# Online Optimization of File Transfers in High-Speed Networks

Md Arifuzzaman
University of Nevada, Reno
Reno, Nevada, USA
arif@nevada.unr.edu

Engin Arslan
University of Nevada, Reno
Reno, Nevada, USA
earslan@unr.edu

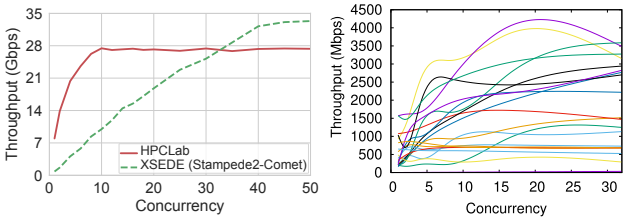
## ABSTRACT

File transfers in high-speed networks require network and file system parallelism to increase resource utilization and reach high speeds. However, creating arbitrarily large numbers of I/O threads and network connections overwhelms system resources and causes fairness issues. In this paper, we introduce `Falcon` that combines a novel utility function with state-of-the-art online optimization algorithms to discover the optimal transfer settings that achieves high performance, while keeping system overhead low and ensuring fair resource sharing between competing transfers. Our extensive evaluations in several dedicated and production high-speed networks show that `Falcon` can find near-optimal solution in as little as 20 seconds and outperforms existing transfer application by 2× to 6×. Furthermore, unlike existing transfer optimization solutions that result in unfair resource allocation when multiple transfers run concurrently, `Falcon` is guaranteed to converge to Nash Equilibrium with the help of its game theory-inspired utility function.

**ACM Reference Format:**
Md Arifuzzaman and Engin Arslan. 2021. Online Optimization of File Transfers in High-Speed Networks. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458817.3476208

## 1 INTRODUCTION

Rapid evolution of instrument technologies along with growing storage and compute capacities have led to unprecedented increase in amount of data generated by scientific applications. For example, advancements in high-throughput genome sequencing technology increased output size per single run from around 5 MB in 2006 to more than 5 GB in 2018, a three order of magnitude increase in 12 years. As science projects are increasingly *distributed* and *collaborative*, growing data sizes demands high-speed data transfers to move data between geographically separated institutions in a timely manner. Yet, majority of data transfers fall short to reach beyond few gigabit-per-second throughput despite the availability of high-speed networks with up to 100 Gbps bandwidth, dedicated data transfer nodes, and high-throughput parallel file systems, mainly due to lack of scalable data transfer applications. Legacy file transfer applications (e.g., FTP, scp) are initially designed for low-speed internet transfers (i.e., in the order of megabytes per second), thus they fail to perform well for large-scale scientific transfers in high-speed networks. Specifically, network and I/O parallelism is essential to reach beyond few gigabit-per-second transfer throughput in high-speed networks since neither single network connection nor single I/O thread is sufficient to reach higher speeds.




(a) Impact of Concurrency on Throughput (b) Optimal Concurrency

**Figure 1: Although transferring multiple files concurrently helps to improve transfer throughput significantly, the optimal value of concurrency depends on many static and dynamic factors, thus necessitates an adaptive solution.**

Single TCP stream is typically limited to 30 Gbps throughput due to memory and CPU limitations. Similarly, single file read/write speed is in less than 10 Gbps with hard drives and less than 30 Gbps with solid state drives even with disk striping, such as RAID. Yet, these numbers are much lower in productions systems due to resource interference. To validate this, we transferred a $500 \times 1GiB$ files between two isolated servers in same local area network that are configured with RAID array (with 4 NVMe SSDs) storage system and connected through 40 Gbps link. Figure 1(a) shows that transferring one file-at-a-time (i.e., *concurrency* = 1) obtains less than 8 Gbps throughput in one network (i.e., HPCLab) and less than 2 Gbps throughput in another networks (i.e., XSEDE) mainly due to write I/O limitations. However, transferring multiple files simultaneously increases throughput to around 30 Gbps in both networks, $3 - 15x$ increase compared to baseline configuration. On the other hand, finding the optimal concurrency value is a challenging task due to large search space and prohibitive cost of exhaustive profiling. Deriving accurate analytical models for production systems is also nearly impossible due to challenges associated with collecting real-time performance metrics from all components of end-to-end transfers that span across multiple clusters and network domains that are managed by separate entities. Specifically, HPC clusters and research network providers typically do not share performance metrics for system resources such as load on storage servers and queue size of network devices in the real-time which would play significant role on the observed transfer performance.

Figure 1(b) illustrates the impact of concurrency on transfer throughput when it is used to transfer various types of datasets (e.g.,

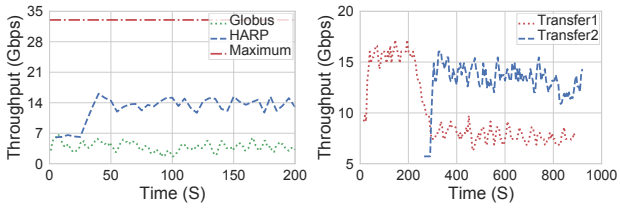

*SC '21, November 14–19, 2021, St. Louis, MO, USA*

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476208

lots of small files and single very large file) in different networks settings (e.g., dedicated local-area network and shared wide-area network). Evidently, the optimal concurrency level is not same for all transfers as it depends on many factors. Even more challenging is that the optimal solution can be different for identical transfers (i.e., transfer of the same dataset between the same end points) over time due to change in background traffic. While high concurrency values typically yield high performance, using arbitrarily large values can lead decrease in throughput in addition causing congestion (both at the network and file system level) and unfair resource allocation between competing transfers [7]. As a result, while concurrent file transfers is essential to increase resource utilization and achieve high transfer performance, *the optimal concurrency level depends on many static (e.g., network and file systems settings) and dynamic (e.g., network congestion) factors, making it challenging to estimate.* Previous work in this are involve heuristic [9, 12], supervised learning [11, 29, 39], and real-time optimization [47, 48] models. Despite yielding higher throughput than baseline settings, heuristic models fail to offer robust performance in all networks as they fail to incorporate dynamic conditions into their models. Supervised learning models can make precise predictions, however, deriving an accurate and flexible model requires large amount of historical data to be collected in wide range of transfer conditions (e.g., dataset, background traffic, etc.), which could take weeks or months. Real-time optimization algorithms can discover the optimal settings in the runtime, however, existing solutions in this area suffer from long convergence time and failure to provide fairness and stability guarantees in shared environments.

In this paper, we introduce `Falcon` which combines game theory-inspired utility function with the state-of-the-art online search algorithms to discover optimal concurrency level swiftly while ensuring fairness among competing transfers. As opposed to previous high-speed transfer optimization solutions which solely focus yielding higher transfer throughput, `Falcon` innovates a novel utility function to discover "just-enough" concurrency that can obtain near-optimal transfer performance while lowering system overhead and ensuring fair resource allocation. Our extensive evaluations in various network settings with up to 40 Gbps bandwidth show that `Falcon`– with the help of fast and efficient online optimization models–achieves $2-6\times$ higher throughput compared to the state-of-the-art file transfer optimizations solutions. *More importantly,* `Falcon` *is the first algorithm that guarantees fairness between competing transfers by incorporating regret terms into its utility function.* In summary, we make following unique contributions to the field:

- We introduce `Falcon` which innovates a novel game theory-inspired utility function to evaluate the performance of transfer settings which rewards high throughput and penalizes increased system overhead.
- We implement three online optimization algorithms (i.e., Hill Climbing, Gradient Descent, and Bayesian Optimization) for `Falcon` to swiftly sweep large search space in real-time. The results show that Gradient Descent and Bayesian Optimization can discover optimal transfer settings in as little as 20 seconds and yield close-to-maximum transfer throughput in all network settings while ensuring fair resource allocation between competing transfers.




(a) Single Transfer Performance (b) Competing Transfers (HARP)

**Figure 2: State-of-the-art transfer optimization solutions, Globus [9] and HARP [11] are unable to achieve high-performance due to lack of adaptive parallelism (a). They also fail to guarantee fairness between competing transfers (b).**

- We evaluate the performance of `Falcon` in several shared and production high-speed networks with different bottleneck scenarios to validate its effectiveness in wide range of conditions.

## 2 MOTIVATION

Although file transfer optimization has been studied extensively in the past, we identify two major issues with existing solutions as *failure to guarantee high performance* in all networks and *unfair resource sharing* when multiple transfers compete. We next discuss that these two issues cannot be overcome through simple extensions of existing solutions due to potential side-effects; *increased overhead on system resources* in particular.

**Poor Transfer Performance:** Despite the availability of high-speed networks and high-capacity parallel file systems, existing file transfer applications and services fall short to take full benefit of these resources due to lack of adaptive resource parallelism. Figure 2(a) demonstrate the performance of two state-of-the-art file transfer optimization solutions for a transfer between Comet and Stampede2 supercomputer that are connected with 40 Gbps network bandwidth. Globus [9] uses fixed and mostly suboptimal transfer settings hence obtains less than 6 Gbps throughput. On the other hand, despite attaining higher throughput than Globus, HARP [11] also underperforms by obtaining only around 50% of maximum throughput. This is mainly because of the fact that its lacks historical data in this network, so makes predictions by analyzing transfer logs gathered in 10 Gbps networks. While collecting new data to re-train HARP for this network will improve its performance, it can take weeks to months to collect sufficient amount of data, which is not feasible to perform in all networks.

**Unfair Resource Sharing:** In the presence of multiple independent file transfers competing for limited available resources, convergence to a fair stable state is desired behavior. This well-known concept in game theory which requires competing agents either use same fixed strategy or periodically update their strategy using a symmetric, strictly concave utility function [45]. Thus, transfer optimization algorithms that employ fixed strategy (e.g., Globus [3]) will converge to a stable state, however they will fail to adapt to changing conditions since the optimal strategy depends on dynamic conditions such as the number of competing transfers. On the other hand, solutions that tune transfer settings only once at


**Figure 3: Network topology used for single transfer experiment in Emulab.**

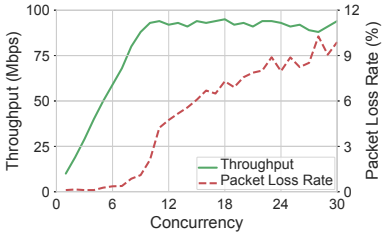



**Figure 4: While running concurrent transfers is required to improve file transfer throughput, aggressive use of it would lead to congestion in the network .**

the beginning of the transfer (e.g., HARP [10]) fails to provide fair resource sharing between the competing transfers as "*late-comers*" will have an unfair advantage by choosing a setting that favors them with high transfer throughput. Yet, extending existing transfer solutions to run the optimization process periodically to adapt changing conditions will not work either since their throughput-oriented utility models (i.e., higher transfer throughput yields higher utility) do not meet strictly concave utility requirement of fair convergence. As an example, the utility, $u$, of a transfer task creates $n$ concurrent transfers (i.e., $concurrency = 4$) each obtaining $t$ throughput[1] can be calculated by

$$u(n,t) = \sum_{i=1}^{n} t = n \times t \tag{1}$$

when utility is set to be linearly proportional to transfer throughput. Fair resource sharing requires the second derivation of utility functions to be negative, but it returns 0 for Equation 1, thus it cannot guarantee fairness between competing transfers. Figure 2(b) illustrates this behavior for HARP which utilizes historical analysis to find a transfer setting that maximize transfer throughput [11]. It is clear that when the second HARP transfer joins, it chooses a setting that favors to itself and yields nearly 2× of the throughput of the first transfer.

**Overburdened Network and End Hosts:** A naive solution to maximize transfer throughput in high-speed networks while ensuring fairness can be implemented by using fixed transfer setting that involves high values for concurrency parameter, such as 30. However, high levels of concurrency can overwhelm end system and network resources by creating too many processes and network connections. To demonstrate this, we evaluate the performance of a file transfer when concurrency is set to values between 1 and 32 in a simple network topology as depicted in Figure 3. While hardware limits for disk read/write speed 1 Gbps, we throttle read I/O throughput of a single process to 10 Mbps to emulate the behavior of parallel file systems in which concurrent I/O yields higher

[1]Since each transfer thread uses the same congestion control algorithm to transfer similar size files between the same end points, they will attain similar throughput as most commonly used TCP variants (e.g., Reno, Cubic, HTCP, and BBR) guarantees fairness among competing flows with the same RTT [17, 31]

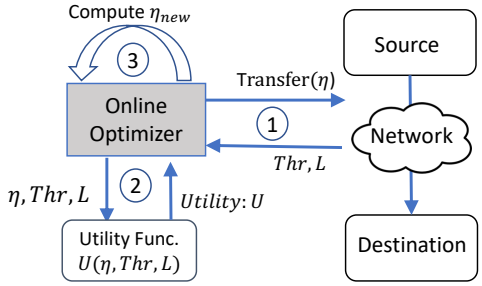



**Figure 5: `Falcon` uses online optimization to discover optimal transfer settings quickly. The utility function rewards high throughput while penalizing increased system overhead to ensure convergence to fair and optimal solution.**

throughput compared to single threaded I/O. Since network bandwidth is 100 Mbps, 10 concurrent transfers is needed to achieve 100 Mbps aggregate I/O throughput, thereby reaching to maximum transfer speed. Although more than ten concurrent transfers also yields 100 Mbps throughput, it results in significant increase in packet loss due to network congestion at the bottleneck link as shown in Figure 4. Specifically, while packet loss is below 2% when concurrency is smaller than 10, it increases drastically and reaches to 10% for concurrency value of 32. In addition increased packet loss, high concurrency values also overburden end hosts and storage systems due processing overhead of concurrent processes/threads [7]. *Consequently, there is a need for high-speed file transfer optimization solution that can tune the level of transfer parallelism in real-time to provide* ***high-performance and fair bandwidth sharing*** *while keeping system overhead at minimum.* We therefore introduce `Falcon` that combines a game-theory inspired utility function with state-of-the-art online optimization algorithms to address these limitations.

## 3 FALCON: ONLINE HIGH-SPEED FILE TRANSFER OPTIMIZATION ALGORITHM

`Falcon` implements online learning to find optimal transfer settings in the runtime as illustrated in Figure 5. It adopts a black-box approach and uses sample transfers to evaluate different transfer settings. The benefit of this black-box approach is the generality and applicability to a diverse set of network systems without making any assumption about underlying infrastructure. Such abstraction makes it possible to develop intuitive understanding of the system conditions through simple performance metrics such as throughput and packet loss rate. `Falcon` first selects a transfer setting, $\eta$ and runs a sample transfer to evaluate its performance. Once the sample transfer is executed for a sufficient amount of time, it captures performance metrics and uses a utility function ,$U$, to convert a set of performance metrics to a scalar value that quantifies the effectiveness of the transfer setting $\eta$. Finally, it processes the calculated utility value using online optimization algorithms (e.g., Bayesian optimization, online gradient descent) to predict a new transfer setting $\eta_{new}$ closer to the optimal. Compared to existing optimization algorithms for high-speed file transfers, `Falcon` makes two novel and key contributions. First, it innovates a *game-theory inspired utility function* that incorporates regret (i.e., penalty) term for increased packet loss and concurrent transfer count to keep system

overhead at minimum and ensure **fairness among competing transfers**. Second, it implements three state-of-the-art *online optimization algorithms*, Hill Climbing, Gradient Descent, and Bayesian Optimization, to scan search space and **converge to the optimal expeditiously**. We first apply `Falcon` to tune the number of concurrent transfers (i.e., $\eta = \{concurrency\}$) as it has been shown to be the most effective parameter in the optimization of large-scale file transfers [11, 33, 47]. In § 4.4, we demonstrate that `Falcon` can be extended to tune additional transfer parameters to offer more precise solutions for long-running bulk data transfers. Note that `Falcon` operates in the application layer, so it does not rely on any specific transport protocols and can easily be extended to support any transfer application including FTP, GridFTP, and bbcp. In fact, multiparameter optimization of `Falcon` is implemented in GridFTP as it gives a flexibility to tune several transfer settings and readily available in most production high-performance computing clusters.

## 3.1 Utility Function

Utility functions need to involve a regret term to converge to a fair and optimal solution [23, 45, 51, 52], thus `Falcon` incorporates packet loss into its utility function as

$$u(n_i, t_i, L_i) = n_i t_i - n_i t_i L_i \times B \tag{2}$$

where $n_i$ is the number of concurrent file transfers, $t_i$ is average throughput of each transfer, and $L_i$ is packet loss rate. $B$ is a constant coefficient that is used to determine the severity of punishment for increased packet loss. While the value of $B$ can be customized for specific application scenarios, we find that $B = 10$ works well with most commonly used TCP variants (i.e., TCP Cubic and Reno, and HSTCP) by keeping packet loss rate below 1% while achieving over 95% network utilization. As a result, the utility function in the form of Equation 2 addresses overhead and fairness issues when suboptimal concurrency values cause increased packet loss. However, as high-performance networks with up to 40/100 Gbps capacity are being built, transfer bottlenecks are shifting toward end hosts. For example, the network service provider of most research and education institutions in the U.S., Internet2, supports 100 Gbps connectivity to most sites and is upgrading its backbone capacity to 400 Gbps [4]. On the other hand, it is challenging, if not impossible, to attain 100 Gbps I/O throughput in production clusters due to inevitable resource interference. Moreover, most HPC clusters use data transfer nodes with 10/40 Gbps Network Interface Cards (NICs), limiting maximum possible transfer rate to smaller values compared to network bandwidth. Consequently, little to no packet loss is observed in many production systems, necessitating an additional penalty term to limit the excessive use of concurrency. We therefore propose a cost function that penalizes the use of high concurrency by incorporating the value of concurrency into the utility function as

$$u(n_i, t_i, L_i) = n_i t_i - n_i t_i L_i \times B - n_i t_i \times n_i C \tag{3}$$

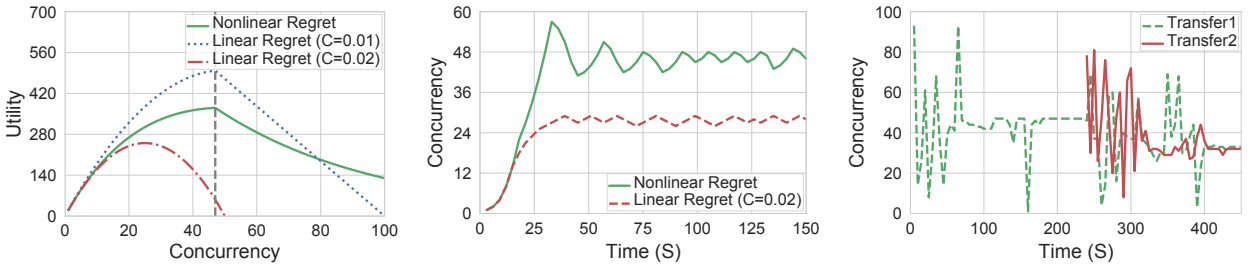
where $C$ is a constant coefficient that is used to adjust the rate of penalty for increased concurrency. Previous studies show that the utility functions that incorporate monotonically increasing penalty terms in linear form guarantees high performance for single transfer and optimal and fair convergence for competing transfers (i.e., Nash Equilibrium) [23, 52] . However, we find that it is challenging to achieve both high-performance and fair and optimal convergence when concurrency regret is incorporated in linear forms similar to Equation 3. Figure 6 presents **estimated** utility value when $C$ is set to 0.01 (nearly 1% punishment for each concurrency) and 0.02 (2% punishment) when the optimal concurrency level is 48; i.e., 48 concurrent transfers are needed to reach full I/O and network utilization. When $C$ is set to 0.02, the utility value peaks at concurrency value 25 which in turn results in lower than maximum throughput. We experimentally validate this behavior by configuring I/O limitation per process (as described in § 2) to require 48 concurrent I/O threads to reach maximum possible throughput in Figure 6(b). The transfer converges to suboptimal concurrency value of 26 hence obtains 45% lower transfer throughput than the optimal. Smaller $C$ values such as 0.01 are able to converge to optimal configurations for single transfer scenario both theoretically (Figure 6(a)) and empirically(Figure 6(c)), but results in suboptimal convergence behavior when there are multiple competing transfers due to increased sensitivity to measurement jitters. Figure 6(c) illustrates that although the utility function with linear penalty of 1% (i.e.,$C = 0.01$) converges to the optimal solution when there is only one transfer in the system, it fails to do so when the second one joins. The optimal solution requires both transfers to create 24 concurrent transfers to yield maximum throughput with minimal overhead, they converge to $36 - 38$ concurrent transfers which overburdens system resources unnecessarily .

To address these issues, we test a nonlinear form of regret for concurrency as

$$u(n_i, t_i, L_i) = \frac{n_i t_i}{K^{n_i}} - n_i t_i L_i \times B \tag{4}$$

where $K$ is constant. As throughput improvement ratio is not directly proportional to increased concurrency (i.e., the ratio of gain starts to lower at higher concurrency values), the value of $K$ can be tuned to require small but non-negligible gain (e.g., 1%) for increasing concurrency values. By doing so, we ensure that the utility will increase as long as non-negligible amount of throughput gain is observed and decrease upon exceeding the optimal concurrency value. Figure 6(a) and 6(b) show that the utility function that incorporates penalty for concurrency in a nonlinear form converges to the optimal both theoretically and empirically for single transfer optimization. It also converges to a fair and optimal solution when multiple transfers compete as presented in § 4.2.

In the presence of multiple computing transfer tasks, each Falcon agent will enter a regret minimization dynamics to lower packet loss and concurrency punishment. Intuitively, packet loss can only stay the same or increase as the number of concurrent transfers is increased. Thus, the term $1 - L_i \times B$ will follow monotonically decreasing pattern for the increasing number of concurrent transfers. Thus, the utility function given in Equation 4 is guaranteed to be concave as long as $\frac{n_i t_i}{K^{n_i}}$ is concave. It is also true when transfers are sender-limited (i.e., transfer bottleneck is I/O or NIC) since packet loss rate, $L_i$, will be return 0. Therefore, if the second derivative of $\frac{n_i t_i}{K^{n_i}}$ is negative (i.e., first derivative is strictly decreasing), then the utility function in the form of Equation 4 is guaranteed to be strictly concave, a condition that needs to be satisfied to converge to fair and optimal state.




(a) Linear vs nonlinear regret for concurrency (b) Suboptimal performance of linear regret 2% (c) Suboptimal convergence for linear regret 1%

**Figure 6: Comparison of linear (Eq 3) and nonlinear (Eq 4) forms of regret for concurrency in the utility function. Linear form of regret either fails to yield high performance for single transfer ($C = 0.02$ in (a) and (b)) or causes suboptimal convergence when multiple agents compete (c). Thus, `Falcon` incorporates concurrency penalty into its utility function in a nonlinear form.**

*Proof:* Let's denote $\frac{n_i t_i}{K^{n_i}} = f(n)$, then second derivative of $f(n)$ becomes

$$f''(n) = t_i K^{-n_i} \ln K(-2 + n_i \ln K) \quad (5)$$

Since, $t_i$, $n_i$, and $K$ are non-negative values, $t_i K^{-n_i} \ln K$ will return greater than 0. Thus, $f''(n)$ can only be negative if $(-2 + n_i \ln K$ is negative. Consequently, Equation 4 is guaranteed to be strictly concave as long as $n_i < \frac{2}{\ln K}$. Hence, the value of $K$ defines the upper limit for the number of concurrent transfers, $n$, that can be created before $f''(n)$ moves out of strictly concave region. Setting $K$ to 1.01 will expect at least 1% increase in throughput when comparing two consecutive concurrency values to prefer the larger one and require the optimal concurrency level to be less than or equal to 200 to guarantee Nash Equilibrium. Our experimental analysis show that while lower the value of $K$ helps to increase the upper limit for concurrency value, it causes stability issues in the case of competing transfers due to increased sensitivity to throughput fluctuations. We therefore set $K = 1.02$ (i.e., at least 2% throughput gain required for each new concurrent transfer) to strike a balance between stability and reduced upper limit. Please note that larger $K$ values, such as 1.10, will further improve resiliency against measurement noises, however it will cause transfers to converge to suboptimal results when the optimal concurrency is high since the optimal solution falls out of concave region.

## 3.2 Online Search Algorithm

A naive approach to discover the optimal solution can be implemented by evaluating the performance of all possible transfer settings (i.e., brute-force method), but is not feasible due to large search space and expensive nature of evaluating different settings. Specifically, it takes several seconds to accurately measure the performance of a transfer configuration due to connection establishment cost[2] as well as slow convergence of TCP transfers in long fat networks. We therefore implemented Hill Climbing, Gradient Descent, and Bayesian Optimization to discover the optimal solution quickly. In addition, dynamic nature of resource interference in networks and file systems require search algorithms to always search for the optimal to adapt the changing conditions and maximize the utility. Thus, we configured the optimization algorithms to keep exploring the search space throughout the transfer. Note that `Falcon` uses a separate thread to gather and process performance metrics, thus the optimization process does not interfere with the transfer performance.

**Hill-Climbing:** In Hill-Climbing algorithm, the search process first determines the search direction to follow, then evaluates potential values sequentially as long as the utility is increasing at which point the search direction is reversed to continue the search in the other direction. In the context of the optimal concurrency exploration, the search process starts with a minimum concurrency value of 1 and increments it by one as long as the utility is higher than previous value. Even if the current optimal is detected, the search process will keep evaluating higher and lower values periodically in order to detect and adapt to changes. To compare two concurrency values $n_i$, and $n_{i+1}$ of two consecutive time intervals with corresponding utility values $u_i$ and $u_{i+1}$, we first calculate the rate of change, $\gamma_i$, as follows:

$$\gamma_i = \frac{u_i - u_{i+1}}{u_{i+1}}$$

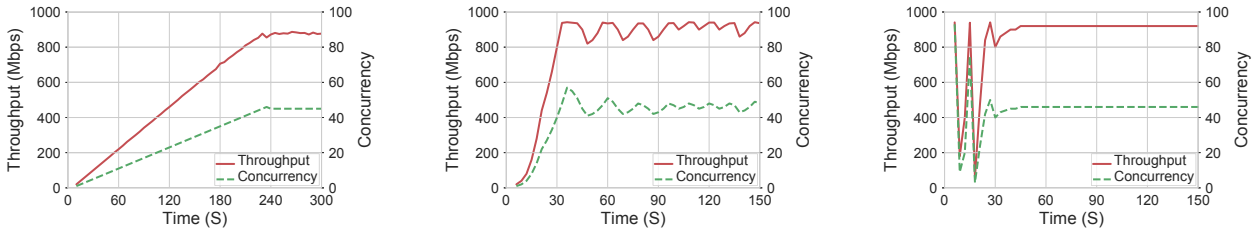
If $\gamma$ is greater than a non-negative threshold (3% by default), the search process will continue the search in the same direction to evaluate next concurrency value. As an example, if the search direction is downward (i.e., evaluating concurrency values lower than the current one) and $\gamma$ is larger than the threshold, then it will test $n_i - 1$ unless $n_i = 1$.

**Gradient Descent:** Online Gradient Descent (OGD) is extensively used for online convex function optimization due to its ability to adapt step size dynamically. Since the utility function in Equation 4 is strictly concave when $n \leq 100$, we can apply OGD[3] to search for the optimal concurrency value for transfers. As GD requires gradient (i.e., slope) calculation, we estimate it as follows: For a given concurrency value $n$, we test concurrency values $n + \epsilon$ and $n - \epsilon$ using sample transfers and calculate their utility values, $u_1$ and $u_2$, respectively. Then, the gradient can be approximated by $\gamma = \frac{u_2 - u_1}{2n\epsilon}$. Note that since concurrency can only take integer values, we use 1 for the $\epsilon$. For example, to calculate the gradient at $n = 40$, `Falcon` will evaluate concurrency values of 39 and 41 through sample transfers and calculate $\gamma$ using corresponding utility values.

Note that gradient cannot be used directly to estimate the next concurrency value as its scale is different. Thus, we convert $\gamma$ to

[2]Concurrency requires new processes and network connections to be created

[3]We can convert the utility to cost function by multiplying it with $-1$ to apply Gradient Descent.




(a) Hill Climbing (b) Gradient Descent (c) Bayesian Optimization

**Figure 7: Performance comparison of Hill Climbing, Gradient Descent, and Bayesian Optimization in finding the concurrency level in Emulab. While Gradient Descent and Bayesian Optimization can converge to the optimal in less than 30 seconds, it take more than 250 second for Hill Climbing.**

the rate change for concurrency by dividing it to the utility of $n - \epsilon$ as $\Delta = \frac{\gamma}{u_{n-\epsilon}}$ and use it to predict next concurrency value, $n_{new} = n + \Delta$. To further improve the convergence speed while avoiding to take arbitrarily large steps due to sampling errors, we use a monotonically increasing learning factor $\theta$ to gradually build confidence over search direction and adjust the steps size dynamically via $n_{new} = n + \theta\Delta$. We initiate $\theta$ to a relatively small value and increase it as long as the search moves in the same direction in consecutive intervals. Specifically, we initialize $\theta$ to 1 and increase it by one at each time step as long as the gradient is positive and reset to the initial value otherwise. `Falcon` runs Gradient Descent continuously even after it discovers the current optimal to adapt dynamic conditions. It does this by checking higher and lower values around the current optimal to find if they yield higher utility than the current solution.

**Bayesian Optimization:** Although Gradient Descent offers quick convergence over Hill Climbing method by taking advantage of the concavity of the utility function, it has two drawbacks. First, gradient calculation requires two sample transfers (i.e., $n + 1$ and $n - 1$) to be executed, which may slowdown convergence speed since each sample transfer takes at least $3 - 5$ seconds to accurately evaluate the performance of a transfer setting. Second, despite taking advantage of gradient to adjust step sizes, it requires careful tuning of step size parameters, $\theta$ and $\gamma$, to take full benefit of GD, which can be a daunting task. We therefore implement sequential model based optimization, more precisely Bayesian Optimization (BO), to address these limitations using a non-parametric search method.

BO is widely used for black-box optimizations especially when cost function is expensive to evaluate [27, 43, 49, 50]. It aims to estimate analytical form of black-box functions by processing observed events via surrogate models, such as Gaussian Process. It starts with a prior probability and calculates posterior probability after an event is observed using Bayes' theorem which states that if one of the conditional probabilities is known along with individual probabilities of two dependent events, then the other conditional probability can be calculated as

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{6}$$

where $A$ and $B$ are the two conditionally dependent variables with associated probabilities of $P(A)$ and $P(B)$ and conditional probabilities of $P(A|B)$ and $P(B|A)$.

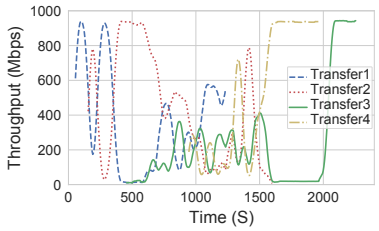
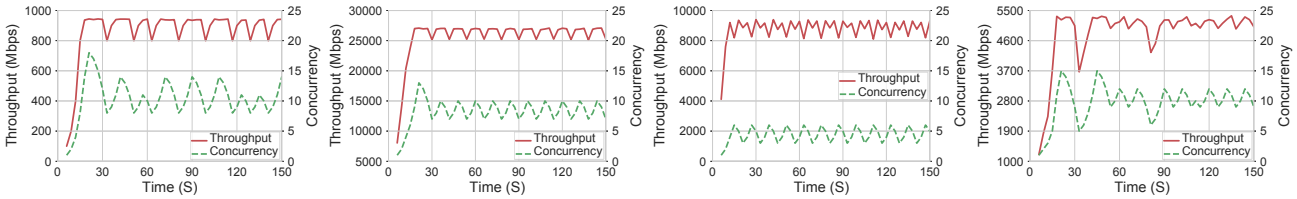



**Figure 8: Hill Climbing suffers from slow convergence speed and fails to achieve fairness among competing transfers.**

In the context of online transfer optimization, BO seeks to find a distribution that can capture the behavior of utility function with respect to value of concurrency parameter. It starts the process with a random sampling phase during which the utility of randomly selected concurrency values are evaluated. Although large number random sampling allows BO to discover the search space better, it increases the search time, thus we limit the random sampling phase to three samples in our experiments. We also set the prior distribution to uniform distribution to avoid any potential bias for optimal solution. We use Gaussian Process as a surrogate model and limit the number of past observations used in the model to 20 in order to (i) adapt dynamic system conditions quickly and (ii) lower the computational cost of Gaussian Process. While using entire history is helpful to find the optimal solution and build confidence over time, it hampers the discovery of new optimal when system conditions change. Limiting the number of past observations forces the optimizer to keep exploring the search space periodically and discover the new optimal quickly. Moreover, Gaussian Process (GP) is notorious for its expensive computational cost which increases as more data becomes available. Using a fixed number of past observations guarantees that GP processing delay stays in the order of milliseconds. Moreover, we utilize GP-Hedge algorithm [13] to tune the hyperparameters of BO, such as exploration-exploitation ratios and acquisition functions, in real-time.

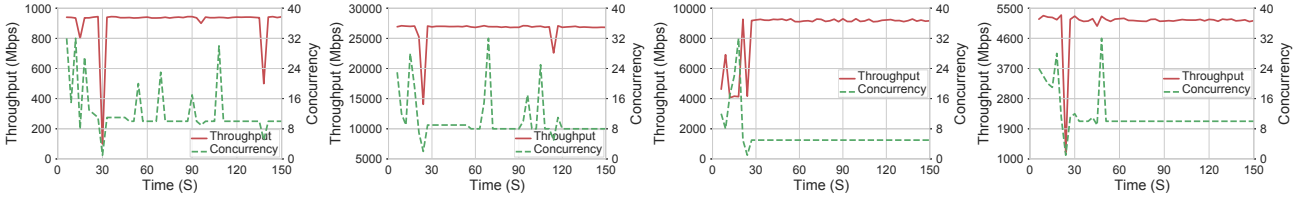
## 4 EVALUATION

We assess the performance of `Falcon` in four networks as listed in Table 1 out of which Campus Cluster and XSEDE are production HPC clusters, HPCLab is an isolated lab cluster, and Emulab is an emulated network testbed. Except Emulab, all clusters have parallel




(a) Emulab (b) HPCLab (c) Campus Cluster (d) XSEDE

**Figure 9: Performance evaluation of Falcon with Gradient Descent in different networks**




(a) Emulab (b) HPCLab (c) Campus Cluster (d) XSEDE

**Figure 10: Performance evaluation of Falcon with Bayesian Optimization in different networks**

| Testbed | Storage | Bandwidth | RTT | Bottleneck |
|---|---|---|---|---|
| Emulab | RAID-0 SSD | 1G | 30ms | Network |
| XSEDE | Lustre | 10G | 40ms | Disk Read |
| HPCLab | NVMe SSD | 40G | 0.1ms | Disk Write |
| Campus Cluster | GPFS | 10G | 0.1ms | NIC |

**Table 1: Specifications of test environments. OSG and Comet sites are used for XSEDE experiments.**

file systems or RAID arrays as storage solution due to which concurrent file transfers is required to achieve full I/O utilization. Since Emulab nodes have direct-attached single disk storage volumes, we throttle per process disk read throughput to necessitate concurrent I/O accesses to reach maximum performance, similar to parallel file systems. We also configured network topology in Emulab in a way that (Figure 3) network bandwidth becomes the bottleneck once sufficient number of concurrent transfers are created. We use OSG and Comet clusters for XSEDE experiments, which are connected via high-speed network. HPCLab and Campus Cluster servers are located in the same local-area network, thus delay between the hosts are less than a millisecond. To determine the bottleneck for file transfers, we used profiling tools (e.g., Iperf [5], bonnie++ [1]) that can capture "true" capacity of resources. We set the duration of sample transfers (i.e., evaluating the performance of a concurrency value) to 3 seconds in local area transfer and 5 seconds for wide-area transfers. Finally, we used a dataset containing $1000 \times 1$ GB files to conduct the transfers. § 4.4 presents results of Falcon when it's used for small and mixed datasets transfer optimizations.

## 4.1 Efficiency for Single Transfer

We first assess the performance of Falcon with Hill Climbing, Gradient Descent (GD) and Bayesian Optimization (BO) in terms of convergence speed and transfer throughput for single transfer scenario when the optimal is concurrency level is high. Specifically, we use Emulab and throttle per-process I/O throughput to require 48 concurrent transfers before network bandwidth can become the bottleneck. The results as illustrated in Figure 7 indicate that Hill Climbing takes nearly 7× longer to converge to the optimal compared to GD and BO as it uses a fixed step size of 1 while searching for the optimal solution. The impact of slow convergence is exacerbated when multiple transfers are executed in parallel as demonstrated in Figure 8. Although using a strictly concave function for utility calculations guarantees Nash Equilibrium between competing transfer tasks (i.e., agents) as long as all agents employ a similar strategy, it takes extremely long for Hill Climbing due to slow convergence speed. On the other hand, both GD and BO can converge to the optimal after $4-5$ sample transfers. Therefore, we only evaluate the performance of Falcon with GD and BO search algorithms in the rest of the paper.

Figure 9 and 10 present the performance of Falcon in all four networks with GD and BO online search implementations. We limit the I/O performance to 100 Mbps per process in Emulab with 1 Gbps network bandwidth, so 10 concurrent transfers is needed for full utilization. BO, after few initial random sampling, learns the optimal search region and focuses around concurrency value of 10 as shown in Figure 10(a). Since the number of past observations used in the surrogate model is limited, it periodically conducts search space exploration even after converging to the optimal. GD, on the other hand, starts the search with a initial concurrency value of 2 but converges to the optimal quickly by adjusting the step size. Upon convergence, the concurrency value bounces between 9 and 11 as it evaluates higher and lower values for continuous optimization. Although BO also runs continuous search, it can adjust the frequency of explorations based on recent observations, thereby offering more consistent performance over GD. Both BO and GD are able to reach over 25 Gbps throughput in HPCLab which requires around 9 concurrent transfers to reach maximum write I/O throughput. Similarly, BO and GD yields similar high

performance in Campus Cluster and XSEDE transfers by obtaining around 9.2 Gbps and 5.4 Gbps transfer throughput, respectively.

## 4.2 Convergence Analysis for Competing Falcon Transfers

We next assess the efficiency and fairness of Falcon when multiple transfer tasks run simultaneously. Note that this is different than concurrency in a way that concurrency transfers multiple files for same transfer task whereas competing transfer tasks refer transfer that are transferring completely separate datasets and likely submitted by different users. We assume that all agents (i.e., transfer tasks) uses the same utility function as in Equation 4 and employ the same (either GD or BO) search optimization algorithm. Figure 11 shows that GD supports fair resource sharing between the competing transfers while sustaining high network utilization. Compared to BO (as presented in Figure 12), GD leads to smaller fluctuations in throughput upon convergence as its adaptive step size policy constrains the search region in a small space once the optimal has been discovered. On the other hand, unlike single transfer scenario, Falcon BO agents do not settle to a fixed concurrency value when multiple agents compete since BO agents take larger steps during exploration phase. Nevertheless, average throughput of competing flows is nearly identical with the help of strictly concave utility function.

For HPCLab transfers (Figure 11(b) and 12(b)), we observe that when the second and third Falcon transfer agents join, they can quickly grab their fair share (i.e., $12-13$ Gbps for two transfers and $7-8$ Gbps for three transfers). Moreover, when one of the transfers completes, the remaining one(s) can quickly increase their utilization to sustain high performance. Figure 13 presents concurrency values for Falcon-GD agents when they compete against each other in Emulab where the bottleneck link capacity is 1 Gbps and I/O limit per process is 20 Mbps. Thus, it requires 48 concurrent transfers to reach maximum utilization. When there is only one transfer in the system, it quickly converges to the optimal concurrency value of 48 to maximize throughput. Once the second transfer joins, the first transfer reduces its concurrency to $20-33$ range to let the second transfer claim its fair share. Note that even if fair resource sharing can also be achieved when both transfers uses concurrency value of 48, it will result in higher packet loss despite obtaining the same throughput (i.e., around 500 Mbps for each transfer) when they both create $20-25$ concurrency transfers as illustrated in Figure 4. When the third transfer joins, all they all select concurrency values around $10-23$ to make sure that total concurrency value large enough to fully utilize available resources yet not too high to cause extreme network packet loss, computation, and and I/O overhead. Moreover, Falcon agents can also quickly notice the departure of competing transfers and increase their concurrency accordingly to claim the available network bandwidth.

## 4.3 Comparison to State-of-the-Art

Figure 14 compares the single transfer performance of Falcon against two state-of-the-art file transfer optimization solutions for the transfer of 1 TB dataset that consisting of $1000 \times 1$ GB files. Globus [3] is a web-based transfer service that is widely used in science and education community to schedule large file transfers. It relies on heuristic solution to tune concurrency along with parallelism and pipelining. It uses fixed settings to configure the value of transfer settings, thus fails to react to dynamic conditions. HARP [10], on the other hand, relies on historical data to drive regression models to estimate transfer throughput based on concurrency, parallelism and pipelining parameters. We observe that while Globus is too conservative when selecting the number of concurrent transfers to avoid congestion, HARP can be too aggressive to maximize throughput. Although HARP can reconfigure the transfers settings in the runtime to adapt changes [11], its performance is inherently limited due to relying on historical data.

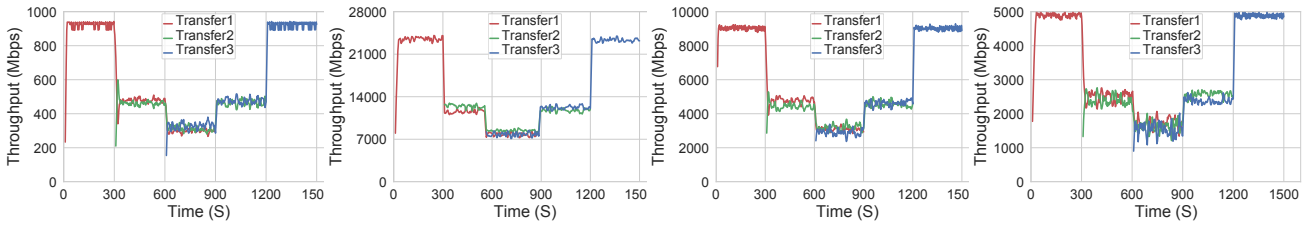
Globus underperforms significantly compared to Falcon in all three networks. Specifically, it yields around 9 Gbps throughput in HPCLab whereas Falcon attains more than 22 Gbps. HARP, on the other hand, yields $25-35\%$ lower throughput than Falcon in HPCLab and XSEDE networks while obtaining comparable results in Campus Cluster. The convergence behavior of HARP in the presence of multiple HARP agents, however, is far from optimal as late comers have higher advantage over existing transfers as illustrated in Figure 2(b). Specifically, even though overall system utilization increases from 16 Gbps to around 22 Gbps when the second HARP transfer starts, the first transfers only yields half the throughput of the second transfer, causing fairness issues.

## 4.4 Multi-parameter Optimization

Although concurrency is the most effective parameter in increasing transfer throughput due to offering both I/O and network parallelism [33, 47], additional transfer parameters such network *parallelism* and command *pipelining* can be tuned to further improve transfer performance, especially for long running transfers. *Parallelism* determines the number of concurrent network connections to create to transfer each file, which can be helpful to improve the performance in the case of transferring few large files. *Pipelining*, on the other hand, sends multiple file transfer commands to source and destination servers back-to-back such that the transfer of next file can start immediately after the transfer of previous file completes. *Pipelining* is mainly helpful when transferring many small files by eliminating the pauses between consecutive transfers, by amplifying the impact of short pauses. In terms of system overhead, *parallelism* can overburden network resources by creating too many concurrent flows. On the contrary, *pipelining* has negligible impact on system resources since it is merely command caching on source and destination servers. Therefore, we modified the cost function in Equation 4 to incorporate a penalty term for *parallelism* as

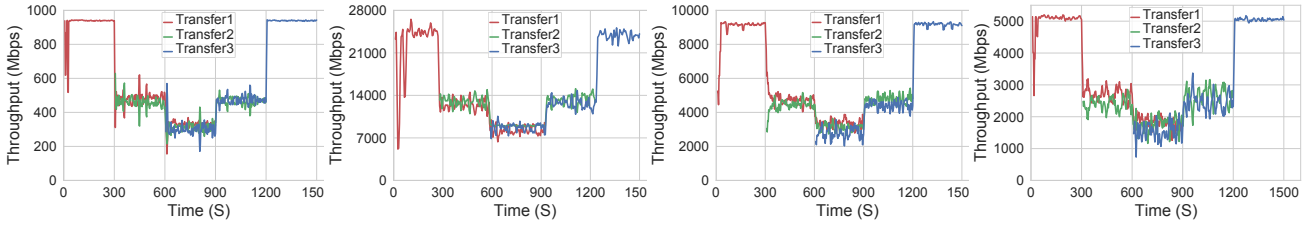
$$u(n_i, t_i, L_i) = \frac{(n_i \times p_i)t_i}{K^{n_i \times p_i}} - n_i t_i L_i \times B \tag{7}$$

where $p_i$ refers to the level of *parallelism*. Note that *parallelism* can be used in together with concurrency, so $n_i \times p_i$ is used to calculate the total number of network connections created for a given transfer. As an example, if *parallelism* is set to 4 while concurrency is set to 5, then Falcon will transfer 5 files simultaneously and create 4 network connections for each file, resulting in a total of 20 network connections to be used. To optimize the search process, we adopted conjugate gradient descent which provides efficient search for multi-parameter optimization problems [18].




(a) Emulab (b) HPCLab (c) Campus Cluster (d) XSEDE

**Figure 11: Stability analysis of Falcon with Gradient Descent upon convergence when multiple Falcon agents compete.**




(a) Emulab (b) HPCLab (c) Campus Cluster (d) XSEDE

**Figure 12: Stability analysis of Falcon with Bayesian Optimization upon convergence when multiple Falcon agents compete.**

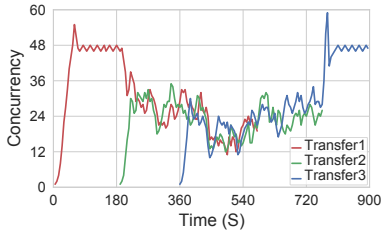


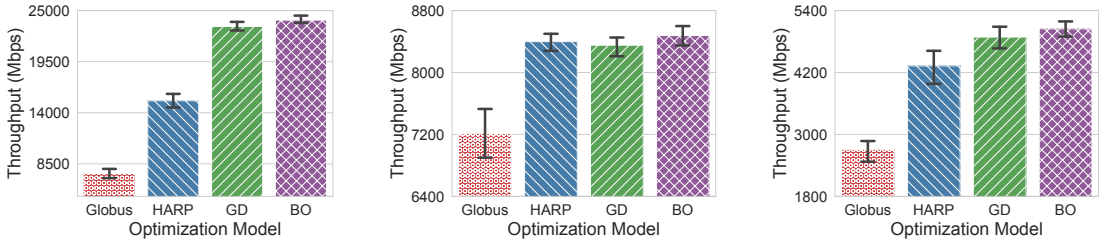

**Figure 13: Falcon-senders reduce their concurrency values when new transfers join to offer fair resource sharing with minimal system overhead.**

Figure 15 demonstrates the performance comparison of multi-parameter optimization for transfers from Stampede2 to Comet clusters that are connected with 40 Gbps network bandwidth and 60 ms delay. We evaluate the performance for the transfer of three different datasets as *small* that contains files whose size range between 1 KiB to 10 MiB for total of 120 GiB, *large* which contains files whose size range between 100 MiB to 10 GiB with a total of 1 TiB, and *mixed* that contains all files in *small* and *large* datasets with a total size of 1.2 TiB. We observe that Falcon yields up to 30% higher throughput when used to tune (*concurrency, parallelism, and pipelining* altogether (Falcon_MP) compared to its performance when tuning only *concurrency* (Falcon) for *small* and *mixed* datasets. This can be attributed to the the importance of command *pipelining* when dataset contains very small files. On the other hand, it results in 18% decrease in overall throughput for *large* dataset which can be attributed to two reasons. First, the utility function for multiparameter optimization (Equation 7) is not strictly concave function, thus there is no guarantee that it will converge to the optimal solution. Second, multiparameter optimization takes significantly longer time (up to 3× longer) to converge to a solution compared to single parameter optimization counterpart, causing more time to be spent in during the search phase in which throughput is typically lower than the throughput after convergence.

## 4.5 Friendliness Towards Non-Falcon Transfers

To evaluate the Falcon's friendliness to Globus and HARP, we run Falcon along with the others for the transfer of a 1.1TiB dataset that consists of files whose size range between 100MiB and 10GiB between Stampede2 and Comet clusters. When alone, Falcon is able to yield 26 – 28 Gbps throughput as shown in Figure 15. When the Globus transfer is started, is selects the concurrency value of 2 and obtains 4.9 Gbps throughput. Then, we initiate the HARP transfer which creates 11 concurrent transfers based on real-time sample transfer results and transfer logs in the historical dataset. Its throughput stabilizes at around 10.5 Gbps without affecting the performance of the Globus transfer as end-to-end transfer capacity is more than their cumulative throughput. Finally, we start the Falcon transfer at around 120s. The Gradient Descent (GD) implementation increases its concurrency gradually and converges to 16 – 18, which returns 12 – 13 Gbps throughput as shown in Figure 16(a). Although GD evaluates higher concurrency values and observes an increase in throughput, the improvement rate does not meet the desired level (i.e., nearly 2% for every concurrency value) once aggregate utilization nears to the capacity. As a result, it affects the performance of Globus and HARP transfers only marginally (around 15 – 20%). Therefore, it is fair to say that the GD implementation "plays well" in the presence of non-Falcon transfers by utilizing the spare capacity and abstaining aggressive behaviour against the competing flows.

On the other hand, Bayesian Optimization (BO) model is more aggressive against the non-Falcon transfers as it can probe very high concurrency values (> 40) during the initial search phase and




(a) HPCLab (b) Campus Cluster (c) XSEDE

**Figure 14: Performance comparison of Falcon against state-of-the-art file transfer optimization algorithms when transferring 1 TB dataset.**

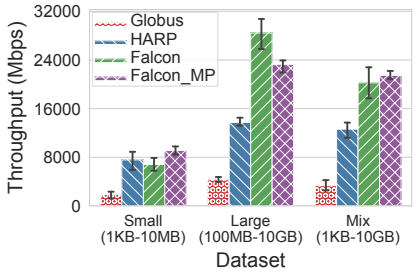
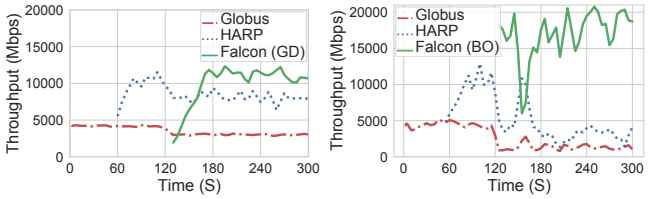



**Figure 15: Performance evaluation of Falcon for multi-parameter optimization.**




(a) Falcon-Gradient Descent (b) Falcon-Bayesian Optimization

**Figure 16: Performance analysis of Falcon when competing against non-Falcon transfers.**

obtain higher utility. In contrast, When multiple Falcon-BO transfers compete and one of them selects a high concurrency value, the others will respond by increasing their concurrency as well, causing all transfers to suffer from increased packet loss and/or decreased throughput to concurrency ratio. As a result, all transfer agents lower their concurrency to improve their utility. In the case of competition against HARP and Globus transfers, however, BO observes higher throughput gain which counteracts the increase in the concurrency value thereby leading to higher utility. In Figure 16(b), Falcon-BO creates 41 transfers to maximize its gain in exchange of severe performance degradation (around 70%) for HARP and Globus transfers performance as shown Figure 16(b). These results indicate that GD is a better fit for transfer optimizations in shared networks as it is able to maximize resource utilization without having severe impact on the performance of non-Falcon transfers.

## 4.6 Bayesian Optimization vs Gradient Descent

**Transfer Throughput:** Both Bayesian Optimization (BO) and Gradient Descent (GD) perform fairly well in all networks as they achieve high utilization and fair resource sharing. On the other hand, BO is able to yield slightly higher throughput for single transfer scenario as it is able to adjust its exploration frequency dynamically. In other words, as opposed to GD which evaluates the performance of alternative transfer settings around the optimal at fixed intervals (typically once in every 20 seconds) to detect and adapt to changing network conditions, BO can lower its search frequency if it realizes that system conditions are more or less stable. By doing so, it spends less time with suboptimal settings and attains higher average throughput. In the presence of competing transfers, however, BO performs similar to GD in terms of continually evaluating alternative settings as actions taken by any one of the competing Falcon agents triggers all others to search for the new optimal. As a result, they both achieve similar and close-to-optimal aggregate throughput when multiple transfers compete for the same resources.

**Search Phase Stability:** GD explores the search space is a more systematic way by identifying the search direction first and gradually increases its step size as it builds confidence. On the other hand, BO can probe any value in the defined search space during intial search phase, which could result in sample transfers with very high concurrency or parallelism values. This in turn may overwhelm end hosts and networks if the search space is not configured properly. For instance, if maximum values of concurrency and parallelism are defined as 32 for both parameters, then BO may probe a transfer setting when both concurrency and parallelism is set to 32, causing 1,024 network connections to be created. One potential fix to this problem involves dynamically configuring maximum values for search parameters in a way that they will be set to relatively small values and incremented only if the optimal is found to be closer to the maximum settings. For instance, maximum value of concurrency can be set to 16 at the beginning of the transfer and is incremented to 32 only after BO finds the concurrency value of 16 to be the the optimal configuration. It is important to note that such dynamic approach will cause a delay in the search discovery of the optimal setting as it will limit BO's ability to full scale search at the beginning. Another and potentially more important concern is the effective implementation of dynamic search space adjustments in the presence of competing transfers since the optimal transfer settings will not be static in such conditions. This may also cause competing transfers to possibly have different search space, violating critical requirement to achieve fair resource sharing among competing agents. As a result, GD offers more predictable

and conservative solution to explore large search spaces without overwhelming system resources.

**Convergence Stability:** When there is only one transfer in the system, both GD and BO offer relatively stable performance as they either keep the optimal solution or evaluate values around the optimal. On the other hand, GD offers more robust performance when multiple transfer tasks execute simultaneously as it has more restricted action space compared to BO which can test the opposite ends of the search space in consecutive intervals.

**Ease of Use:** Although we use GP-Hedge to tune most hyperparameters of BO, it still requires upper limit to be defined for concurrency value in order to draw the boundaries of search space. While setting upper limit to large values (e.g., 100) will ensure to find the optimal in almost all networks, it has two implications. First, it will likely to probe very large concurrency values during random sampling phase which may overwhelm end hosts and file system. Second, it will expand the search space significantly thereby increasing the convergence time. On the contrary, GD can detect the limits itself based on gradient and utility calculations. On the other hand, GD requires fine tuning of step size calculations for $\theta$ and $\gamma$ parameters.

## 5 RELATED WORK

As the trend toward more data-intensive applications continues, developers and users need to invest significant effort into moving large datasets between distributed sites efficiently. To keep up with increasing transfer rates, Internet-2 has upgraded its backbone network bandwidth to 400 Gbps [4]. Although there are efforts to reduce the amount of data transferred across wide area networks, such as data reduction [38, 44] and in-situ processing [16, 30], it is critical to fully utilize available network bandwidth for the end-to-end performance required by most commercial and scientific applications, where data transfer is inevitable. For example, it is estimated that cosmology simulations will create 50 PiB data monthly, part of which needs to be moved across supercomputer centers and storage sites which requires roughly 1TiB/hour transfer rates [15]. Even though the existing high-speed networks with hundred-gigabit-per-second bandwidth partially mitigates network bottlenecks, many users still experience difficulty reaching the theoretical maximum throughput, causing underutilization of resources. Moreover, previous work on transfer optimization between cloud datacenters mainly focus on the scheduling of transfers to mitigate network congestion [26, 32, 37]. We believe that this work can complement transfer scheduling solutions as `Falcon` will ensure that selected network paths are utilized efficiently to avoid resource wastage.

ESNet developed ScienceDMZ [6, 19, 40] and Data Transfer Nodes (DTN) [2] to improve the performance of high-speed file transfers. ScienceDMZ mainly helps to separate science flows from regular internet traffic to mitigate interference. It also creates a special path for scientific flows to bypass firewalls and other middleware devices to eliminate performance issues. While ScienceDMZs and custom DTNs address some of the issues, several major reasons for poor transfer performance remain unsolved, such as low disk I/O performance, poor transport protocol performance, and buffer size limitations. A common way to address performance problems for file transfers is tuning application-layer transfer settings such as pipelining [21], parallelism [22, 28, 36], concurrency [33], buffer size [25], block size [41], and striping [8]. These parameters when tuned carefully can significantly improve the end-to-end data transfer performance. There have been several attempts to tune some of these application-layer parameters to maximize transfer throughput using heuristic models [9, 12], historical data-based models [11, 29, 39], and stochastic approximation [34, 48]. Ito et al. [24] proposed Golden Section Search [42] algorithm to automatically adjust the number of parallel TCP connections for the GridFTP transfer protocol. Thulasidasan et al. applied dynamic right-sizing [20], an automatic and scalable buffer management technique for enhancing TCP performance, to wide-area data transfers [46]. Prasanna et al. [14] proposed direct search optimization to dynamically tune transfer parameters on the fly based on measured throughput for each transferred chunk. While these solutions offer better performance over supervised models, they typically take long time to converge to a solutions. Moreover, fairness is not guaranteed in solutions as they simply focus on increased throughput.

Globus [3] is a widely-adopted, robust data transfer service which uses heuristic approach to tune three application-layer parameters, pipelining, parallelism and concurrency. Yun et al. proposed ProbData [48] to tune the number of parallel streams and buffer size for memory-to-memory TCP transfers using stochastic approximation. ProbData is able to explore the near-optimal configurations through sample transfers but it takes several hours to converge which makes it impractical to use for most transfers in high-speed networks that last less than few hours [35]. Also, shared nature of production systems causes background traffic to change drastically over several hours, so it may even fail to converge due to large variations in sample transfers. Moreover, it does not guarantee stable and fair resource allocation in shared environments which is significant barrier in adoption by scientific community as most of the production-level high speed networks (i.e., XSEDE, ESNet, Internet-2 etc.) are shared by hundred of users if not more. Yildirim et al. proposed PCP [47] to the values of application layer parameter in the runtime. It uses simple hill climbing method to identify the optimal value, thus lead to suboptimal performance in most cases.

## 6 CONCLUSION AND FUTURE WORK

File transfers in high performance networks require end-to-end parallelism to efficiently utilize available resources. However, determining the optimal level of parallelism is challenging task as suboptimal solutions can lead to underutilization or overwhelmed network and file systems. Previous work in this area implemented heuristic and supervised learning solutions both of which fail to satisfy high resource utilization while inducing low overhead to end systems and networks. To address this problem, we introduce `Falcon` that combines novel utility function with state-of-the-art online optimization techniques to guarantee high performance, fair resource sharing, and minimal overhead. Specifically, `Falcon` constructs a novel utility function that rewards high throughput while penalizing for increased packet loss and the number of active concurrent processes. To minimize the search overhead and converge to optimal transfer settings quickly, `Falcon` utilizes online search algorithms Hill Climbing, Gradient Descent and Bayesian Optimization models. The experimental results show that `Falcon` yields up to 6× higher throughput compared to state-of-the-art solutions while

keeping its overhead at minimum. More importantly, Falcon converges to a fair and stable state in the presence of multiple independent transfers. We further demonstrate that Falcon can easily be extended to tune multiple transfer parameters, paving the way for high-precision tuning for long-running science workflows with stringent performance expectations.

As a future direction, we will evaluate the performance of Falcon for emerging congestion control algorithms such as BBR [17] to check the feasibility of developing a congestion control-agnostic solution. Moreover, we aim to explore cross-layer optimization solutions to tune application- (e.g., the number of concurrent transfers) and transport- (e.g., loss and delay tolerance) layer parameters together to remove redundancies and yield higher overall performance. Finally, we are working toward the release of open-source version of Falcon. To this end, we intend to develop cloud-based web service to deploy Falcon which will facilitate its adoption by a wider user community by eliminating tedious installation process.

## REFERENCES


[1] 2021. Bonnie++. https://linux.die.net/man/8/bonnie++.
[2] 2021. Data Transfer Nodes. http://fasterdata.es.net/science-dmz/DTN/.
[3] 2021. Globus. https://www.globus.org.
[4] 2021. Internet2 Next Generation Infrastructure Update. https://internet2.edu/internet2-next-generation-infrastructure-update-29-packet-nodes-connected-by-40-400g-links-gdt-install-completes-on-schedule/.
[5] 2021. Iperf. https://iperf.fr.
[6] 2021. Science DMZ. https://fasterdata.es.net/science-dmz/.
[7] Ismail Alan, Engin Arslan, and Tevfik Kosar. 2015. Energy-aware data transfer algorithms. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.
[8] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. 2005. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 54.
[9] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke. 2012. Software as a Service for Data Scientists. *Commun. ACM* 55:2 (2012), 81–88.
[10] Engin Arslan, Kemal Guner, and Tevfik Kosar. 2016. HARP: predictive transfer optimization based on historical analysis and real-time probing. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 288–299.
[11] Engin Arslan and Tevfik Kosar. 2018. High-Speed Transfer Optimization Based on Historical Analysis and Real-Time Tuning. *IEEE Transactions on Parallel and Distributed Systems* 29, 6 (2018), 1303–1316.
[12] Engin Arslan, Bahadir A Pehlivan, and Tevfik Kosar. 2018. Big data transfer optimization through adaptive parameter tuning. *J. Parallel and Distrib. Comput.* 120 (2018), 89–100.
[13] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 1995. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, 322–331.
[14] P. Balaprakash, V. Morozov, R. Kettimuthu, K. Kumaran, and I. Foster. 2016. Improving Data Transfer Throughput with Direct Search Optimization. In *2016 45th International Conference on Parallel Processing (ICPP)*. 248–257. https://doi.org/10.1109/ICPP.2016.36
[15] Julian Borrill, Eli Dart, Brooklin Gore, Salman Habib, Steven T Myers, Peter Nugent, Don Petravick, and Rollin Thomas. 2015. Improving Data Mobility & Management for International Cosmology: Summary Report of the CrossConnects 2015 Workshop. (2015).
[16] Jose J Camata, Vitor Silva, Patrick Valduriez, Marta Mattoso, and Alvaro LGA Coutinho. 2018. In situ visualization and data analysis for turbidity currents simulation. *Computers & Geosciences* 110 (2018), 23–31.
[17] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. *Queue* 14, 5 (2016), 50.
[18] Yu-Hong Dai and Yaxiang Yuan. 1999. A nonlinear conjugate gradient method with a strong global convergence property. *SIAM Journal on optimization* 10, 1 (1999), 177–182.
[19] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. 2014. The science dmz: A network design pattern for data-intensive science. *Scientific Programming* 22, 2 (2014), 173–185.
[20] Mike E Fisk and Wu-chun Feng. 2001. *DYNAMIC RIGHT-SIZING IN TCP*. Technical Report. Los Alamos National Laboratory.
[21] N. Freed. [n.d.]. SMTP service extension for command pipelining. http://tools.ietf.org/html/rfc2920.
[22] T. J. Hacker, B. D. Noble, and B. D. Atley. 2005. Adaptive Data Block Scheduling for Parallel Streams. In *Proceedings of HPDC '05*. ACM/IEEE, 265–275.
[23] Elad Hazan. 2016. Introduction to online convex optimization. *Foundations and Trends® in Optimization* 2, 3-4 (2016), 157–325.
[24] Takeshi Ito, Hiroyuki Ohsaki, and Makoto Imase. 2008. GridFTP-APT: Automatic parallelism tuning mechanism for GridFTP in long-fat networks. *IEICE transactions on communications* 91, 12 (2008), 3925–3936.
[25] T. Ito, H. Ohsaki, and M. Imase. 2008. On parameter tuning of data transfer protocol GridFTP for Wide-Area Networks. *International Journal of Computer Science and Engineering* 2(4) (Sept. 2008), 177–183.
[26] Su Jia, Xin Jin, Golnaz Ghasemiesfeh, Jiaxin Ding, and Jie Gao. 2017. Competitive analysis for online scheduling in software-defined optical WAN. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 1–9. https://doi.org/10.1109/INFOCOM.2017.8056969
[27] Donald R Jones, Matthias Schonlau, and William J Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13, 4 (1998), 455–492.
[28] R. P. Karrer, J. Park, and J. Kim. 2006. TCP-ROME:performance and fairness in parallel downloads for Web and real time multimedia streaming applications. In *Technical Report*. Deutsche Telekom Laboratories.
[29] Rajkumar Kettimuthu, Gayane Vardoyan, Gagan Agrawal, and P Sadayappan. 2014. Modeling and optimizing large-scale wide-area data transfers. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 196–205.
[30] James Kress, Randy Michael Churchill, Scott Klasky, Mark Kim, Hank Childs, and David Pugmire. 2016. Preparing for in situ processing on upcoming leading-edge supercomputers. *Supercomputing Frontiers and Innovations* 3, 4 (2016), 49–65.
[31] Yee-Ting Li, Douglas Leith, and Robert N Shorten. 2007. Experimental evaluation of TCP protocols for high-speed networks. *IEEE/ACM Transactions on networking* 15, 5 (2007), 1109–1122.
[32] Chuan Lin, Yuanguo Bi, Guangjie Han, Jucheng Yang, Hai Zhao, and Zheng Liu. 2018. Scheduling for Time-Constrained Big-File Transfer Over Multiple Paths in Cloud Computing. *IEEE Transactions on Emerging Topics in Computational Intelligence* 2, 1 (2018), 25–40. https://doi.org/10.1109/TETCI.2017.2755692
[33] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. 2019. Data transfer between scientific facilities–bottleneck analysis, insights and optimizations. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 122–131.
[34] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Peter H Beckman. 2018. Toward a smart data transfer node. *Future Generation Computer Systems* (2018).
[35] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Nageswara SV Rao. 2018. Cross-geography scientific data transferring trends and behavior. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 267–278.
[36] D. Lu, Y. Qiao, and P. A. Dinda. 2005. Characterizing and Predicting TCP Throughput on the Wide Area Network. In *Proceedings of ICDCS '05*. IEEE, 414–424.
[37] Ping Lu, Kaiyue Wu, Quanying Sun, and Zuqing Zhu. 2015. Toward online profit-driven scheduling of inter-DC data-transfers for cloud applications. In *2015 IEEE International Conference on Communications (ICC)*. 5583–5588. https://doi.org/10.1109/ICC.2015.7249212
[38] Tao Lu, Eric Suchyta, Dave Pugmire, Jong Choi, Scott Klasky, Qing Liu, Norbert Podhorszki, Mark Ainsworth, and Matthew Wolf. 2017. Canopus: A Paradigm Shift Towards Elastic Extreme-Scale Data Analytics on HPC Storage. In *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 58–69.
[39] MD SQ Zulkar Nine and Tevfik Kosar. 2020. A Two-Phase Dynamic Throughput Optimization Model for Big Data Transfers. *IEEE Transactions on Parallel and Distributed Systems* 32, 2 (2020), 269–280.
[40] Sean Peisert, William Barnett, Eli Dart, James Cuff, Robert L Grossman, Edward Balas, Ari Berman, Anurag Shankar, and Brian Tierney. 2016. The medical science DMZ. *Journal of the American Medical Informatics Association* 23, 6 (2016), 1199–1201.
[41] Mohammad Javad Rashti, Gerald Sabin, and Rajkumar Kettimuthu. 2016. Long-haul secure data transfer using hardware-assisted GridFTP. *Future Generation Computer Systems* 56 (2016), 265–276.
[42] Mary C Seiler and Fritz A Seiler. 1989. Numerical recipes in C: the art of scientific computing. *Risk Analysis* 9, 3 (1989), 415–416.
[43] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
[44] Renan Souza, Vítor Silva, Alvaro Coutinho, Patrick Valduriez, and Marta Mattoso. 2016. Online Input Data Reduction in Scientific Workflows. In *WORKS: Workflows in Support of Large-scale Science*.
[45] Pratiksha Thaker, Matei Zaharia, and Tatsunori Hashimoto. [n.d.]. Learning and utility in multi-agent congestion control. *optimization* 24, 10 ([n. d.]), 11–18.

[46] Sunil Thulasidasan, Wu-chun Feng, and Mark K Gardner. 2003. Optimizing GridFTP through dynamic right-sizing. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*. IEEE, 14–23.

[47] Esma Yildirim, Engin Arslan, Jangyoung Kim, and Tevfik Kosar. 2015. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. *IEEE Transactions on Cloud Computing* 4, 1 (2015), 63–75.

[48] D. Yun, C. Q. Wu, N. S. V. Rao, Q. Liu, R. Kettimuthu, and E. Jung. 2017. Data Transfer Advisor with Transport Profiling Optimization. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. 269–277.

[49] Anatoly Zhigljavsky and Antanasz Zilinskas. 2007. *Stochastic global optimization*. Vol. 9. Springer Science & Business Media.

[50] A Žilinskas and J Žilinskas. 2002. Global optimization based on a statistical model and simplicial partitioning. *Computers & Mathematics with Applications* 44, 7 (2002), 957–967.

[51] Martin Zinkevich. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *ICML*. AAAI Press, 928–936. http://www.aaai.org/Library/ICML/2003/icml03-120.php

[52] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 928–936.