J. Parallel Distrib. Comput. 120 (2018) 89–100

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Big data transfer optimization through adaptive parameter tuning

Engin Arslan [a,*], Bahadir A. Pehlivan [a], Tevfik Kosar [b]

[a] University of Nevada, Reno, 1664 N Virginia St, Reno, NV 89557, United States
[b] University at Buffalo, SUNY, 338 Davis Hall, Buffalo, NY 14260, United States

HIGHLIGHTS

- Application-level transfer parameters can improve transfer throughput significantly.
- Dataset partitioning considerably affects data transfer throughput for mixed datasets.
- Adaptive protocol tuning is necessary to adjust transfer parameters to network specific conditions.

ARTICLE INFO

Article history:
Received 13 September 2017
Received in revised form 30 April 2018
Accepted 4 May 2018

Keywords:
Application-level protocol tuning
Throughput optimization
Wide-area data transfers

ABSTRACT

Obtaining optimal data transfer performance is of utmost importance to today's data-intensive distributed applications and wide-area data replication services. Tuning application-layer protocol parameters such as pipelining, parallelism, and concurrency can significantly increase efficient utilization of the available network bandwidth as well as the end-to-end data transfer performance. However, determining the best settings for these parameters is a challenging problem, as network conditions can vary greatly between sites and over time. Poor protocol tuning can cause either under- or over-utilization of network resources and thus degrade transfer performance. In this paper, we present three novel algorithms for application-layer parameter tuning and transfer scheduling to maximize transfer throughput in wide-area networks. Our algorithms use heuristics to tune the level of control channel pipelining (for small file optimization), the number of parallel data streams per file (for large file optimization), and the number of concurrent file transfers to increase I/O throughput (for all types of files). The proposed algorithms improve the transfer throughput up to 10x compared to the baseline and 7x compared to the state-of-the-art solutions. We also propose adaptive tuning to adjust the values of parameters based on real-time observations. The results show that adaptive tuning can further improve transfer throughput by up to 24% compared to the heuristic approach.

1. Introduction

Despite the increasing availability of high-speed wide-area networks and the use of modern data transfer protocols designed for high performance, file transfers in practice attain only a fraction of theoretical maximum throughput, leaving networks underutilized and users unsatisfied. This fact is due to a number of confounding factors, such as under-utilization of end-system CPU cores, low disk I/O speeds, server implementations not taking advantage of parallel I/O opportunities, background traffic at inter-system routing nodes, and unsuitable system-level tuning of networking protocols.

The effects of some of these factors can be mitigated to varying degrees through the use of techniques such as control-channel pipelining, transport-layer parallelism, and concurrent transfers using multiple data channels. **Pipelining** targets the problem of transferring a large numbers of small files [10,12,19]. In most control channel-based transfer protocols, transfer of a file must complete and be acknowledged before the next file can be requested. This may cause a delay of more than one RTT between individual file transfers especially in cases where the client is located far from source and destination (i.e., third-party transfers). With pipelining, multiple transfer commands can be queued up at the server, reducing the delay between transfer completion and receipt of the next command from the client. **Parallelism** sends different portions of the same file over parallel data streams (typically TCP connections), and can achieve high throughput by aggregating multiple streams [15,35,41]. **Concurrency** refers to transferring multiple files simultaneously using different control and data channels and is especially useful for increasing I/O concurrency in parallel disk systems [21,27,39]. The degree to which these techniques are utilized, however, has the potential to negatively

* Corresponding author.
E-mail address: earslan@unr.edu (E. Arslan).

https://doi.org/10.1016/j.jpdc.2018.05.003
0743-7315/

impact the performance of the transfer and the network as a whole. Too little use of one technique, and the network might be underutilized; too much, and the network might be overburdened to the detriment of the transfer and other users. Furthermore, the optimal level of usage for each technique varies depending on the network and end-system conditions, meaning no single parameter combination is optimal for all different scenarios.

We propose dynamic transfer optimization algorithms to determine which combination of parameters is "just right" for a given transfer task. Main contributions of this paper are: (i) optimization of dataset clustering for heterogeneous datasets that include both small and large files; (ii) a heuristic approach to estimate parameter values to be used in transfer; (iii) three novel transfer scheduling algorithms to improve overall transfer throughput; and (iv) an adaptive parameter tuning algorithm to decide optimal values in the run-time. We have run extensive experiments using real and synthetic datasets in both wide- and local-area networks. The experimental results are very promising, and our algorithms outperform other existing solutions in this area.

The remainder of this paper is organized as follows. The next section presents the related work. Section 3 introduces protocol tuning and scheduling algorithms we propose. Section 4 presents the performance evaluation results of our algorithms. Section 5 concludes the paper with our discussion and future directions.

2. Related work

Several solutions are proposed to improve utilization of a single path by means of parallel streams [4,14,29,38], pipelining [8,10,12], and concurrent transfers [24,26,27,36]. Several file transfer tools tried to statically tune a subset of these parameters in an effort to improve the end-to-end data transfer throughput [20,22,23,25,40]. Liu et al. [27] developed a tool to determine the optimal number of concurrent file transfers based on observed transfer throughput. While the presented performance increase is significant, setting the number of concurrent transfers just by considering transfer throughput would lead to opening too many processes and connections hence overloads the end system and network. Thus, both transfer throughput and system overhead needs to be considered when tuning concurrency. We propose a dynamic tuning algorithm for concurrency in Section 4.3 using a cost function which rewards high throughput and low concurrency values.

We developed three highly-accurate models [18,42,43] which would require as few as three sampling points to provide accurate predictions for the optimal parallel stream number. These models have proved to be more accurate than existing similar models [16,29] which lack in predicting the parallel stream number that gives the peak throughput. We also have developed algorithms to determine the best sampling size and the best sampling points for data transfers by using bandwidth, Round-Trip Time (RTT), or Bandwidth-Delay Product (BDP) [37].

In addition, several approaches are proposed to tune multiple transfer parameters at the same time using heuristics [3,7], offline modeling [5,6,30], and adaptive [34] techniques. Globus Online [3] sets pipelining, parallelism, and concurrency parameters to predetermined values for three different file sizes (i.e., less than 50MB, larger than 250MB, and in between). However, the protocol tuning Globus Online performs is non-adaptive; it does not change depending on dataset and network settings and does not perform well in various scenarios. In our earlier work [7], we proposed a heuristic approach that relies on dataset and network settings (shown in Algorithm 1). We extend [7] by (i) investigating the impact of dataset partitioning on transfer throughput; (ii) running extensive experiments in different networks using real-word datasets; and (iii) improving the heuristic solution by enhancing with adaptive tuning to adjust the value of concurrency in real-time.

Modeling based approaches use historical data to derive models to define a relationship between transfer parameters and throughput. HARP [5,6] derives polynomial models to relate application-layer transfer parameter to transfer throughput and solves these models for maximum throughput to find the corresponding values for transfer parameters. While it can obtain close-to-optimal throughput results, it requires historical data to contain up-to-date information for different network conditions (such as background traffic) which hinders its deployment.

PCP [34] tries to identify optimal values of transfer parameters in the run-time similar to our adaptive tuning approach. However, there are several key differences between PCP and this work including but not limited to: (i) PCP partitions dataset into a fixed number of clusters (i.e., five), whereas we analyze impact of creating different number of clusters in this work and show that using only two clusters is sufficient to benefit from dataset partitioning; (ii) PCP transfers each file cluster separately whereas our MC and ProMC algorithms transfer multiple clusters simultaneously to mitigate the effect of small file transfers over average throughput; and (iii) PCP's adaptive search algorithm is rather simple compared to our multi-phase quick search algorithm due to which it spends more time in search phase thus resulting in poor overall transfer throughput.

Other approaches aim to improve throughput by opening flows over multiple paths between end-systems [17,32], however there are cases where individual data flows fail to achieve optimal throughput because of end-system bottlenecks.

3. Dynamic protocol tuning algorithms

Pipelining, parallelism, and concurrency play a significant role in affecting achievable transfer throughput. However, setting the optimal levels for these parameters is a challenging problem, and poorly-tuned parameters can either cause underutilization of the network or overburden the network and degrade the performance due to increased packet loss, end-system overhead, and other factors. These transfer parameters can be set to any integer values, however system administrators may define upper limits as too large values may cause system to crash. If not set, the default values (0,1,1) are used for pipelining, parallelism, and concurrency, respectively. Pipelining value 0 means that no outstanding command will be stored at the end servers. Parallelism value 1 means that only one data channel will be used transfer each file. Finally, concurrency value 1 will enforce to transfer one file at any given time.

3.1. Impact of pipelining, parallelism and concurrency on transfer throughput

To analyze the effects of pipelining, parallelism and concurrency on the transfer of different file sizes, we initially conducted experiments for each of the parameters separately, as shown in Figs. 1 and 2. We run our experiments on XSEDE [33] and LONI [28] production-level high-bandwidth networks whose specifications are given in Table 1. Although both of the networks have 10 Gbps network bandwidth between sites, XSEDE provides higher throughput in end-to-end (disk-to-disk) transfers despite high RTT between its sites. This is mainly due to better storage I/O performances at the XSEDE sites.

We generated five datasets for five different file sizes and transferred each dataset, only changing one parameter (i.e., pipelining, parallelism or concurrency) at a time to observe the individual effect of each parameter. Then, we introduced other parameters one by one. Figs. 1(a) and 2(a) show that pipelining can increase