

# GreenDataFlow: Minimizing the Energy Footprint of Global Data Movement

MD S Q Zulkar Nine<sup>1</sup>, Luigi Di Tacchio<sup>1</sup>, Asif Imran<sup>1</sup>, Tevfik Kosar<sup>1</sup>, M. Fatih Bulut<sup>2</sup>, Jinho Hwang<sup>2</sup>

<sup>1</sup>Department of Computer Science and Engineering, University at Buffalo, Buffalo, New York

<sup>2</sup>IBM TJ Watson Research Center, Yorktown Heights, New York

Email: {mdsqzulk, luigidit, asifimra, tkosar}@buffalo.edu, {mfbulut, jinho}@us.ibm.com

**Abstract**—The global data movement over Internet has an estimated energy footprint of 100 terawatt hours per year, costing the world economy billions of dollars. The networking infrastructure together with source and destination nodes involved in the data transfer contribute to overall energy consumption. Although considerable amount of research has rendered power management techniques for the networking infrastructure, there has not been much prior work focusing on energy-aware data transfer solutions for minimizing the power consumed at the end-systems. In this paper, we introduce a novel application-layer solution based on historical analysis and real-time tuning called GreenDataFlow, which aims to achieve high data transfer throughput while keeping the energy consumption at the minimal levels. GreenDataFlow supports service level agreements (SLAs) which give the service providers and the consumers the ability to fine tune their goals and priorities in this optimization process. Our experimental results show that GreenDataFlow outperforms the closest competing state-of-the-art solution in this area 50% for energy saving and 2.5× for the achieved end-to-end performance.

## I. INTRODUCTION

The era of artificial intelligence (AI) has made data the most important resource, in turn the efficient data handling is the key to use compute, network, and storage resources more effectively. Not like compute and storage resources, the network resource needs more sophisticated control as it involves the end-to-end efficiency. The annual data transfer rate over global IP networks has already exceeded zettabyte scale [51]. The energy footprint of this global data movement is estimated at more than 100 terawatt hours per year at the current rate, costing more than 20 billion US dollars annually to the world economy in addition to the environmental side effects [27], [44], [41], [51], [22]. This fact has resulted in considerable amount of work focusing on power management and energy efficiency in hardware and software systems [12], [47], [59], [28], [16], [17], [49], [36], [30], [52], [46] as well as on power-aware networking [6], [41], [24], [31], [23], [21].

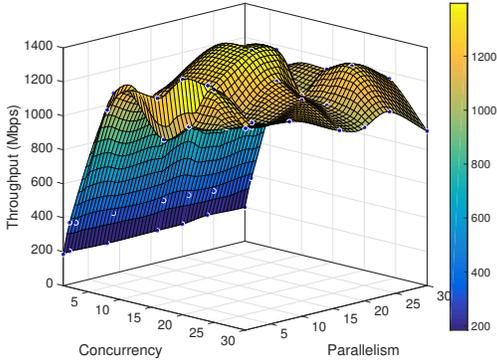
Majority of the existing work on power-aware networking focuses on reducing the power consumption on networking devices (i.e., routers, switches, and hubs). Gupta et al. [27] were amongst the earliest researchers to advocate conserving energy in the networking infrastructure. They suggested different techniques such as putting idle sub-components (i.e., line cards, etc.) to sleep [26], which were later extended by other researchers. Nedeovski et al. proposed adapting the rate at which switches forward packets depending on the traffic [43].

IEEE Energy Efficient Ethernet Task Force proposed the 802.3az standards [1] for making Ethernet cards more energy efficient. They defined a new power state called Low-Power Idle (LPI) that puts the Ethernet card to low power mode when there is no network traffic. Other related research in power-aware networking has focused on architectures with programmable switches [25], switching layers that can incorporate different policies [33], and power-aware network protocols for energy efficiency in network routing [13].

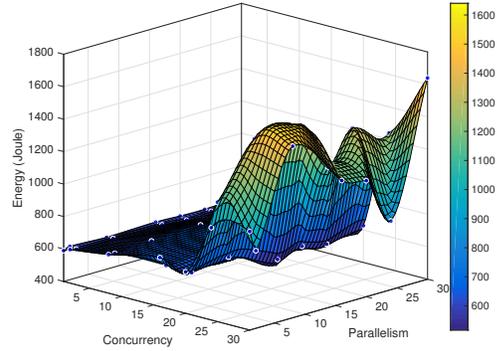
The existing approaches suffer from the following drawbacks: (1) the solution is too costly (i.e., replacing all switches with energy efficient ones); (2) the solution is unpractical in the short term (i.e., replacing TCP with a more energy-efficient version); (3) the solution penalizes performance while increasing energy efficiency (i.e., sleeping some components while not in use). In this paper, we propose an application-layer solution called GreenDataFlow which is low cost, very easy and practical to deploy, and does not penalize the performance while increasing energy efficiency. With the added benefits and simplicity to adopt, service providers can directly benefit from GreenDataFlow as they can offer it as-a-service offering in their cloud platforms, while making sure that the SLA requirements of customers are satisfied using GreenDataFlow's SLA-based algorithms.

GreenDataFlow provides novel two-phase dynamic optimization models to minimize energy and increase throughput at the same time. It is based on mathematical modeling with offline knowledge discovery and adaptive online decision making. During the offline analysis phase, we analyze historical transfer logs to perform knowledge discovery about the characteristics of the past transfers with similar requirements. During the online phase, we use the discovered knowledge from the offline analysis along with real-time investigation of the network condition to optimize the protocol parameters for both minimal energy consumption and maximum transfer throughput. Our models use historical knowledge about the network and data to reduce the real-time investigation overhead while ensuring near optimal results for each transfer. Specifically our contributions in this paper are as follows:

- 1) GreenDataFlow achieves minimizing the energy footprint of an end-to-end big data transfer by operating in the application-layer, without any need to change the existing infrastructure nor the low-level networking stack, which makes integration of GreenDataFlow to



(a) Achieved Throughput for different  $cc$  and  $p$



(b) Energy Consumption for different  $cc$  and  $p$

Figure 1: Achieved throughput and energy consumption of a single transfer under different parameter combination. Surface interpolation is perform using piece-wise cubic spline.

existing applications easier.

- 2) GreenDataFlow integrates knowledge-based offline analysis with real-time tuning to achieve close-to-optimal end-to-end data transfer throughput while reducing energy consumption.
- 3) GreenDataFlow applies adaptive tuning in real-time and uses pre-computed mathematical optimization based decisions to provide faster convergence toward maximally achievable throughput.
- 4) GreenDataFlow outperforms state-of-the-art solutions in this area in terms of accuracy, convergence speed, energy efficiency, and achieved throughput. Our experimental results show that GreenDataFlow outperforms the closest competing state-of-the-art solution in this area up to 50% for energy saving and up to  $2.5\times$  for the achieved end-to-end performance. When we compare it with baseline cases (without any optimization), the energy savings go up to 80% and performance improvement reaches  $10\times$ .

The rest of the paper is organized as follows: Section II gives a formal definition of the problem; Section III discusses the challenges in optimization; Section IV presents our novel two-phase dynamic optimization model; Section V evaluates our model; Section VI describes the related work in this field; and Section VII concludes the paper.

## II. PROBLEM FORMULATION

Large-scale data transfers can get suboptimal performance and high energy footprint in a long RTT WAN network due to the protocol inefficiency introduced in different layers. Changing the protocol stack requires low-level updates (e.g., modifications to TCP), and its adaptation by large-scale needs considerable time and effort. Therefore, application level solutions are more lucrative and easy to deploy in the user space which makes the adaptation of the solution hassle-free. Application level data transfer protocol parameters (i.e., concurrency, parallelism, pipelining, and buffer-size) can

have different impacts on transfer throughput and energy consumption of files with different sizes under certain network conditions. These parameters can be tuned to increase the data transfer throughput and decrease the energy footprint significantly. Figure 1 shows the achieved throughput and energy consumption of a single transfer under different parameter combinations. A short description of these parameters is given below.

**Concurrency** ( $cc$ ) refers to the task level parallelism. It controls the number of server processes where each process can transfer an individual file. It can accelerate the transfer throughput when a large number of files need to be transferred. Concurrency can also take advantage of parallel file systems (e.g., GPFS, Lustre) with multiple concurrent servers and metadata management.

Each server process can transfer a different portion of a file in parallel. We define the number of parallel streams for each server as **Parallelism** ( $p$ ). It is a good choice to transfer medium and large files. We can get the full performance of parallelism with parallel file systems where files are divided and distributed on different disks. The total number of parallel data streams can be expressed as  $(cc \times p)$ . Increasing number of parallel data streams can increase the achievable throughput, however, excessive use of streams may lead to packet loss and force TCP to initiate slow-start phase that may lead to severe throughput loss.

Control channel idleness is a major bottleneck to transfer a large number of files. After each file transfer, the server process sends an acknowledgment to initiate the next file transfer. This acknowledgment can take at least one Round-Trip-Time (RTT) between each transfer. This one RTT delay may seem innocent, however, it may hurt the overall throughput of a dataset containing a large number of small files significantly in a long RTT network. Because small files take short time to transfer and then each file needs to wait for one RTT to get acknowledgment from previous transfer. Moreover, TCP will shrink window size to zero if it detects data channel

idleness. These issues can be solved by queuing multiple file transfer requests without waiting for the acknowledgments. This technique can transfer large number files like a single large file. We define the size of the outstanding file transfer request queue as **Pipelining** ( $pp$ ).

Energy consumption is a major concern in data centers and end-systems that perform very large scale data transfers. Minimizing the energy consumption can reduce the data center operating cost-effectively while utilizing the network links efficiently. Energy consumption can be measured using specialized hardware meters. However, we need an energy consumption model to estimate the real-time consumption that can be used to facilitate fine-grained power tuning. Energy consumption can be estimated from the current load of the system, such as - CPU utilization ( $\mu_{cpu}$ ), memory utilization ( $\mu_{mem}$ ), disk usage ( $\mu_{disk}$ ), and network interface card utilization ( $\mu_{nic}$ ). In literature, there exist many models to predict the actual energy consumption using these load information. We have used a linear model to predict power consumption, which is presented in Section IV-A1.

Our aim is to perform an energy constrained optimization of the data transfer performance. End users or data center operations team may set these constrained optimization problem based on their requirements and priorities. In this work, we introduce easy to describe energy-aware Service Layer Agreement (SLA) categories that a user or an administrator can initiate. SLA is a contract between the user and the service provider where these two parties agree on service quality and specific rights of both parties. It may include the description of services agreed to be provided, monitoring and reporting of quality of service (QoS) matrices, the consequence of not meeting the requirements, and escape clause (the situation where service guarantee promised does not apply). For energy efficient transfers, SLA can be defined in three major ways: (1) Throughput guarantee (**Type-T**), (2) Constraint over total energy usage (**Type-E**), and (3) Constraint over instantaneous power consumption (**Type-P**). We translate the SLAs into appropriate optimization problems and solve them off-line. These SLA categories are explained below.

**(1) Throughput Guarantee:** A user may need an overall throughput guarantee that means the achievable throughput  $T_{act}$  must be at least the throughput specified in SLA,  $T_{sla}$ . In such case, the service provider would try to maintain the SLA requirement using as less energy,  $E$  as possible. Therefore, the optimization problem can be expressed as :

$$\begin{aligned} & \underset{\{cc,p,pp\}}{\operatorname{argmin}} && (E) \\ & \text{subject to.} && T_{act} \geq T_{sla} \end{aligned} \quad (1)$$

**(2) Constraint over Total Energy Usage:** User or administrator may want to minimize energy cost by putting a constraint over the total energy consumption and request for the best possible throughput under this constraint. Therefore, actual total energy consumption ( $E_{act}$ ) can be constrained by energy consumption level specified in SLA, ( $E_{sla}$ ). The

optimization problem is to maximize throughput,  $T$  under a specified energy constraint and can be expressed as:

$$\begin{aligned} & \underset{\{cc,p,pp\}}{\operatorname{argmax}} && \frac{1}{\tau_f - \tau_s} \int_{\tau_s}^{\tau_f} T \\ & \text{subject to.} && E_{act} \leq E_{sla}. \end{aligned} \quad (2)$$

Where  $\tau_s$  and  $\tau_f$  are the starting time and the finish time respectively.

**(3) Constraint over Instantaneous Power Consumption:** Sometimes spikes in instantaneous power consumption can be very expensive, as power grid imposes a high penalty for such spikes. The user may want to put constraint over instant power consumption. That means the user wants to maximize the throughput with a guarantee that instant power consumption should not exceed the power limit mentioned in the SLA. To do that we need to model the instant power,  $\varphi$  first based on resource utilization.

$$\varphi = f(\mu_{cpu}, \mu_{mem}, \mu_{disk}, \mu_{nic}) \quad (3)$$

$$\begin{aligned} & \underset{\{cc,p,pp\}}{\operatorname{argmax}} && \frac{1}{\tau_f - \tau_s} \int_{\tau_s}^{\tau_f} T \\ & \text{subject to.} && \varphi_i \leq \varphi_{sla}. \end{aligned} \quad (4)$$

For all three cases, we need to schedule the compute resources to the server processes in a way that minimizes the energy consumption or keep it below the SLA constraint and simultaneously increase throughput. Dynamic load balancing among the parallel streams also helps to alleviate extra load from the congested streams.

Several assumptions are made to design our optimization model. Those are explained below:

*Assumption 1.* For disk-to-disk transfer, the achievable throughput ( $T_{act}$ ) should be bounded by the bottleneck resource that can be end-to-end link bandwidth or disk read/write speed ( $v_{read}$  and  $v_{write}$ ) at the source and destination.

$$T_{act} \leq \min\{BW, v_{read}, v_{write}\} \quad (5)$$

*Assumption 2.* We aim to optimize the application-level parameters and our solution is agnostic of the underlying file system. On the other hand, if the target data is stored on a parallel file system, that would benefit more from our optimization.

*Assumption 3.* Our model is also agnostic of the underlying reliable transport protocol. Any reliable transport protocol can work with our model. In this paper, we apply our optimizations to GridFTP [38] which is based on TCP. GridFTP is widely used in the scientific community, and it supports easy tuning of parameters such as parallelism, concurrency, and pipelining.

*Assumption 4.* Our model can work with/without Remote Direct Memory Access (RDMA) technology. Surely RDMA technology can reduce the overhead introduced by the kernel copy. However, it requires special hardware (i.e., RDMA NIC) support which can be a deployment barrier for the RDMA technology in end users.

### III. CHALLENGES

The main challenge is the dynamic nature of the shared network links. Achievable transfer throughput can change from time to time during large-scale data transfers. To get required performance, the transfer needs online update and micro-tuning of the parameters, and real-time resource scheduling. Searching for updated parameters (i.e., solving the optimization problems mentioned in Section II) during the transfer is expensive and introduces delay or transfer may continue with suboptimal parameters until search finishes. Both can introduce throughput loss to the transfer. To address the issue we use offline analysis of historical transfer logs to solve optimization problems. In this case, we need to solve the optimization problem for all possible SLA requirements which may be infeasible and take a considerable amount of memory to store solutions. We introduce an intelligent way to overcome the issue (discussed in Section IV-A). We store the solution as the key-value pair. During the actual transfer, we periodically check network conditions and if needed we can look up for the new parameter settings in constant time. We introduce a two-phase optimization model which combines the benefits of offline analysis and online tuning to ensure the SLA requirements. We introduce some design-specific challenges below.

**Challenge 1. (Cost of offline analysis)** Offline analysis itself introduces extra energy and latency cost. If not performed efficiently, it may defeat the purpose of offline analysis, when the combined cost of offline analysis  $Cost(Offline)$  and the cost of actual data transfer,  $Cost(Tr_{tuned})$  exceed the cost of the transfer without optimization,  $Cost(Tr_{no\_opt})$ . So the constraint can be imposed as:

$$Cost(Offline) + Cost(Tr_{tuned}) < Cost(Tr_{no\_opt}) \quad (6)$$

**Challenge 2. (Packet loss)** Any packet loss can reduce the TCP window size (cwnd) significantly (depending on the TCP variant) which reduces the achievable transfer throughput. Packet loss can happen for a range of reasons, such as: (1) congestion, (2) bottleneck network devices with low capacity (routers/switch/middleboxes), (3) software bugs in network devices, and (4) faulty links. We need a mechanism to detect the static packet losses (2-4) and eliminate them manually. We also need a mechanism that can efficiently signal congestion beforehand to avoid the TCP window size reduction.

**Challenge 3. (Fairness)** We need to ensure bandwidth usage fairness among the contending transfers. This means our model should not be too aggressive to increase the achievable throughput for only a specific set of users or for specific transfers.

**Challenge 4. (Resource scheduling to sender/receiver processes)** We have explained in Section II that the energy consumption can be modeled as a function of resource utilization. Here, the goal is to find the optimal number of server processes ( $cc$ ) with multi-threaded sockets ( $p$ ) and pipelining ( $pp$ ) and

Symbol	Description
$T$	Throughput of the data transfer
$E$	Energy consumption of the transfer
$\varphi$	Instantaneous power consumption
$cc$	Concurrency
$p$	Parallelism
$pp$	Pipelining
$bs$	End system buffer size
$\mathbb{S}$	Total number of streams, $\mathbb{S} = cc \times p$
$\theta$	Set of parameters, $\theta = \{cc, p, pp, bs\}$
$BW$	Bandwidth
$\eta$	Energy efficiency, $Data/Energy$
$cc_{limit}, p_{limit}$	User limit on corresponding parameters
$\tau$	Time
$\epsilon$	Throughput fluctuation tolerance bound
$\mu$	Utilization of corresponding resource
$\mu_{all}$	Set of all resource utilization
$I$	Interpolant of Throughput or Energy log

Table I: Description of different symbols

then schedule the compute resources to the concurrent processes in a way to maintain the SLA agreement. Current CPU schedulers are not designed to meet such optimization goal. As an example, Linux uses Completely Fair Scheduling (CFS) and Real-time scheduler (RT) to schedule the CPU. CFS tries to mimic the perfectly fair scheduling. CFS achieves fairness by calculating the time slice as a fraction of the total number of running processes. In our case, not all the concurrent processes are pumping data at the same rate and some server processes may have more workload than the others. We may need to limit the CPU usage of certain server processes to maintain a steady power consumption rate. Current CFS scheduler cannot make such forced external modification. Cgroup is a Linux kernel tool that can provide more fine-grained external control over the resource scheduling.

### IV. MODEL DESIGN

We introduced two models to address the problem: (1) distributed and (2) centralized. In the distributed solution, there is no centralized control and all the users run their own optimization model and converge to receive a fair share of network throughput. However, in this situation, many users may end up solving the same or similar optimization problems. On the other hand, the centralized solution takes the burden of running optimization from the end users and performs optimization based on historical logs and shares it with the end users. The centralized scheduling is efficient for intra/inter data center large-scale transfers due to the fact that it can be integrated with Traffic Engineering (TE) module for joint optimization of throughput and energy consumption. The centralized scheduler can optimize data center energy consumption more efficiently than the distributed solution.

#### A. Distributed Approach

Our distributed approach has two phases: (1) offline optimization, (2) dynamic tuning. Offline optimization takes

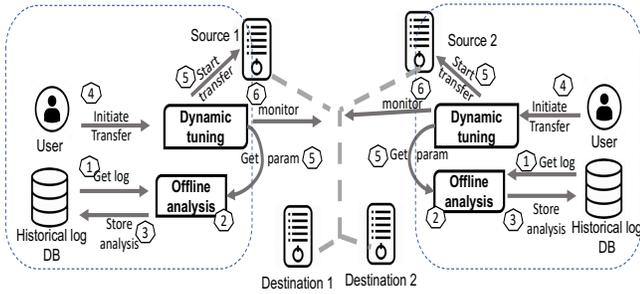


Figure 2: Distributed approach overview.

historical data transfer logs as input and solves the SLA based optimization problem beforehand. We choose this strategy so that optimization does not introduce latency during the actual transfer. The cost of offline optimization can be amortized over many subsequent transfers. As we have explained in Section II different SLA requirements have different optimization objective. However, all of them are actually looking for optimal application-level parameters. In a shared environment, where dynamic fluctuation of traffic is very common, the static parameters from the offline analysis may become sub-optimal during the transfer. A real-time tuning is necessary to cope with such fluctuation as we have to maintain strict SLA constraint. An overview of the model is depicted in Figure 2.

1) *Offline Optimization*: The Offline optimization can be broken down into parts: (1) storing historical logs, (2) external load modeling, (3) clustering and data interpolation, (5) constrained optimization, and (6) energy modeling.

**Step 1 – Storing Historical Logs**: Historical data transfer logs are collected periodically during the transfer and stored in a log server. These logs collect information about the network characteristics (e.g., round trip time, buffer size, queuing delay, packet loss rate), application level parameters ( $cc$ ,  $p$ ,  $pp$ ), end system resource information (e.g., CPU, memory, NIC), data set information (e.g., size, number of files/objects), energy-related information (e.g., CPU utilization, memory utilization, NIC card utilization, disk I/O utilization). These logs provide insight to optimize transfers energy-efficiently under different circumstances. Still, logs are prone to errors (e.g., recorded achieved throughput may be greater than the actual link bandwidth) and may have missing values. Standard data interpolation techniques are used to predict those values. A preprocessing module takes care of such case.

**Step 2 – External Load Modeling**: In a shared network environment, data transfer task has to compete with other contending transfers. Contending transfers can be a mixture of known incoming/outgoing transfers in both source/destination and completely unknown transfers. Some bandwidth may be wasted by TCP congestion and slow start. However, the optimal number of streams can offset slow start and congestion loss significantly. We can model the achievable throughput of

a data transfer as:

$$T_{act} = BW - \sum T_{ext\_known} - \sum T_{ext\_unknown} - \sum \delta_{slow\_start} - \sum \delta_{congestion} \quad (7)$$

As we have periodically collected logs, it is easy to estimate the  $\delta_{slow\_start}$  and  $\delta_{congestion}$  just analyzing the achieved throughput of the subsequent time intervals. We can also estimate the combined throughput of known transfers. Therefore, from Equation (7), we can get a rough estimate of the combined throughput of the unknown external traffic.

**Step 3 – Clustering and Interpolation**: Similar types of transfers can be optimized using similar parameter combinations. Categorizing logs into groups based on their similarity could provide us a more structured view of the log information. After analyzing the logs we come to the conclusion that some parameters have direct precedence over other parameters. We use *Hierarchical Agglomerative Clustering* [42] which is the most suitable clustering technique for such cases.

We are interested to find the optimal parameters under different external traffic load. In Step-2 we get a rough estimate of external traffic. Our experiment shows that achievable throughput,  $T_{act}$  and energy consumption,  $E_{act}$  can be directly impacted by the application level parameters,  $\theta[:]=\{cc, p, pp\}$ . The relation is strictly non-linear and follows a continuous cubic pattern. Therefore, we modeled both throughput and energy using piece-wise cubic spline interpolation (Figure 1). This technique stitches multiple cubic functions with smoothness guarantee up to the second derivative. All the continuity constraints and the smoothness constraints are linear. Therefore, the coefficients can be computed by solving the system of linear equations. Interpolants can be written for each cluster  $c_i$  as:

$$\begin{aligned} T_{c_i} &= I_1(p, cc, pp, bs) \\ E_{c_i} &= I_2(p, cc, pp, bs) \end{aligned} \quad (8)$$

**Step 4 – Constrained Optimization**: An overview of constrained optimization for different types of SLA is explained in Section II. Running those optimizations in real-time may add extra latency during the transfer. One may argue that the real-time optimization can be performed concurrently with the transfer. Still, the transfer would run under sub-optimal solution until optimization finishes. Pre-computing these optimizations during offline phase can have two major benefits: (1) it eliminates any real-time latency for optimal parameters, and (2) these precomputed results can be reused for many subsequent transfers, which can effectively amortize the initial cost of analysis. As the user can put constraints over throughput or energy or instant power consumption, during offline phase we would have to solve the optimization problem for all possible SLA values of throughput or energy or power, which is not feasible. We solve this issue by intelligently analyzing the historical logs so that we can eliminate infeasible SLA constraint values. For example, to perform a transfer we need

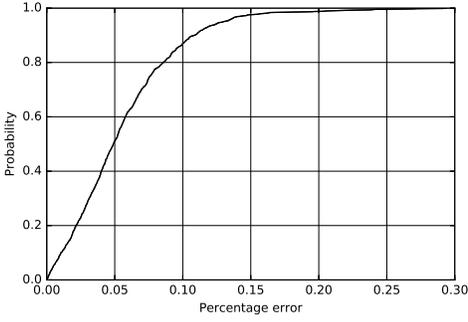


Figure 3: Regression error CDF of power model.

a minimum level of energy consumption, therefore, any SLA value below the threshold is infeasible. We can set a feasible region of SLA values. We also observed that, when two SLA values are close, they produce similar solutions. Therefore, instead of solving the optimization problem for all possible SLA values we partition the SLA feasible region based on historical log data and solve one optimization problem for each partition. This approach can give us a discrete number of SLA levels for both throughput ( $k_{th\_level}$ ), energy ( $k_e\_level$ ), and power consumption ( $k_p\_level$ ).

#### Step 5 – Energy model:

When multiple server processes are involved, the total energy consumption of the end system can be estimated as the summation of energy consumption of the concurrent server processes,  $e_i$  (energy consumption of  $i$ -th process).

$$E_\tau = \sum_{i=1}^{cc} e_i(\mu_{cpu,i,\tau}, \mu_{mem,i,\tau}, \mu_{disk,i,\tau}, \mu_{nic,i,\tau}) \quad (9)$$

In order to estimate the energy consumption of the end systems during the data transfers, we built a linear regression model with the following input features: CPU utilization, memory usage, number of disk reads and writes, number of bytes read and written to disk, number of bytes sent and received over the network, and number of packets sent and received. We collected 4467 samples by monitoring a workstation under different CPU and network loads, and measured the actual power consumption of the machine using a Yokogawa WT210 power meter. We used 70% of the samples for training and 30% for testing. Figure 3 shows the CDF of the prediction error for the test set. As the figure shows, most predictions ( $\geq 90\%$ ) suffer from a very small error ( $\leq 10\%$ ).

2) *Dynamic Tuning*: Dynamic tuning is the heart of the protocol. This is the real-time monitoring of the health of the data transfers, simultaneously it controls the aggressiveness of the protocol (fairness constraint), while ensuring strict SLA requirement. As we have three different categories of SLAs, we need three different strategies as well. However, the core control is mostly similar. An overview of the tuning module

---

#### Algorithm 1: Dynamic Tuning (Distributed)

---

```

// Expected Energy Efficiency,
 $\eta = Data_{total}/E_{sla}$ ; Queuing Delay,  $Q_{rtt}$ ;
number of streams,  $S = cc \times p$ 
1  $SLA\_type \leftarrow translate\_SLA(SLA)$ 
2  $\theta_{initial} \leftarrow get\_params(req, TYPE = median)$ 
3  $update\_resource\_groups(E_{sla},$ 
    $TYPE = median)$ 
4  $data\_transfer(req, \theta_{initial})$ 
5 Periodically check:
6 if  $SLA\_type == 'Energy Constraint'$  then
7   if  $Data_{curr}/E_{curr} \leq \eta$  then
8      $E_{left} \leftarrow E_{sla} - E_{curr}$ 
9     if  $T_{curr} \leq T_{pred} - \epsilon$ :  $back\_off\_control()$ 
10     $\theta_{new} \leftarrow get\_params(Q_{rtt}, th_{prev}[i:j], E_{left})$ 
11  else
12     $increase(cc\_limit, p\_limit, \alpha_{cc}, \alpha_p)$ 
13     $opportunistic\_decrease(E_{left})$ 
14     $\theta_{new} \leftarrow get\_params(Q_{rtt}, th_{prev}[i:j], E_{left})$ 
15  end
16 else if  $SLA\_type == 'Throughput Guarantee'$  then
17   if  $T_{curr} \neq T_{sla} \pm \epsilon$  then
18      $T_{goal} \leftarrow T_{sla} + (T_{sla} - T_{curr})$ 
19     if  $T_{curr} \leq T_{sla} - \epsilon$ :  $back\_off\_control()$ 
20     else:  $increase(cc\_limit, p\_limit, \alpha_{cc}, \alpha_p)$ 
21      $opportunistic\_increase(T_{goal})$ 
22      $\theta_{new} \leftarrow get\_params(Q_{rtt}, th_{prev}[i:j], T_{goal})$ 
23   end
24 else if  $SLA\_type == 'Power Constraint'$  then
25    $update\_resource\_groups(\varphi_{sla},$ 
    $TYPE = fixed)$ 
26   if  $T_{curr} \leq T_{pred} - \epsilon$  then
27      $T_{goal} \leftarrow T_{pred} + (T_{pred} - T_{curr})$ 
28      $back\_off\_control()$ 
29   else
30      $increase(cc\_limit, p\_limit, T_{goal})$ 
31   end
32    $\theta_{new} \leftarrow get\_params(Q_{rtt}, th_{prev}[i:j], T_{left})$ 
33    $\mu[1:cc][:] \leftarrow get\_resource\_utilization()$ 
34    $update\_resource\_groups(E_{left})$ 
35 Periodically call:
36  $SSet\_low, th[1:S] \leftarrow check\_stream\_perf(cc, p)$ 
37  $redistribute\_pipelining(SSet\_low,$ 
    $th[1:S], cc, p)$ 
38 if required:  $update\_resource\_groups(E_{left})$ 

```

---

is introduced in Algorithm 1. To achieve energy efficiency we need to restrict the resource utilization of the transfer processes. We used cgroup to predefine some resource groups where resource utilization can be restricted up to defined levels. Transfer processes can be assigned to those resource group in real-time. Large-scale transfers take a long time, therefore, network load fluctuation is a reality. During the transfer, there may be some time-interval when the network becomes congested due to newly initiated external transfers and achievable throughput may drop up to a certain level. To maintain fairness, we introduce a back-off control to reduce parameters (Algorithm 2). When queuing delay drops for a certain amount of time, it reduces parallelism level by predefined  $\beta_1$  (Line 4-6). When it detects a significant

**Algorithm 2: Stream Back off Control Algorithm**

```

1 procedure back_off_control()
2    $Q_{rtt} \leftarrow \text{measure\_queuing\_delay}()$ 
3    $\text{pkt\_loss\_rate} \leftarrow \text{get\_packet\_loss\_rate}()$ 
4   if  $Q_{rtt} \ll Q_{rtt}.\text{expected}$  then
5     reduce( $p\_limit, \beta_1$ )
6   end
7   if  $\text{pkt\_loss\_rate} \ll \text{pkt\_loss\_rate}.\text{threshold}$  then
8     reduce( $cc\_limit, \beta_2$ )
9   end

```

**Algorithm 3: Centralized Scheduling**

**input :** Transfer request,  $\text{req} = \{\text{src}, \text{dest}, \text{SLA}\}$ ,  
Link information,  $L[1 : \text{all}]$ , where,  
 $L[i] = \{\text{BW}, \text{RTT}, T_{\text{exist}}\}$

```

1 Periodically receive:
2 network_view()
3 ledger  $\leftarrow$  existing_transfer_status()
4 if request, req is received: Queue.put(req)
5 while Queue is not empty do
6    $T_{\text{ext}} \leftarrow \text{compute\_external\_load}(\text{ledger})$ 
7   if req.SLA == 'Type-1' then
8      $E_{\text{exp}}, \theta \leftarrow \text{get\_params}(\text{Link}, T_{\text{ext}}, T_{\text{sla}})$ 
9   else if req.SLA == 'Type-2' then
10     $T_{\text{exp}}, \theta \leftarrow \text{get\_params}(\text{Link}, T_{\text{ext}}, E_{\text{sla}})$ 
11  else
12     $T_{\text{exp}}, \theta \leftarrow \text{get\_params}(\text{Link}, T_{\text{ext}}, \varphi_{\text{sla}})$ 
13  end
14  send_parameters( $\theta$ )
15 end

```

**Algorithm 4: Centralized Micro-Tuning**

```

input : Periodic updates,
         $U[1 : \text{all}] = \{EP_s, EP_d, T, E, Q_{rtt}, \text{Status}\}$ 
1 if link capacity reduced due to failure/maintenance then
2   scale_down_params(..., BW=new_capacity)
3 end
4 for  $u_i$  in  $U[1 : \text{all}]$  do
5   if  $u[\text{status}] == \text{'FINISHED' or 'ABORTED'}$  then
6     redistribute_params()
7   end
8   if  $u[\text{status}] == \text{'SLA violation'}$  then
9     micro_tune()
10  end
11 end

```

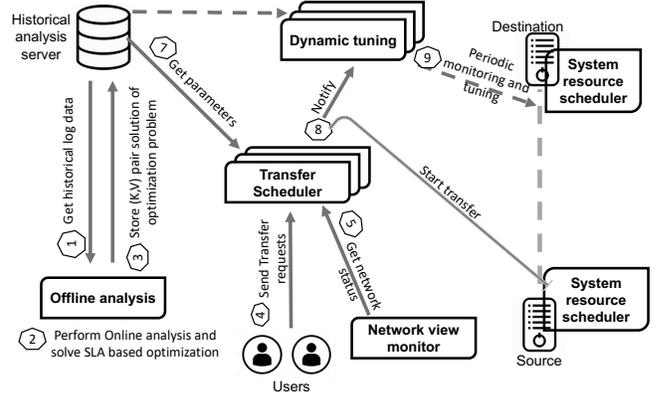


Figure 4: Centralized approach overview.

increase in packet loss rate, it reduces the concurrency level by predefined  $\beta_2$  (Line 7-9). During these time intervals achieved throughput may go below the throughput mentioned in SLA, which must be compensated to guarantee the SLA requirement. We introduce an opportunistic strategy that is - whenever possible (time intervals with low external load) target for a solution better than SLA requirement so that we have enough buffer performance to cover the inevitable throughput drop during external load spike. Dynamic tuning (Algorithm 1) for different SLAs are explained below.

**Constraint 1 – Throughput Guarantee:** (Line 16-23) It periodically checks the throughput,  $T_{act}$  and checks whether the value is outside a tolerance bound  $T_{sla} \pm \epsilon$ . When it is below the bound, dynamic tuning initiates back-off control (Line 17-19), otherwise, it initiates `opportunistic_increase()` module. In an uncongested link, it can provide continuous throughput guarantee, however, a congested link may force the transfer to reduce the parameters (ensure fairness) that directly impact the throughput.

`opportunistic_increase()` compensates this inevitable throughput drop with a new throughput goal  $T_{goal}$  that is higher than  $T_{sla}$  (Line 20-21). During uncongested time interval dynamic tuning module will ask offline analysis module to provide parameters that can achieve  $T_{goal}$  (Line 22). We also introduce a buffer capacity of achievable throughput higher than  $T_{sla}$ . This buffer size is based on historical data

analysis. Opportunistic increase function tries to fill this buffer, whenever it senses available throughput. This buffer proactively offsets any future throughput degradation.

**Constraint 2 – Energy Constraint:** (Line 6-15) We have used a similar strategy for the energy constrained tuning as well. Here we are mostly interested in energy efficiency of the transfer,  $\eta = \text{Data}_{total}/E_{total}$ . This is the expected efficiency to maintain the energy constraint as mentioned in SLA. When the current efficiency goes below the expected efficiency, that means the transfer is using more energy to transfer unit amount of data than expected. It could be due to the congestion in the link where retransmission and waiting for acknowledgment can reduce  $\eta$  significantly. We can easily check whether this is the case, just by measuring current throughput and queuing delay of the network and initiate the back-off control to reduce the parameters (Line 7-10). Another reason could be the unnecessary use of application-level parameters. Redundant concurrency levels could open more server processes which would lead to extra energy consumption. In such case, we ask offline analysis to provide new parameters. We also introduced opportunistic decrease of energy consumption whenever possible to offset the extra energy consumption happened due to suboptimal parameters

Specifications	IBM IDCN	XSEDE
Bandwidth (GBps)	1	10
RTT (ms)	65	40
Buffer size (MB)	8	32
File system		Lustre
Cores	2	16
Memory (GB)	2	32

Table II: System and network specification of test sites

or congestion. It can be done by switching processes to more energy constraint resource groups.

**Constraint 3 – Power Constraint:** (Line 24-32) This constraint asks to limit instantaneous power consumption of the transfer. However, in case of heavy congestion, it reduces the parameters. In a congestion-free network, it tries to increase the concurrency without violating the instantaneous power consumption limit.

### B. Centralized Approach

The main advantage of centralized scheduling is the scheduler has a global view of the network status and overall transfer load. Therefore, the external load can be estimated more precisely. In distributed scheduling, there may be parameter over-shoot and under-shoot before all of them converge to an optimal level. However, this oscillation can be reduced in centralized approach, as the scheduler has global network view and all transfer periodically send status to the scheduler. An overview of the centralized approach is shown in Figure 4. The centralized approach has three components: (1) offline analysis, (2) transfer scheduling, and (3) dynamic tuning. Transfer scheduler performs the offline historical analysis similar to distributed approach and pre-computes the optimization problems to use them during real-time transfers.

After explaining away the external load, the available link bandwidth should not exceed the combined throughput guarantee of Type-1 SLA and combined predicted throughput of Type-2 SLA and Type-3 SLA. This module can work as an integration in Traffic Engineering (TE) module of SDN. To make the solution more scalable, we can delegate controls in a hierarchical manner.

1) *Centralized Transfer Scheduler:* Initially, the scheduler (Algorithm 3) clusters all the transfer requests based on source, destination, and their SLA requirements. Then it aggregates SLA requirements of each cluster. As we can see, each of the SLA groups in a single link is actually the external load for one another. The scheduler receives periodic updates from the participating transfers and has a more precise knowledge about the parameter distribution. It periodically updates external load estimation (Line 6). Therefore, the centralized approach can ask offline analysis module for parameters with precise external load (Line 6-13), unlike distributed approach that starts with parameters for median external load and then converges.

The centralized scheduler also performs micro tuning periodically when necessary (Algorithm 4). In case of link capacity reduction due to failure or maintenance, scheduler scales down the parameters for all contending transfers. When a transfer finishes or is aborted, it notifies the scheduler so that it can redistribute the newly released parameters to existing transfers without violating energy and power constraints. In case of SLA violation, it redistributes the parameters among the contending transfer using more simpler `micro_tune()` routine.

## V. EVALUATION

We performed experiments on a wide-area network link between IBM datacenters located in Washington, D.C. and San Jose, CA. We also used XSEDE, a production level high-speed computing infrastructure for large-scale scientific computations. An overview of systems and network information is provided in Table II.

We compared our model with many existing data transfer solutions. However, there has been done very little work on energy efficient data transfer optimization. We evaluate our model against the model proposed by Alan et. al. [2], `globus-url-copy` (`guc`) [19], Globus Online (GO) [14], `scp`, SFTP, Rsync, Rclone [48], and CloudFuse [15].

Alan et. al. provide a High Throughput Energy-Efficient Transfer Algorithm (HTEE) that uses heuristics based approach to balancing the achieved throughput and energy consumption. It starts with one channel and periodically increases it by 2 until it reaches to a user-defined limit. Then it computes ( $T_{act}/E_{act}$ ) ratio for each level and picks the best one. This periodic additive increase is slow and keeps the transfer sub-optimal until it searches the whole parameter space which is still  $O(n)$  when the user-defined value is  $n$ . `globus-url-copy` and Globus Online are GridFTP based data transfer tools to achieve high performance during transfer. However, they are not energy optimized tools. `Scp` and SFTP are widely used secure file transfer tools. Rsync and Rclone are high-performance data synchronization applications. CloudFuse provides cloud-based Managed File Transfer (MFT) service and offers migration, sync and other file management capabilities to the end users.

### A. Comparison with Other Solutions

Figure 5 shows an elaborate experimentation and performance analysis of different state-of-the-art solutions and our proposed approach. In literature, very little work is done to optimize both throughput and energy of a data transfer with SLA specifications. Most of the models do not support SLA. Therefore, to make comparison fair we set the SLA of our model in two extreme cases - (1) Maximum achievable throughput (MaxTh) and (2) Minimum possible energy consumption (MinPow). To test the efficiency of different types of file transfers we tested all data transfer solutions for small (1 - 5MB), medium (100 - 500MB), and large (1 - 4GB) files.

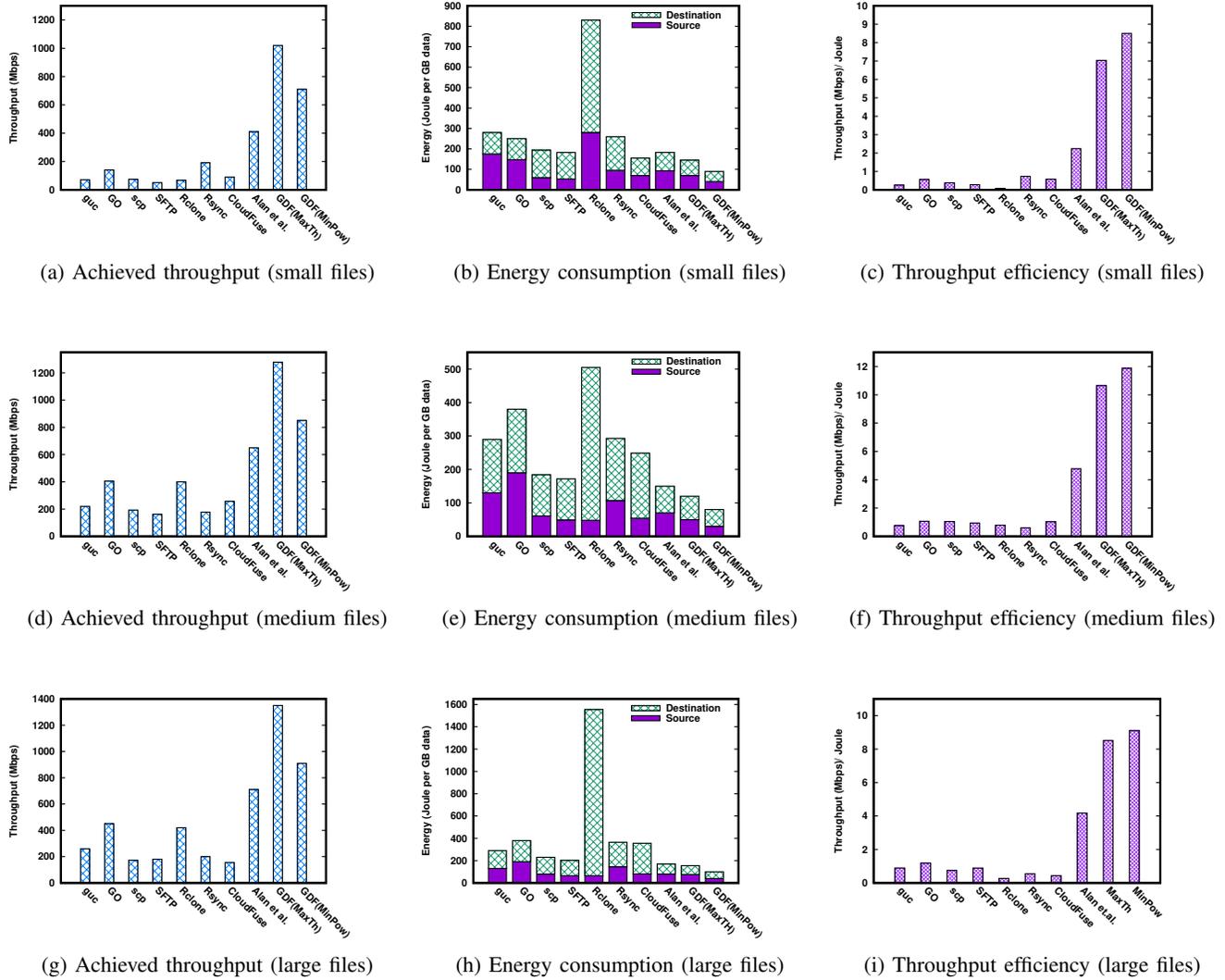


Figure 5: Achievable throughput and corresponding energy consumption of different optimization objectives.

1) *Performance of Medium File Transfers:* Figure 5 (d-f) contain performance comparison for medium files. We compared achieved throughput, energy consumption, and the throughput efficiency,  $(T_{act}/E_{act})$ . To accelerate the medium file transfer, a moderate choice of concurrency  $cc$  and parallelism  $p$  is helpful in an uncongested link. Multiple files with multiple segments can be transferred simultaneously. However, an initial best-known parameter choice along with dynamic tuning and effective end-system resource scheduling can increase energy efficiency as well.

As we can see off-the-shelf tools like `scp` and `SFTP` perform poorly due to single data channel allocation and also the control channel inefficiency in long RTT WAN. Their energy consumption is also high because low throughput transfer needs more time to finish and even though resource

utilization is low, longer time of execution increases the static power component (power consumption when the resource is idle and waiting). Similarly, `globus-url-copy` (`guc`), a GridFTP based tool, also performs poorly with base-line parameter settings ( $cc = 1$  &  $p = 1$ ).

GridFTP is designed for multi-threaded transfers and more resource intensive than `scp` and `SFTP` when used with a single channel. So, it suffers from low throughput while consuming more energy. On the other hand, Globus Online is a statically tuned cloud service that uses GridFTP protocol. Due to the use of multiple streams ( $cc \times p > 1$ ), we observe that it can reach up to  $2\times$  performance improvement compare to `scp`, `SFTP`, and `guc`. However, it consumes  $2\times$  more energy. The reason is that statically assigned parameters may not be optimal for all external traffic levels. There is no way

to limit the resource utilization as well.

`Rclone` and `Rsync` are file synchronization tools. By default, `Rclone` uses 4 parallel data connections to transfer a single file. We observe that it can achieve similar performance as `GO`, however, there is no way to do any dynamic adjustment to parallel connections. We see an unusually high energy consumption at the destination. It may be due to the extra work it has to do for sync operation for four parallel connections. On the other hand, `Rsync` performance is lower than `Rclone` as it does not use any parallel connections for file transfer, therefore, under-utilizes the available bandwidth. However, it consumes less energy due to its single connection syncing.

`CloudFuse` achieves slightly better performance than `guc`. It consumes slightly less energy compared to `Rsync` and `guc`. As we see in the Figure, Alan et. al. model performs much better than solutions discussed above due to the fact that it performs an on-line parameter search and after deciding on the best parameter, it transfers rest of the data efficiently. However, there is no real-time control on parameters, therefore, when external traffic changes those parameters may become sub-optimal. And additive parameter search may take a toll on the achieved throughput. However, due to the search for energy-efficient parameters, it can manage to keep energy consumption less than other approaches mentioned above.

Both of `GreenDataFlow` algorithms (`MaxTh` and `MinPow`) outperform all the listed solutions. `MaxTh` provides  $6\times$  throughput performance improvement over the baseline performance of `globus-url-copy` and almost  $2\times$  improvement over the closest competitor Alan et al. model due to the historical analysis and real-time tuning of the parameters. As it achieves high throughput, the execution time reduces as well which reduces the static power consumption along with constrained resource scheduling in end-systems. `MinPow` is aimed to decrease the total energy consumption. It consumes  $8\times$  less energy than the energy-hungry `rclone`,  $3\times$  less energy than base-line `guc` and almost 36% less energy than the closest competitor Alan et. al.

2) *Performance of Small and Large File Transfers:* Disk-to-disk small file transfers are very challenging, as we have to perform a lot of disk read/write operations. Without a proper pipelining level, control channel idleness can introduce delay among subsequent file transfers. As we can see in Figure 5 (a-c), overall achieved throughput for all approaches are lower than the achieved throughput of medium (Figure 5(d)) and large files (Figure 5(g)). However, the achievable throughput difference is quite similar to medium file transfers except the cases where `Rclone` performs worst than `Rsync` and `CloudFuse`. Alan et al. model achieves low throughput compared to medium files, because small file transfers suffer badly for sub-optimal parameter choices. Therefore, additive parameter search takes more toll on small file transfers compared to the medium files. Our `MaxTh` model reaches  $2.5\times$  performance increase compared to the closest competitor Alan et al. model, and our energy optimized `MinPow` consumes  $2\times$  less energy compared to it.

Parallelism is the most important parameter for large file

transfers as we want to parallelize multiple segments of a large file. Concurrency can add extra boost on performance, however, a very high value can over-burden the network. Figure 5 (g-i) shows the performance of large file transfers. It can be seen that the overall throughput performance is better than medium file transfers. However, the performance pattern for `guc`, `GO`, `SCP`, `SFTP`, `rsync`, `CloudFuse` is roughly similar to medium files because of the use of fixed parameter settings. Energy consumption performance is also similar to medium files except for a huge spike in `rclone` destination, as the file size grows the energy consumption increases rapidly. Alan et al. model also performs better compare to its performance on small and medium files. Our model outperforms Alan et al. model and achieves  $2\times$  performance boost in throughput. On the other hand, our `MinPow` model, consumes 70% less energy compare to Alan et al. model.

### B. SLA-based Performance Analysis

We have discretized the SLA levels of throughput, energy and instant power and eliminated the infeasible regions. Figure 6 shows the performance for different SLA levels. It can be seen that SLA violations are rare unless there exist over-subscription of throughput or severe capacity reduction for a long period of time. Most of the cases in Type-T SLA (Figure 6 (a-c)), our model can achieve performance over  $T_{sla}$ , due to the `opportunistic_increase` strategy. It also keeps the resource utilization manageable by putting dynamic restriction on usage. SLA violation error is ranged from 3% to 6%. For energy constrained (Type-E) SLA, we observed the SLA violation occurs due to heavy congestion which forces retransmission and initiates slow start phase. Moreover, excessive concurrent processes can consume extra power while congesting the network. As our model cautiously monitors and budgets the required future energy usage, it can achieve high accuracy in SLA commitment. Instant power consumption constraint forces the transfer to be assigned in a restricted resource group and never changes it to guarantee this constraint, however, it may fix parameters in real-time to achieve the expected throughput. This constraint produces some interesting results, as we can see, it consumes more total energy to achieve a throughput similar to Type-T and Type-E SLAs. Due to the strict resource constraint, transfer may take long time to finish that leads to more static power consumption.

### C. Protocol Fairness Analysis

Both of our distributed and centralized approaches are designed to maintain fairness among the contending transfers while maximizing the overall WAN utilization. We have tested our model in 10 Gbps XSEDE WAN with four contending transfers. Figure 7 shows the performance of different transfer approaches. We can see that `scp` can achieve throughput around 500 Mbps and all contending users can get an equal share, however, the network utilization is very low. `Rsync` also gets a similar performance with a good fairness among the users. This is due to the single channel that is not enough

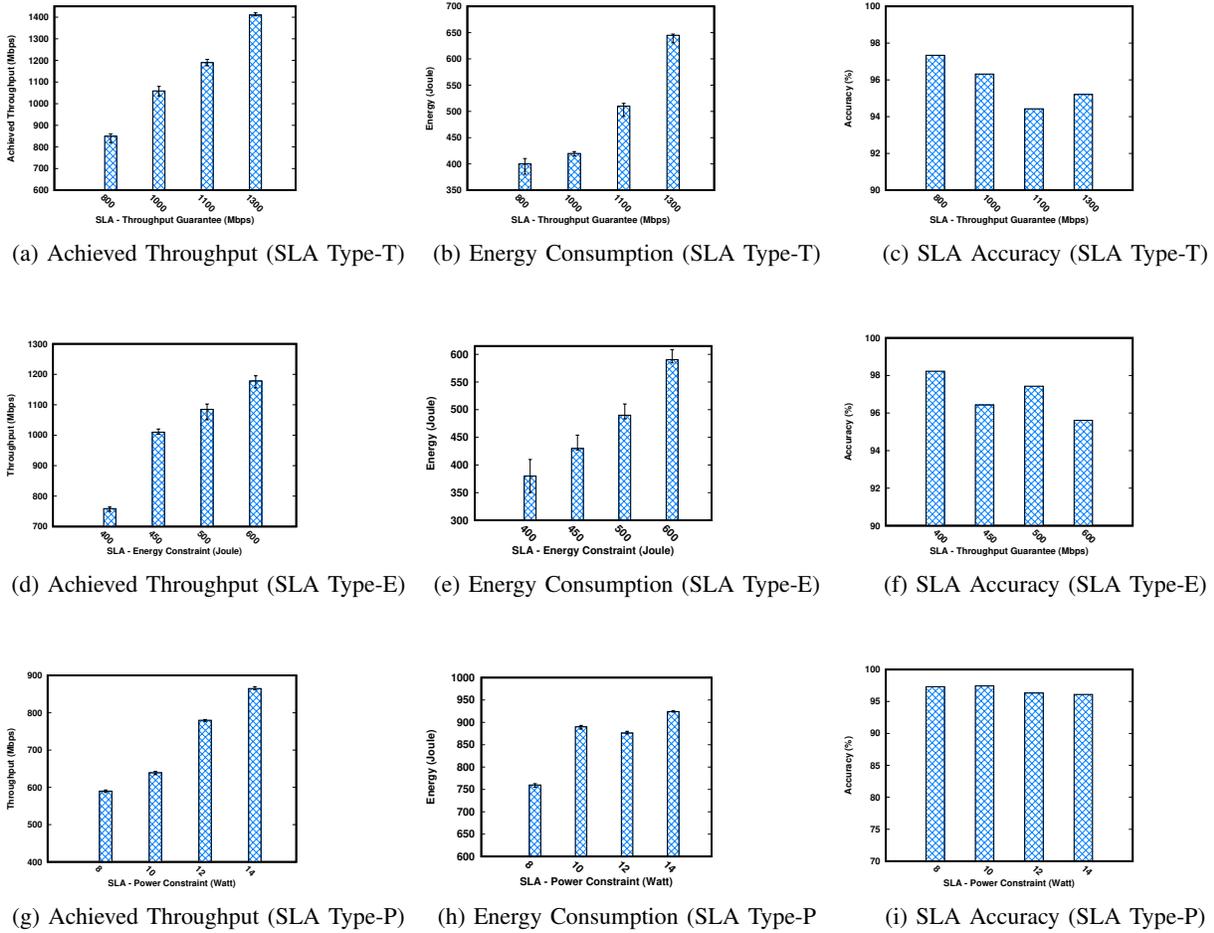


Figure 6: Achieved throughput, energy consumption and SLA commitment accuracy for different SLA types. (a-c) SLA with throughput guarantee, (d-f) SLA with energy constraint, and (g-i) SLA with power constraint.

to fill up the 10 Gbps WAN link. Both of them achieve 20% of the network utilization. Alan et al. model achieves higher throughput and the network utilization is almost 40%, however, we can see the performance is not fairly distributed among the users. If all the transfer start at the same time, then Alan et al. model can get a fair share. If external traffic changes during the search process, then it can choose parameters unfairly. Both the distributed and centralized models provide superior utilization of the network. The distributed approach can achieve almost 82% network utilization where centralized approach can reach up to 90% utilization. In the distributed approach, the users need to sense the network periodically and waste some throughput while converging. However, centralized approach converges faster as it knows about other contending transfers and estimates external load more precisely. Due to the proper back-off control, it can become less aggressive towards the other contending transfers.

## VI. RELATED WORK

The work on network throughput optimization focuses on tuning transfer parameters such as parallelism, pipelining, concurrency and buffer size. The first attempts to improve the data transfer throughput at the application layer were made through buffer size tuning. Various dynamic and static methods were proposed to optimize the buffer size [32], [45], [50]. However, Lu et al. [40] showed that parallel streams can achieve a better throughput than buffer size tuning and then several others [5], [29], [56], [55] proposed throughput optimization solutions by means of tuning parallel streams. Another transfer parameter used for throughput optimization was pipelining, which helped in improving the performance of transferring large number of small files [20], [18], [11], [54]. Liu et al. [39] optimized network throughput by concurrently opening multiple transfer sessions and transferring multiple files concurrently. They proposed increasing the number of concurrent data transfer channels until the network perfor-

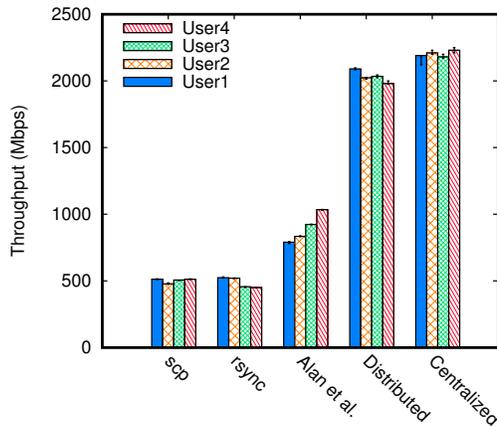


Figure 7: Fairness analysis of different approaches.

mance degrades. Globus Online [4] offers fire-and-forget file transfers through thin clients over the Internet. It partitions files based on file size and transfer each partition using partition-specific protocol parameters. However, the protocol tuning Globus Online performs is non-adaptive; it does not change depending on network conditions and transfer performance.

The work on power-aware networking focuses on saving energy at the networking devices. Gupta et al. [27] were among the earliest researchers to advocate conserving energy in networks. They suggested different techniques such as putting idle sub-components (i.e. line cards, etc.) to sleep [26], which were later extended by other researchers. S. Nedeovski et al. [43] proposed adapting the rate at which switches forward packets depending on the traffic load. IEEE Energy Efficient Ethernet task force proposed the 802.3az standards [1] for making ethernet cards more energy efficient. They defined a new power state called low power idle (LPI) that puts the ethernet card to low power mode when there is no network traffic. Other related research in power-aware networking has focused on architectures with programmable switches [25] and switching layers that can incorporate different policies [33]. Barford et al. proposed power-aware network protocols for energy-efficiency in network design and routing [13]. Bertozzi et al. [10] investigated the energy trade-off in networking as a function of the TCP receive buffer size and show that the TCP buffering mechanisms can be exploited to significantly increase energy efficiency of the transport layer with minimum performance overheads.

Several highly-accurate scheduling algorithms [37], [7], [8] and predictive models [9], [58], [57], [35], [34] were developed which require as few as three sampling points to provide very accurate predictions for the parallel stream number giving the highest transfer throughput for the wired networks. Yildirim et al. analyzed the combined effect of parallelism and concurrency on data transfer throughput [53]. Alan et al. analyzed the effects of parallelism and concurrency

on end-to-end data transfer throughput versus total energy consumption in wide-area wired networks using precalculated values for these parameters and proposed a heuristic approach to improve them [2], [3].

## VII. CONCLUSION

In this paper, we introduced a novel set of data transfer algorithms (collectively called GrenDataFlow) based on historical analysis and real-time tuning, which can achieve high data transfer throughput while keeping the energy consumption during the transfers at the minimal levels. GreenDataFlow supports service level agreements (SLAs) which give the service providers and the consumers the ability to fine tune their goals in this optimization process. Our experimental results show that GreenDataFlow outperforms existing solutions in this area both in terms of energy saving and the achieved end-to-end performance. Considering the massive energy footprint of global data movement, our presented GreenDataFlow techniques have a great potential to decrease this footprint and contribute to the efforts of achieving greener Internet.

## ACKNOWLEDGEMENTS

This project is in part sponsored by the National Science Foundation (NSF) under award numbers OAC-1724898 and OAC-1842054, and by IBM under award number ocrw1771224.

## REFERENCES

- [1] IEEE energy efficient ethernet standards. 10.1109/IEEEESTD.2010.5621025, Oct. 2010.
- [2] I. Alan, E. Arslan, and T. Kosar. Power-aware data scheduling algorithms. In *Proceedings of IEEE/ACM Supercomputing Conference (SC15)*, November 2015.
- [3] I. Alan, E. Arslan, and T. Kosar. Energy-Performance Trade-offs in Data Transfer Tuning at the End-Systems. *Sustainable Computing: Informatics and Systems Journal*, Under Review, 2014.
- [4] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke. Software as a service for data scientists. *Communications of the ACM*, 55:2:81–88, 2012.
- [5] E. Altman and D. Barman. Parallel tcp sockets: Simple model, throughput and validation. In *Proceedings of IEEE INFOCOM*, 2006.
- [6] G. Ananthanarayanan and R. Katz. Greening the switch. In *In Proceedings of HotPower, December 2008*.
- [7] E. M. Bahsi, E. Ceyhan, and T. Kosar. Conditional workflow management: A survey and analysis. *Scientific Programming*, 15(4):283–297, 2007.
- [8] M. Balman and T. Kosar. Data scheduling for large scale distributed applications. In *the 5th ICEIS Doctoral Consortium, In conjunction with the International Conference on Enterprise Information Systems (ICEIS'07). Funchal, Madeira-Portugal*. Citeseer, 2007.
- [9] M. Balman and T. Kosar. Dynamic adaptation of parallelism level in data transfer scheduling. In *International Conference on Complex, Intelligent and Software Intensive Systems*, pages 872–877. IEEE, 2009.
- [10] D. Bertozzi, A. Raghunathan, L. Benini, and S. Ravi. Transport protocol optimization for energy efficient wireless embedded systems. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, page 10706. IEEE Computer Society, 2003.
- [11] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster. Gridftp pipelining. In *Proceedings of TeraGrid*, 2007.
- [12] D. Brooks, V. Tiwari, and M. Martonosi. Watch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 83–94, New York, NY, USA, 2000. ACM.

- [13] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsang, and S. Wright. Power awareness in network design and routing. In *In Proceedings of IEEE INFOCOM, April, 2008*.
- [14] K. Chard, I. Foster, and S. Tuecke. Globus: Research data management as service and platform. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, page 26. ACM, 2017.
- [15] Why cloudfuze: Quickly connect with powerful providers like google drive, dropbox, or box from a single screen and login. <https://www.cloudfuze.com/why-cloudfuze/>, 2016.
- [16] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proc. of Workshop on Modeling, Benchmarking, and Simulation*, 2006.
- [17] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News*, 35(2):13–23, 2007.
- [18] K. Farkas, P. Huang, B. Krishnamurthy, Y. Zhang, and J. Padhye. Impact of tcp variants on http performance. *Proceedings of High Speed Networking*, 2, 2002.
- [19] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
- [20] N. Freed. SMTP service extension for command pipelining. <http://tools.ietf.org/html/rfc2920>.
- [21] W. Fu and T. Song. A frequency adjustment architecture for energy efficient router. *ACM SIGCOMM Computer Communication Review*, 42(4):107–108, 2012.
- [22] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. It's not easy being green. *ACM SIGCOMM Computer Communication Review*, 42(4):211–222, 2012.
- [23] E. Goma, M. C. A. L. Toledo, N. Laoutaris, D. Kosti, P. Rodriguez, R. Stanojev, and P. Y. Valentin. Insomnia in the access or how to curb access network related energy consumption. In *In Proceedings of ACM SIGCOMM 2011*.
- [24] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. In *In ACM SIGCOMM CCR, January 2009*.
- [25] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: Scalability and commoditization. In *In ACM PRESTO, pages 5762, 2008*.
- [26] M. Gupta and S. Singh. Energy conservation with low power modes in ethernet lan environments. In *IEEE INFOCOM (MiniSymposium) 2007*.
- [27] M. Gupta and S. Singh. Greening of the internet. In *ACM SIGCOMM, pages 1926, 2003*.
- [28] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir. Using complete machine simulation for software power estimation: The softwatt approach. In *Prpc. of 8th High-Performance Computer Architecture Symp.*, pages 141–150, 2002.
- [29] T. J. Hacker, B. D. Noble, and B. D. Atley. Adaptive data block scheduling for parallel streams. In *Proceedings of HPDC '05*, pages 265–275. ACM/IEEE, July 2005.
- [30] K. Hasebe, T. Niwa, A. Sugiki, and K. Kato. Power-saving in large-scale storage systems with data migration. In *IEEE CloudCom 2010*.
- [31] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proceedings of NSDI 2010*.
- [32] M. Jain, R. S. Prasad, and C. Dovrolis. The tcp bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing. 2003.
- [33] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *SIGCOMM CCR 38(4):5162, 2008*.
- [34] J. Kim, E. Yildirim, and T. Kosar. A highly-accurate and low-overhead prediction model for transfer throughput optimization. *Cluster Computing*, 18(1):41–59, 2015.
- [35] J. Kim, E. Yildirim, and T. Kosar. A highly-accurate and low-overhead prediction model for transfer throughput optimization. In *Proc. of DISCS Workshop*, November 2012.
- [36] R. Koller, A. Verma, and A. Neogi. Wattapp: an application aware power meter for shared data centers. In *Proceedings of the 7th international conference on Autonomic computing*, pages 31–40. ACM, 2010.
- [37] T. Kosar. *Data Placement in Widely Distributed Systems*. PhD thesis, University of Wisconsin–Madison, 2005.
- [38] W. Liu, B. Tieman, R. Kettimuthu, and I. Foster. A data transfer framework for large-scale science experiments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10*, pages 717–724, New York, NY, USA, 2010. ACM.
- [39] W. Liu, B. Tieman, R. Kettimuthu, and I. Foster. A data transfer framework for large-scale science experiments. In *Proceedings of DDC Workshop*, 2010.
- [40] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Modeling and taming parallel tcp on the wide area network. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 68b–68b. IEEE, 2005.
- [41] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *In Proceedings of IFIP Networking, May 2009*.
- [42] F. Murtagh and P. Legendre. Wards hierarchical agglomerative clustering method: which algorithms implement wards criterion? *Journal of classification*, 31(3):274–295, 2014.
- [43] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wether-all. Reducing network energy consumption via rate-adaptation and sleeping. In *Proceedings Of NSDI, April 2008*.
- [44] U. of Minnesota. Minnesota internet traffic studies (mints), 2012.
- [45] R. S. Prasad, M. Jain, and C. Dovrolis. Socket buffer auto-sizing for high-performance data transfers. *Journal of GRID computing*, 1(4):361–376, 2003.
- [46] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for internet-scale systems. In *ACM SIGCOMM computer communication review*, volume 39, pages 123–134. ACM, 2009.
- [47] F. Rawson and I. Austin. Mempower: A simple memory power analysis tool set. *IBM Austin Research Laboratory*, 2004.
- [48] Rclone-rsync for cloud storage. [https://rclone.org/commands/rclone\\_sync/](https://rclone.org/commands/rclone_sync/), 2017.
- [49] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 8:3–3, 2008.
- [50] J. Semke, J. Mahdavi, and M. Mathis. Automatic tcp buffer tuning. *ACM SIGCOMM Computer Communication Review*, 28(4):315–323, 1998.
- [51] C. Systems. Visual networking index: Forecast and methodology, 2015–2020, June 2016.
- [52] S. V. Vrbsky, M. Galloway, R. Carr, R. Nori, and D. Grubic. Decreasing power consumption with energy efficient data aware strategies. *FGCS*, 29(5):1152–1163, 2013.
- [53] E. Yildirim, E. Arslan, J. Kim, and T. Kosar. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. *IEEE Transactions on Cloud Computing (TCC)*, 4(1):63–75, 2016.
- [54] E. Yildirim, J. Kim, and T. Kosar. How gridftp pipelining, parallelism and concurrency work: A guide for optimizing large dataset transfers. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion.*, pages 506–515. IEEE, 2012.
- [55] E. Yildirim and T. Kosar. End-to-end data-flow parallelism for throughput optimization in high-speed networks. *Journal of Grid Computing*, pages 1–24, 2012.
- [56] E. Yildirim, D. Yin, and T. Kosar. Balancing tcp buffer vs parallel streams in application level throughput optimization. In *Proceedings of DADC Workshop*, 2009.
- [57] E. Yildirim, D. Yin, and T. Kosar. Prediction of optimal parallelism level in wide area data transfers. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(12), 2011.
- [58] D. Yin, E. Yildirim, and T. Kosar. A data throughput prediction and optimization service for widely distributed many-task computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6), 2011.
- [59] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Y. Wang. Modeling hard-disk power consumption. In *FAST 2003*.