An XDD file transfer rate profile is a complex composition of file I/O and network throughput profiles. The file transfer dynamics depend in particular on the complex non-linear, possibly chaotic [10], TCP dynamics, modulated by those of file I/O systems; which lead to high statistical variations in measured file transfer rates, as we show in Section IV.

C. Network Transport and File I/O Profiles

We characterize the throughput of network transport and disk file I/O by sweeping over chosen parameters to create individual *throughput profiles* for each component that plot performance as a function of parameter values for data transfers in 1–100GB range. For network transport, we vary three parameters, namely the number of parallel threads, TCP congestion control protocol used, and RTT. We set the host-level TCP/IP buffer sizes to the recommended values for



(a) 1 flow (b) 8 flows

Fig. 5: XDD memory transfer rates for different request sizes (KB)

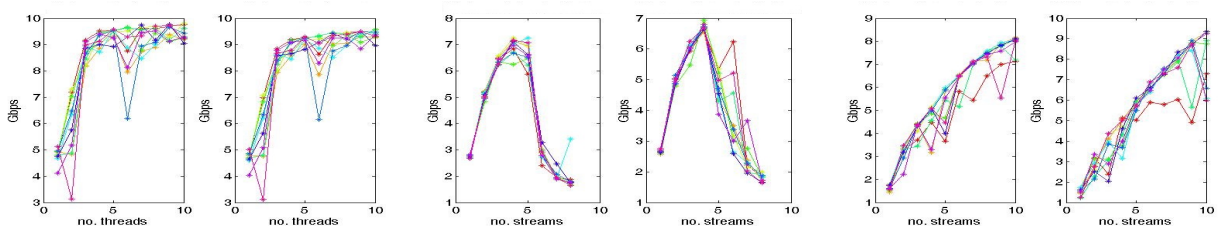
200 ms RTT [26] to cover most cross-country and inter-continental connections and then use iperf to collect TCP throughput measurements over these connections while varying both number of parallel threads and congestion control scheme. Fig. 3 shows the profile obtained when using the CUBIC [21] congestion control module, the default on Linux systems. We repeated the measurements using Hamilton TCP [24] and Scalable TCP [15] congestion control modules and saw throughput rates within a few percent of CUBIC results; hence we do not show those results here. For disk I/O, we use a tool called xddprof to measure performance while varying three parameters: number of parallel read/write threads, buffer sizes, and (for Lustre) stripe sizes and numbers and the choice of either direct or default I/O.

IV. Disk-to-Disk Transfer Measurements

We collected two sets of XDD disk-to-disk file transfer measurements, one from XFS to XFS and one from Lustre to Lustre. Each experiment was repeated 10 times; the repetitions can be regarded as independent of one another. We considered both buffered I/O (the Linux default) and direct I/O options for Lustre. In the latter, XDD avoids the local copies of files on hosts by directly reading and writing into its buffers, which significantly improves the transfer rates.

A. Memory-to-Memory Transfers

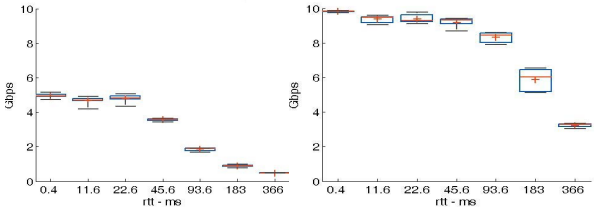
To assess overheads introduced by XDD in its use of TCP flows, we decoupled the filesystems to measure the memory-to-memory transfer rates between XDD's sender and receiver buffers. We show in Fig. 5 results for both 1 and 8 flows as a function of RTT and for different request sizes. With 1 flow, XDD transfer rates vary with request size but are consistently lower than the TCP rates shown in Fig. 3. The highest transfer



(a) XFS read (Left) / write (Right) (b) Lustre default I/O write: 2 (L) / 8 (R) stripes (c) Lustre direct I/O write: 2 (L) / 8 (R) stripes

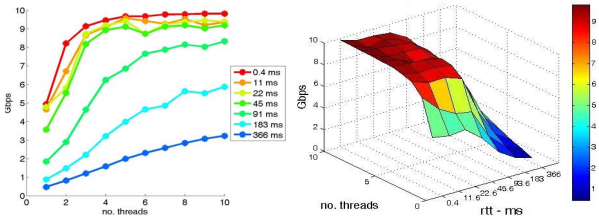
Fig. 6: Transfer rates for RTT = 22.6 ms as a function of number of flows (used interchangeably with the terms "threads" for XFS and "streams" for Lustre). Each line is a different experiment.

rates are achieved with the lowest request size, a result that we attribute to the finer granularity of input data chunks delivered to TCP threads. With 8 flows, the XDD memory transfer rate closely matches the iperf throughput in Fig. 3, which indicates that XDD does not create transfer rate bottlenecks between network transport and file I/O for sufficient number of flows.



(a) 1 flow (b) 10 flows

Fig. 7: XFS file write transfer rates



(a) Write rates: Line plot (b) Write rates: Surface plot

Fig. 8: Mean XFS file write rates

B. XFS-to-XFS File Transfers

We next turn to disk-to-disk transfers. We shall see that file transfer rates are lower and variations more pronounced when filesystems are engaged, despite the fact that our filesystems are capable of greater read and write speeds than the network.

We first consider XFS-to-XFS transfers. The results in Fig. 6(a), for RTT = 22.6 ms, show wide variations among repeated experiments as shown in different colors. (As the read and write transfer profiles are similar for a given configuration, we present only the file write transfer rates in the figures that follow.) Fig. 7 shows the write results in box plot form, for 1 and 10 flows. We see that:

(a) Throughput increases with the number of flows. For instance, whereas the mean throughput peaks at 5 Gbps with 1 flow, the peak (occurring with 0.4 ms-RTT) rapidly jumps to above 9 Gbps with 4 flows, even closely approaching 10 Gbps with 7 flows. In fact, the same is largely true for other RTTs as well, which can be confirmed by the aggregate mean throughput line plot in Fig. 8(a).

(b) Mean throughput generally decreases with RTT, consistent with most data transfer protocols. The surface plots in Fig. 8(b) indicate an *monotonically increasing* trend, a special case of unimodality.

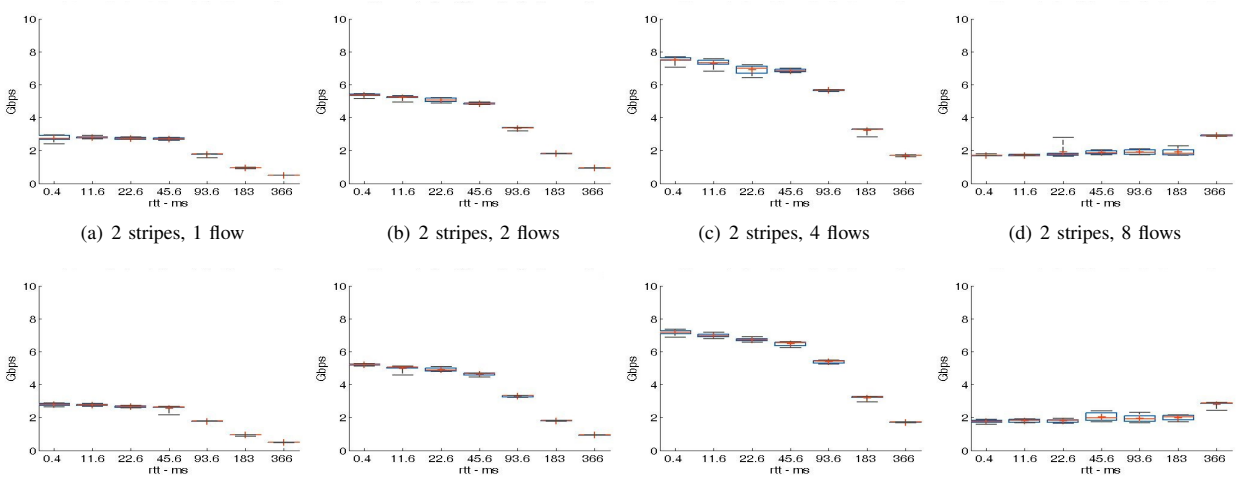
(c) The concave region of the throughput profile (with respect to RTT) is extended with more flows. In particular, as shown in Fig. 7, with 1 flow, the transition point from concave to convex profiles occurs at a much lower RTT than when more flows are used.

C. Lustre-to-Lustre File Transfers: Default I/O

In the default I/O Lustre setup, the number of flows varies from 1 to 8, and the number of stripes is either 2 or 8. Figs 9 shows write transfer rates with 2 and 8 stripes and with flows $n_f \in \{1, 2, 4, 8\}$. Compared to XFS, the overall throughput is much lower, especially for smaller RTTs. (Such differences become less pronounced as RTT increases.) One surprising result in Figs. 9(d) and (h) is that the mean throughput plummets below 2 Gbps (with the exception of the 366 ms-RTT case) and then actually *increases* with RTT.

Fig. 10 provides another perspective on the default I/O Lustre setup results. At lower RTTs, mean throughput peaks at 4 flows, starts to decrease with 5 flows, and takes a nosedive at 6 flows. The sharp drop is delayed at higher RTTs, with throughput peaking at 5 flows for 91 ms RTT and at 6 flows for 183 ms RTT, and increasing all the way through 8 flows for 366 ms RTT. The overall trend with respect to the number of flows is unimodal with respect to the number of flows: a somewhat more complex response than the monotonicity that we saw for XFS transfers and that we will see for Lustre with direct I/O, as discussed in the next subsection.

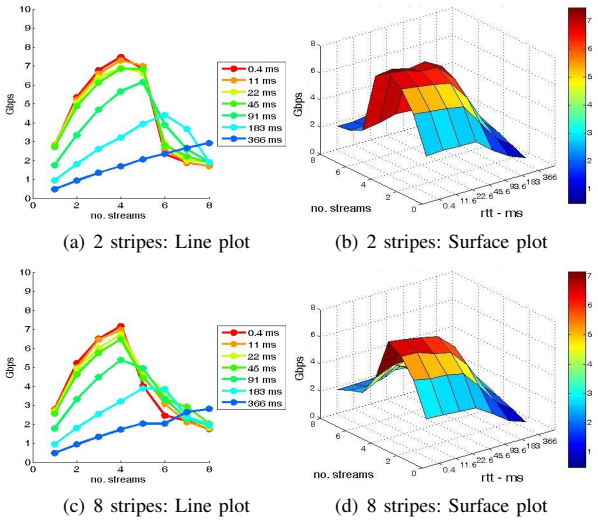
The box plots in Fig. 9 show some minor differences in 2 vs. 8 stripe performance. With 2 stripes, rates are somewhat higher at lower RTTs with 2 flows and 4 flows, whereas with 8 stripes we see slightly higher rates at higher RTTs with 8 flows. However, the line plots Figs. 10(a) and (c) show that the sharp drop in throughput, if any, occurs earlier, at 5 flows, when 8 stripes are used instead of 2 stripes, which demonstrates the longer concave regime (with respect to the number of flows)



(a) 2 stripes, 1 flow (b) 2 stripes, 2 flows (c) 2 stripes, 4 flows (d) 2 stripes, 8 flows

(e) 8 stripes, 1 flow (f) 8 stripes, 2 flows (g) 8 stripes, 4 flows (h) 8 stripes, 8 flows

Fig. 9: Default I/O Lustre file write rates, varying stripes and flows



(a) 2 stripes: Line plot (b) 2 stripes: Surface plot

(c) 8 stripes: Line plot (d) 8 stripes: Surface plot

Fig. 10: Mean default I/O Lustre file write rates

of the latter. This result is also confirmed by the surface plots: the "dome" in Fig. 10(d) is narrower than that in Fig. 10(b).

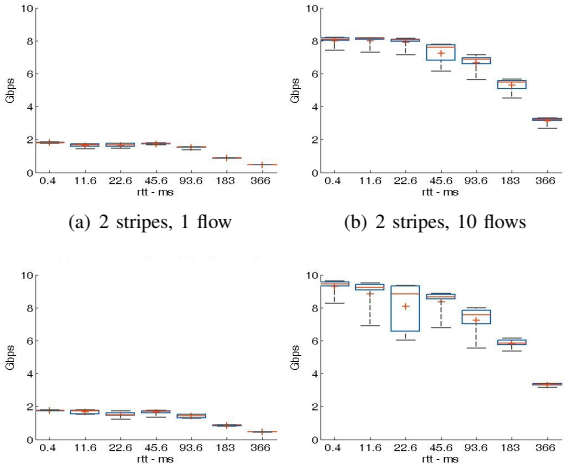
D. Lustre-to-Lustre File Transfers: Direct I/O

We repeated the experiments of the previous section using direct I/O Lustre. The results in Figs. 11 and 12 show that:

(a) Throughput with 1 flow is lower that for default I/O Lustre, but steadily increases with the number of flows. The direct I/O option also introduces more variation, as reflected in elongated box plots.

(b) Using 2 stripes yields somewhat higher transfer rates compared to 8 stripes for lower flow counts. With more flows, overall throughput is higher, and 8 stripes is the better option. Although the peak rates with 10 flows and 8 stripes are lower compared to XFS, they are significantly higher that with default I/O.

(c) We observe in Fig. 12 monotonicity trends similar to those seen for XFS in Fig. 8. The XFS surface, however, is higher and increases faster at lower RTTs, as a result of rates starting higher with one flow and further improving and approaching the peak with additional flows.

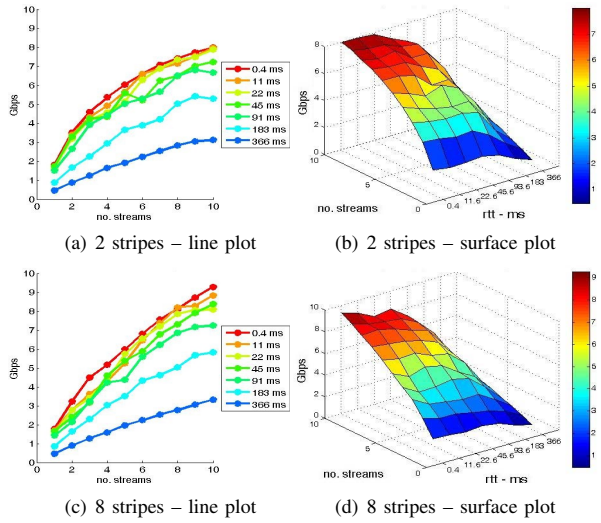


(a) 2 stripes, 1 flow (b) 2 stripes, 10 flows

(c) 8 stripes, 1 flow (d) 8 stripes, 10 flows

Fig. 11: Direct I/O Lustre file write rates, varying stripes and flows

In summary, file transfer rates for both XFS and Lustre are affected by the number of flows. The mean rate increases monotonically with number of flows for both XFS and direct I/O Lustre (Figs. 8 and 12), but degrades beyond a certain point for default I/O Lustre (Fig. 10). The number of stripes in Lustre seems to have less impact on transfer rate than does the number of flows. Additional measurements with request sizes of 65 MB and 145 MB show that the default 8 MB selection used in this paper consistently yields the best performance. This result is expected, as the smaller request size provides finer-resolution data chunks to TCP streams.



(a) 2 stripes – line plot (b) 2 stripes – surface plot

(c) 8 stripes – line plot (d) 8 stripes – surface plot

Fig. 12: Mean direct I/O Lustre file write rates

V. Fast Probing Method for Peak Transfer Rate

We now introduce and evaluate the performance of our fast probing depth-width ($d-w$) method.

A. The $D-W$ Method

Our $d-w$ method exploits the observed *unimodality* of transfer rates with respect to the number of flows; for XFS and Lustre with direct I/O, these rates exhibit the stronger monotonicity property. The basic approach is based on a stochastic gradient search method that starts with the largest number of flows and continues to collect measurements at different configurations while the gradient is positive within a certain window. Due to variations in measurements the gradient can only be estimated approximately and may not always provide the right direction, and indeed can lead to premature termination or over sampling. We mitigate such effects by repeating d measurements at each configuration and considering measurements within a w-window to jump over local variations; the former accounts for the variability and the latter attempts to avoid local maxima.

The method strategically probes multiple configurations with the goal of achieving the highest possible throughput with the fewest probes. Algorithm 1 sketches the method for XFS. (The Lustre version repeats measurements with 2 and 8 stripes at each stage and chooses the one with the higher rate.) We start with the highest flow count, $i = 10$, because we know that more flows typically yield higher transfer rates. We then repeatedly reduce the number of flows at each probing step by the width w. To reduce the risk of premature termination due to variability (e.g., due to insufficient observations), we require two consecutive decreases in the maximum of d traces for each configuration—that is, two consecutive negative gradient estimates—before we stop probing.

Intuitively, a larger d provides stable gradient estimates, thereby increasing the chance of reaching higher transfer rates, but at the cost of extra probing overhead. And, a larger w reduces the number of configurations to be probed, but may skip configurations that contain the true maximum. In particular, a larger w jumps over adjacent configurations whose peak rates are close enough to make the measured rates statistically similar; consequently, the probing process terminates faster.

Algorithm 1 The $d-w$ Algorithm for XFS

1: **initialize**: probe configuration: $i = 10$ flows d times;
2: $\mathcal{T}_{i,max} = \max_{j\in\{1,2,\ldots,d\}} \mathcal{T}_{i,j}$; $\mathcal{T}_{max} = \mathcal{T}_{i,max}$;
3: $flag \leftarrow 0$;
4: **while** $i > w$ and $flag < 2$ **do**
5: $\quad i \leftarrow i - w$; probe configuration with i flows d times;
6: $\quad \mathcal{T}_{i,max} = \max_{j\in\{1,2,\ldots,d\}} \mathcal{T}_{i,j}$;
7: $\quad$ **if** $\mathcal{T}_{i,max} > \mathcal{T}_{i-w,max}$ **then**
8: $\quad\quad \mathcal{T}_{max} = \mathcal{T}_{i,max}$; $flag \leftarrow flag + 1$;
9: $\quad$ **else** $flag \leftarrow 0$;

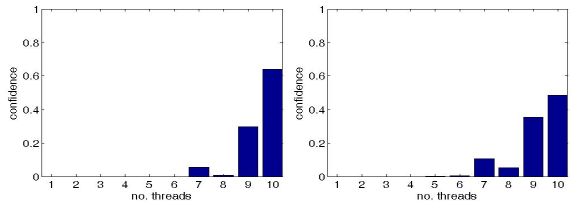
B. Performance Assessment

We evaluate the overall performance of the $d-w$ method in terms of *probing overhead* given by the percentage of a complete configuration sweep that is actually performed. We also evaluate *probing accuracy* given by the percentage of the maximum throughput that it returns. To demonstrate the joint effect of the two probing parameters, we study the performance of various parameter combinations.

The $d-w$ algorithm fails to return the peak throughput if the unimodality condition is not satisfied by the configurations it probed. The fraction of such combinations among those measured is an indication of the likelihood of such a failure. Thus, we estimate the *confidence* of the $d-w$ method by measuring the fraction of configurations that satisfy the unimodality property among the total configurations in our measurements. For XFS, Lustre with direct and default I/O, we estimate the confidences using 700 configurations (10 flows, 10 repetitions, and 7 RTTs) for each request size used in our measurements. The likelihood that the d–w method returns the peak throughput is correlated with high confidence estimates as will be described next,.

1) XFS Write: Fig. 13 shows the confidence estimates for different flow counts, for XFS at 22.6 ms RTT, for four (d, w) combinations. More probes for each configuration, i.e., larger d, correspond to a higher confidence, which also corresponds to peak configuration (10 flows) identified by the d–w method.

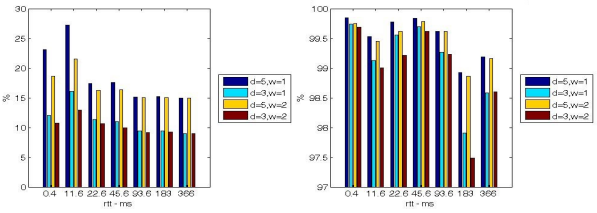
Fig. 14 shows the probing profile for XFS write transfers, including overhead and accuracy. From Fig. 14(b), the probing overhead for most cases is below 20%; that is, a total of less than 20 traces out of 100. The results indicate that d has a greater impact on overhead than w, due largely to the early termination rule for the probing process. The 11.6 ms-RTT case requires the most probing, because as seen in Fig. 8(c), its average throughput curve exhibits a zigzag pattern that extends to the maximum rates, preventing the prompt termination that would be triggered by two back-to-back decreases.



(a) $d = 5$, $w = 1$ (b) $d = 3$, $w = 1$

(c) $d = 5$, $w = 2$ (d) $d = 3$, $w = 2$

Fig. 13: Confidence estimates for XFS; RTT = 22.6 ms.



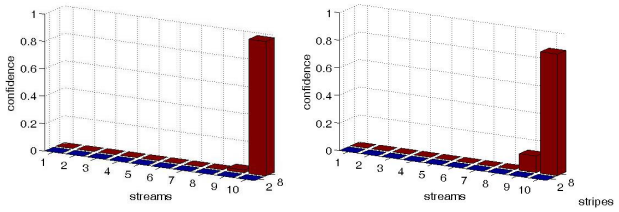
(a) Probing overhead (b) Probed vs. actual peak throughput

Fig. 14: $D-w$ performance for XFS

Fig. 14(c) shows probing accuracy, i.e., the percentage of average probed throughput relative to the actual maximum throughput from a full sweep. Most probes give throughput rates above 99% of the corresponding maximum; for a number of scenarios, notably 0.4 ms and 45.6 ms, they achieve over 99.5%. The relatively lower throughput performance for 183 ms RTT can be attributed to the larger variations among traces seen in Fig. 7(b). Note that for fixed w, increasing d leads to higher throughput, but increasing w alone does not always lead to consistent performance. This result demonstrates the trade-off between faster termination by avoiding probing adjacent configurations with close rates and the missed probing due to certain configurations being skipped.

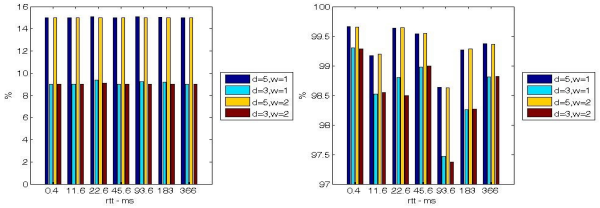
2) Direct I/O Lustre Write: Fig. 15 shows the confidence estimates for various configurations of Lustre with direct I/O, again for 22.6 ms RTT. As seen in Fig. 6(c), the peak rate is achieved with the highest number of flows (10), which also corresponds to high confidence (similar to XFS). We also note that the envelopes of peak throughput rates for both 2 and 8 stripes are better defined (Fig. 6(c)) than for XFS (Fig. 6(a)), which indicates a higher confidence for selecting the configuration with peak transfer rate (8 stripes, 10 flows).

From Fig. 16, we see that the probing overhead is $\leq$15%: even better than in the XFS write case. Meanwhile, the probed transfer rates are $\geq$98% of the actual maxima in most scenarios. We also see that the use of $w = 2$ vs. $w = 1$



(a) $d = 5$, $w = 1$ (b) $d = 3$, $w = 1$

Fig. 15: Confidence estimates for direct I/O Lustre; RTT 22.6 ms.



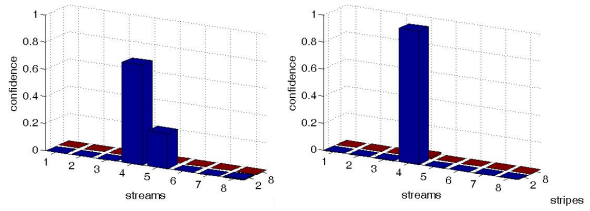
(a) Probing overhead (b) Probed vs. actual peak throughput

Fig. 16: $D-w$ performance for direct I/O Lustre

has little impact on performance, due to the well-behaved envelope of peak throughput rates across the configurations; in particular, the peak configuration is typically selected after two negative gradient computations.

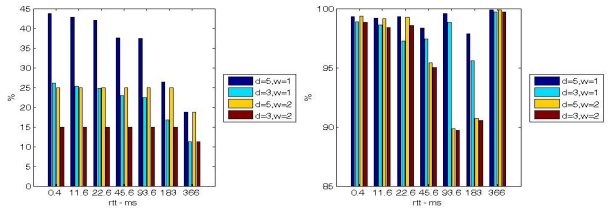
3) Default I/O Lustre Write: In certain workflows, host systems perform many more local I/O operations than long-haul transfers. In these cases, default I/O is a natural choice since it provides higher host transfer rates, even though it is not ideal for (fewer) long-haul transfers. We now consider such cases wherein long-haul transfers are handled by XDD. We see from Fig. 17 that the peak configuration (with 2 stripes and 4 flows) is selected with confidence of 0.7 with $d = 5$ and $w = 1$; but it becomes lower with $d = 3$ and $w = 1$. A close examination of Fig. 6(b) reveals that on rare occasions, the decreasing lineup from $8 \rightarrow 7 \rightarrow 6$ flows results in the sub-optimal configuration with 2 stripes and 8 flows being picked after probing. Because the top throughput at lower RTTs (4 flows) is located almost halfway from the starting point (8 flows), a larger overhead would be incurred, as can be seen from Fig. 18(b). With $w = 2$, the overhead is equivalent to that from a complete probe, since the termination rule will not be checked in time once the probing ends with 2 flows. With the exception of 93.6 ms and 183 ms with $w = 2$, most identified throughput rates are within 95% of peaks, which also correspond to higher confidences. The throughput ($\sim$90%) in the former cases is due mainly to the peak throughput occurring with 5 flows, a case that is not probed with $w = 2$, and the adjacent configurations yielding substantially lower transfer rates.

Our $d-w$ probing method can be applied to other scenarios that require parameter selections based on profiles, such as choosing the number of parallel TCP threads for memory transfers. It strikes a balance between the cost and performance compared to methods that build a full sweep profile



(a) $d = 5$, $w = 1$ (b) $d = 3$, $w = 2$

Fig. 17: Confidence estimates for default I/O Lustre; RTT 22.6 ms



(a) Probing overhead (b) Probed vs. actual peak throughput

Fig. 18: $D-w$ performance for default I/O Lustre

by repeating mesurements at each configuration a certain minimum number of times. The results demonstrate that the $d-w$ method can generate the relevant portions of the transfer rate profiles with a small number of repetitions, while finding configurations with close to peak transfer rates.

VI. Conclusions

We report on extensive disk-to-disk file transfer measurements for Lustre and XFS filesystems over emulated dedicated connections for a wide range of RTTs. Our results provide valuable insights into factors that effect transfer rates. In particular, they reveal surprising interactions between disk and network systems that necessitate careful configuration to achieve good performance. We also find that large variations require repeated measurements to ensure estimation confidence. Our gradient-descent based $d-w$ method avoids the need for repeated time-consuming sweeps of all parameter combinations by probing a small number of measurements to identify configurations that achieve peak rates.

Future directions include detailed analytical development of the $d-w$ method and its variants in terms of performance guarantees. It will be of interest to develop methods that dynamically adapt d and w values based on measurements. Other considerations beyond peak throughput, such as resource utilization, can be incorporated into this framework by using the corresponding measurements. It will also be interesting to study other probing methods that are not based on the gradient descent approach.

Acknowledgments

This work is supported in part by the United States Department of Defense, using resources of the Computational Research and Development Programs, and the Net2013 and RAMSES projects, Office of Advanced Computing Research, Department of Energy, under Contracts DE-AC05-00OR22725 and DE-AC02-06CH11357.

References

[1] On-demand secure circuits and advance reservation system. http://www.es.net/oscars.

[2] W. Allcock, I. Foster, S. Tuecke, A. Chervenak, and C. Kesselman. Protocols and services for distributed data-intensive science. In *AIP Conference Proceedings*, pages 161–163. Institute of Physics, 2000.

[3] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, et al. Software as a service for data scientists. *Communications of the ACM*, 55(2):81–88, 2012.

[4] B. Behzad, S. Byna, M. Snir, et al. Pattern-driven parallel I/O tuning. In *10th Parallel Data Storage Workshop*, pages 43–48. ACM, 2015.

[5] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, pages 177–186. Springer, 2010.

[6] P. H Carns, B. W. Settlemyer, and W. B. Ligon III. Using server-to-server communication in parallel file systems to simplify consistency and improve performance. In *ACM/IEEE Conference on Supercomputing*, page 6. IEEE Press, 2008.

[7] Science DMZ: Data Transfer Nodes, https://fasterdata.es.net/science-dmz/DTN.

[8] Energy Sciences Network. http://www.es.net.

[9] W. Feng, M. Gardner, M. Fisk, and E. Weigle. Automatic flow-control adaptation for enhancing network performance in computational grids. *Journal of Grid Gomputing*, 1(1):63–74, 2003.

[10] J. Gao, N. S. V. Rao, J. Hu, and J. Ai. Quasi-periodic route to chaos in the dynamics of internet transport protocols. *Physical Review Letters*, 2005.

[11] M. Gardner, S. Thulasidasan, and W. Feng. User-space auto-tuning for TCP flow control in computational grids. *Computer Communications*, 27(14), 2004. Special Issue on Network Support for Grid Computing.

[12] T. Ito, H. Ohsaki, and M. Imase. On parameter tuning of data transfer protocol GridFTP for wide-area grid computing. In *2nd International Conference on Broadband Networks*, pages 1338–1344. IEEE, 2005.

[13] T. Ito, H. Ohsaki, and M. Imase. Automatic parameter configuration mechanism for data transfer protocol GridFTP. In *International Symposium on Applications and the Internet*. IEEE, 2006.

[14] T. Ito, H. Ohsaki, and M. Imase. GridFTP-APT: Automatic parallelism tuning mechanism for data transfer protocol GridFTP. In *6th IEEE International Symposium on Cluster Computing and the Grid*, 2006.

[15] T. Kelly. Scalable TCP: Improving performance in high speed wide area networks. *Computer Communication Review*, 33(2):83–91, 2003.

[16] H. J. Kushner and C. G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003. Second Edition.

[17] N. Liu, C. Carothers, J. Cope, P. Carns, R. Ross, A. Crume, and C. Maltzahn. Modeling a leadership-scale storage system. In *Parallel Processing and Applied Mathematics*, pages 10–19. Springer, 2011.

[18] Lustre Basics, https://www.olcf.ornl.gov/kb_articles/lustre-basics.

[19] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[20] N. S. V. Rao, G. Hinkel, N. Imam, and B. W. Settlemyer. Measurements of file transfer rates over dedicated long-haul connections. In *2nd International Workshop on The Lustre Ecosystem*, 2016.

[21] I. Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *3rd International Workshop on Protocols for Fast Long-Distance Networks*, 2005.

[22] B. W. Settlemyer, J. D. Dobson, S. W. Hodson, J. A. Kuehn, S. W. Poole, and T. M. Ruwart. A technique for moving large data sets over high-performance long distance networks. In *IEEE 27th Symposium on Mass Storage Systems and Technologies*, pages 1–6, May 2011.

[23] B. W. Settlemyer, N. S. V. Rao, S. W. Poole, S. W. Hodson, S. E. Hicks, and P. M. Newman. Experimental analysis of 10Gbps transfers over physical and emulated dedicated connections. In *International Conference on Computing, Networking and Communications*, 2012.

[24] R.N. Shorten and D.J. Leith. H-TCP: TCP for high-speed and long-distance networks. In *3rd International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[25] J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley Pub, 2003.

[26] Linux tuning, https://fasterdata.es.net/host-tuning/linux.

[27] B. J. Winer. Latin squares and related designs. In *Statistical Principles in Experimental Design*. McGraw-Hill Book Company, 1962.

[28] XDD - The eXtreme dd toolset, https://github.com/bws/xdd.