Energy-Efficient Data Transfer Optimization via Decision-Tree Based Uncertainty Reduction

Hasibul Jamil, Lavone Rodolph, Jacob Goldverg and Tevfik Kosar
Department of Computer Science and Engineering
University at Buffalo (SUNY), Amherst, NY 14260, USA
Email: {mdhasibu, lrodolph, jacobgol, tkosar}@buffalo.edu

***Abstract*—The increase and rapid growth of data produced by scientific instruments, the Internet of Things (IoT), and social media is causing data transfer performance and resource consumption to garner much attention in the research community. The network infrastructure and end systems that enable this extensive data movement use a substantial amount of electricity, measured in terawatt-hours per year. Managing energy consumption within the core networking infrastructure is an active research area, but there is a limited amount of work on reducing power consumption at the end systems during active data transfers. This paper presents a novel two-phase dynamic throughput and energy optimization model that utilizes an offline decision-search-tree based clustering technique to encapsulate and categorize historical data transfer log information and an online search optimization algorithm to find the best application and kernel layer parameter combination to maximize the achieved data transfer throughput while minimizing the energy consumption. Our model also incorporates the ensemble method to reduce aleatoric uncertainty in choosing optimal application and kernel layer parameters during the offline analysis phase. The experimental evaluation results show that our decision-tree based model outperforms the state-of-the-art solutions in this area by achieving 117% higher throughput on average while consuming 19% less energy at the end systems during active data transfers.**

***Index Terms*—data transfer optimization, energy efficiency, decision search tree, diversity index, uncertainty quantification, historical log analysis.**

I. Introduction

The data requirements of commercial and scientific applications continue to increase at an unprecedented rate, generating more data in a given year than the total amount of data generated in the previous years. The advancements ranging from microscale sensor technologies to macroscale supercomputers and large scientific instruments are propelling multi-institutional experimental collaborations between geographically dispersed entities requiring massive data sharing using high-speed wide-area networks. Internet of Things (IoT), social media, and e-commerce applications generate a similar demand in the industry. It is estimated that the number of devices connected to IP networks will exceed 30 billion by 2023, which will be three times the global population [1]. As a result, annual data movement across the global Internet has already exceeded the zettabyte scale and continues to increase exponentially. This vast data movement comes with a large economic cost as well as a massive energy footprint. The energy consumption of telecommunication networks has recently exceeded 350 terawatt-hours, and the Internet comprises more than 10% of the overall energy consumption in many countries, costing the global economy billions of dollars per year [2].

An extensive portion of the existing research on energy-efficient networking focuses on reducing energy consumption in the core networking infrastructure (e.g., switches, hubs, and routers). State-of-the-art power-aware networking techniques include emerging architectures with programmable switches [3], and power-aware networking protocols designed to consider energy consumption while routing data [4], and putting idle components to sleep [5]. Some of these solutions have significant shortcomings. For example, placing the idle components to sleep can be detrimental to performance while saving power. Also, replacing existing network switches and network protocols with power-aware switches and energy-efficient network protocols is a costly solution and not practical in the short term.

In this paper, we introduce a novel decision-tree based cross-layer optimization solution, which is low cost, very easy and practical to deploy, and does not penalize the performance while increasing energy efficiency. Our approach can successfully co-tune several application-layer and kernel-layer parameters to achieve high data transfer throughput while minimizing energy consumption at the same time. Our model consists of two phases. In the first phase, an offline decision-search-tree based clustering technique is used to encapsulate and categorize historical data transfer log information, and partition the data space from historical data transfer log files that contain network and dataset characteristics [6]. This step also considers the uncertainty introduced by the stochastic nature of background traffic during a transfer. The goal of the offline analysis is to find the intrinsic structure of historical logs by organizing data objects into similarity groups or clusters [7]. In the second phase, an online dynamic search optimization algorithm is used to find the best application and kernel layer parameter combination based on our offline discovered knowledge and the real-time network conditions to maximize the achieved transfer throughput and minimize energy.

The major contributions of this paper include:

- To the best of our knowledge, our novel multifaceted approach is the first to dynamically construct decision trees based on historical data transfer logs encapsulating

network conditions and file characteristics while considering uncertainty introduced from historical data.
- Our approach is the first to tune both application-layer and kernel-layer data transfer parameters utilizing novel decision-search-tree based optimization to maximize the achieved data transfer throughput and reduce energy consumption in real-time.
- Our decision-tree based optimization algorithm achieves on average 117% higher transfer throughput compared to state-of-the-art solutions and 19% less energy usage in end systems during transfer.

The rest of the paper is organized as follows: Section II presents the problem formulation; Section III discusses our proposed decision-search-tree model and its construction; Section IV presents the evaluation of our model; Section V describes the related work in this field; and Section VI concludes the paper.

II. Problem Formulation

In this work, we perform joint tuning of three application-layer parameters (i.e., concurrency, parallelism, and pipelining) and two kernel-layer parameters (i.e., number of active CPU cores and CPU frequency level) to achieve high data transfer performance and low energy consumption at the same time. Concurrency (cc) controls the number of server processes/threads, and these processes/threads can transfer various files independently. As a result, concurrency can accelerate the transfer throughput when many files need to be moved [8] [9] [10]. Parallelism (p) is the number of data connections that each server process can open to transfer different portions of the same file in parallel. Parallelism is a fine option for medium or large file transfers in maximizing transfer throughput [11] [12] [13]. Pipelining (pp) is useful for small file transfers as it eliminates the delay imposed by the acknowledgment of the previous file transfer completion before starting the following one [14] [15] [16]. The kernel-layer parameters, such as the number of active CPU cores (cpu_num) involved in the data transfer and the frequency level of each active CPU core (cpu_freq), also have a significant impact on the achieved transfer throughput and energy consumption.

Given a source endpoint e_s and destination endpoint e_d; a connection link bandwidth b and round-trip-time rtt, a dataset with a total size d_{all}, average file size f_{avg}, buffer size buf_{size}, number of files n, and the load from contending transfers l_{ctd}, the achieved throughput T and end-system energy E consumption can be expressed as following functions:

$$T = f_1\left(b, rtt, f_{avg}, buf_{size}, n, cc, p, pp, cpu_{num}, cpu_{freq}, l_{ctd}\right) \tag{1}$$

$$E = f_2\left(b, rtt, f_{avg}, buf_{size}, n, cc, p, pp, cpu_{num}, cpu_{freq}, l_{ctd}\right) \tag{2}$$

For fixed $b, rtt, f_{avg}, buf_{size}, n$, Equation 1 and 2 could be converted to following functions:

$$T = g_1\left(\theta, l_{ctd}\right) \tag{3}$$

$$E = g_2\left(\theta, l_{ctd}\right) \tag{4}$$

where $\theta = \{cc, p, pp, cpu_{num}, cpu_{freq}\}$ is the list of tunable parameters. The optimization model is expressed with two types of service-level agreement (SLA) based objective functions: (1) minimum energy or energy-constrained SLA and (2) maximum throughput or throughput guarantee SLA. According to the SLA type and the SLA specifications, we need to find the optimal combination of the tunable parameters θ to satisfy the SLA requirements. The energy-constrained optimization model determines the matching node from the decision search tree and constructs the "energy surface" containing the matching node. We then find the minimum energy corresponding to the tunable parameters θ from the energy surface. Similarly, we construct the "throughput surface" containing the matching node from the logs for throughput optimization. We finally find the maximum throughput corresponding to tunable parameters θ from the throughput surface. The throughput and energy SLA based optimization models are expressed as below:

$$\begin{aligned} &\underset{\{cc,p,pp,cpu_{num},cpu_{freq}\}}{\text{argmax}} && \int_{t_s}^{t_f} th \\ &\text{subject to.} && cc \times p \leq \mathbb{N}_{\text{streams}} \\ & && pp \leq \mathbb{P} \\ & && th \leq b \\ & && E \leq E_{sla} \end{aligned} \tag{5}$$

$$\begin{aligned} &\underset{\{cc,p,pp,cpu_{num},cpu_{freq}\}}{\text{minimum}} && \int_{t_s}^{t_f} e \\ &\text{subject to.} && T \geq T_{sla} \end{aligned} \tag{6}$$

where t_s and t_f are the transfer start and end times respectively. Additionally, $\mathbb{N}_{streams}$ and P are the maximum allowable parameter values in the network. Our goal is to solve the maximum throughput SLA optimization model presented in Equation 5 and the minimum energy optimization model presented in Equation 6. As throughput is achieved in a shared environment, other concurrent contending transfers using the same network resources could affect the behavior of achievable throughput. Our historical logs contain information on such transfers. We define the load from those contending transfers as l_{ctd}. There are several assumptions made to create our optimization model, which we describe below:

- *Assumption 1:* All competing transfers at a given time can achieve aggregate throughput, $T = \sum_{i=1}^{N} th_i$, where N is the number of TCP streams for all the competing transfers, and th_i is the throughput of an individual transfer i.
- *Assumption 2:* The fluctuation on transfer behavior (i.e., throughput) depends on all other contending transfers l_{ctd} for given θ=$cc, p, pp, cpu_{num}, cpu_{freq}$ where $b, rtt, f_{avg}, buf_{size}, n$ are fixed.
- *Assumption 3:* Maximum achievable throughput is limited by the end-to-end link bandwidth, disk read speed at the source, or disk write speed at the destination. Given disk read speed v_{read} , disk write speed v_{write} , and the

link bandwidth b, the maximally achievable end-to-end throughput $th_{\max}$ would be:

$$th_{\max} \leqslant \min\{b, v_{write}, v_{\text{read}}\} \quad (7)$$

- *Assumption 4:* Introduced model is agnostic of the end nodes' underlying file systems. Because of parallelism and concurrency, the introduced model would fit superior performance when parallel file systems are used at the end nodes. Performance degradation due to hardware configuration error, storage access delay, and intermediate network system bottlenecks could limit the achievable throughput. Eliminating such bottlenecks might increase the limit of achievable throughput.

We chose to accumulate and store real-world historical log data transfer information; and utilize a decision search tree on the historical log files to cluster and group similar log entries based on network conditions, dataset meta-information, and network characteristics. The purpose of this grouping technique is to generalize relationships between throughput and θ, l_{ctd} as depicted in Equations 3 and 4. The obtained groups of logs are then used for throughput and energy surface construction and these surfaces are consulted to find optimum application and kernel parameter settings to achieve different SLA given $b, rtt, f_{avg}, buf_{size}, n$ are available. Finally, given a set of attribute parameters $b, rtt, f_{avg}, buf_{size}, n$ and a SLA, finding the max throughput associated or min energy associated tunable parameters θ=$cc, p, pp, cpu_{num}, cpu_{freq}$ is the objective of this work.

For example, in the data transfer log sample shown in Table I, we have 5 different groups of attribute/key parameters and their corresponding throughput and transfer energy with different application and kernel parameters. For each group, the historical logs have same $b, rtt, f_{avg}, buf f_{size}, n$ attributes but different throughput and transfer energy resulting from different tunable parameters θ=$cc, p, pp, cpu_{num}, cpu_{freq}$.

III. Decision Search Tree approach

A. How Decision Search Tree Approach Works

To make the search operation more efficient in the offline analysis phase, we construct a decision-search-tree based data clustering from the historical logs. The decision search tree captures the information found in historical logs by grouping similar logs together. The decision search tree is built so that intermediate nodes in the search tree contain the condition to reach leaf nodes, search tree edges contain the particular condition, and leaf nodes contain the log entries associated with the search attributes. Following figure 1 shows building such search trees for an example dataset shown in Table I. The root node contains all the logs, and for a different level of the tree, different attributes present in the historical log are used to cut the root node into multiple child nodes. Which attribute to choose while cutting a node is selected by an attribute selection scheme. The resultant tree could have different depths because of the attribution selection variability at different tree-building levels. In the first tree, as shown in figure 1(a), "$Bandwidth(BW)$" is chosen as the attribute to cut the root node, and all resultant child nodes achieve leaf threshold condition. As a result, the first tree reaches a tree height of 2.

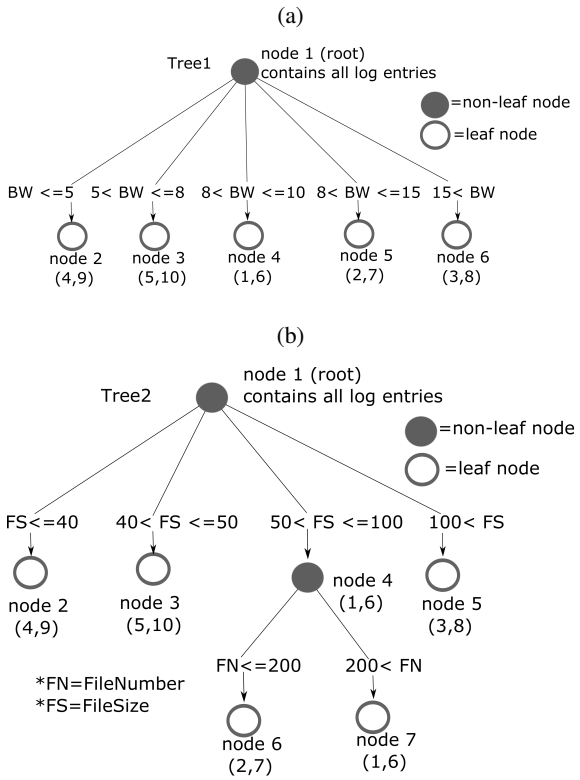

Fig. 1: For Tree1, we are considering bandwidth (BW) as attributes to split nodes and build complete tree from Table I. For Tree2, we consider FileSize (FS) and NumberOfFiles(FN) as attributes to split nodes and build complete tree from Table I. Resultant trees are different in height based on which attribute set we choose while building the tree.

In the second tree, as shown in in figure 1(b), during the first cut "$FileSize$" is selected as the attribute to cut the root node, and in the second cut "$FileNumber$" is chosen to cut node 4, resulting in a tree with depth 3 which is higher than tree height of the previous case. It should be noted that node 4 is the only one that gets cut in level 2 as the other node contained logs small or equal to the leaf threshold. The leaf threshold is the number of minimum logs that makes a node a leaf node, and in this example case, the leaf threshold is equal to 2.

Constructed decision-search-tree based searching for a matching leaf node has a $O(heightOfTree)$ time complexity, and we can build different trees for the same dataset following a different set of attributes. The generated trees, in that case, will be different in height. For a given set of attributes $(f_{avg}, n, rtt, buf_{size}, b)$, to find the matching leaf node in the search tree, the average case search time complexity is better than worst-case search time complexity as not all the leaf nodes will be on the same depth of the tree. Algorithm 1 is

TABLE I: Historical transfer log sample with tunable parameters and corresponding achieved throughput and energy expenditure

Entry No	File Size (kB)	#of Files	RTT(ms)	TCP Buffer Size(MB)	Bandwidth (Mbps)	Throughput (Mbps)	Transfer Energy(J)	C	P	PP	#of CPU	CPU Frequency(Hz)
1	100	250	10	200	10	5	20	1	2	2	2	1.3
2	100	200	8	150	15	3	17	1	1	1	2	1.3
3	50	150	15	250	20	4	15	1	2	1	2	1.3
4	40	150	20	225	5	1	12	1	2	2	2	1.3
5	150	225	15	150	8	5	22	2	3	3	2	1.3
6	100	250	10	200	10	8	15	2	3	3	4	1.5
7	100	200	8	150	15	10	10	3	4	4	4	1.5
8	50	150	15	250	20	8	9	3	1	4	4	1.5
9	40	150	20	225	5	4	7	3	2	3	4	1.5
10	150	225	15	150	8	4	16	2	1	3	4	1.5

used to build the decision tree. For an incoming data transfer request, the associated attributes are first obtained, and then these attributes are used to traverse the tree to reach the leaf node. The matched leaf nodes containing historical logs are then used to construct throughput and energy surfaces. By analyzing those surfaces, optimal application and kernel parameters are finally obtained.

Algorithm 1: Decision tree construction algorithm

```
input : dataset, leaf_threshold, cutNumber
// leaf_threshold is minimum number of logs that
   declare a node as leaf node
output decision search tree
:
availableNodeToCutList=[root] // start from root node
   S_0 that contains all the logs from Historical log
while availableNodeToCutList is not empty do
    currentNode=Pop(availableNodeToCutList);
    chooseAttributeToCut(currentNode);
    // use DI or SD metric to choose which dimension
       to cut the node
    childNodeList=CutNodeOnChoosenAttribute();
    // cut the node on the chosen dimension/attribute
       cutNumber times
    for each childNode in ChildNodeList do
        Assign matched logs to childNode;
        // some number of logs from parent node will
           fall into each child node
        childNodeLogCount=NumberOfLoginNode(childNode);
         if childNodeLogCount <= leaf_threshold
         OR sameChildNodeAttributeValue then
            Continue;
        end
        Push ChildNode to availableNodetocutList ;
    end
end
return root;
```

B. Handling of the Unseen Attributes

The system will return the last matched node as the final node for unseen attributes (i.e., not present in the history log) from an incoming data transfer request. For example, an incoming data transfer request with attributes $f_{avg}, n, rtt, buf_{size}, b$ respectively may not match any entries in the history log. For data transfer requests like this, the search will report application and kernel parameters from the constructed throughput surfaces of the last matched node of the tree where it needs not be a leaf node. If the attributes of the data transfer request will result in a match with node $N1$ then node $N1$ corresponding logs, and those logs contained throughput surfaces will be investigated to report final application and kernel parameters θ. For example, key parameter (100,255,10,200,10) (i.e.,$f_{avg}, n, rtt, buf_{size}, b$) will match node 4 on second tree shown in III(A) for fileSize attribute value (i.e.,100). The fileNumber attribute value (i.e., 255) is closer to node 7 of tree 2, so it will result in a match with node 7 and node 7 corresponding logs will be used to construct throughput and energy surfaces to find final application and kernel parameters θ.

C. Dealing with Uncertainty of Search Result

Uncertainty could be defined as situations consisting of unknown or partial knowledge and this type of uncertainty is fused in any stochastic and partially observable environments [17]. Epistemic uncertainty captures uncertainty in the model parameters and aleatoric uncertainty is the uncertainty caused by the noise that is inherited in the training dataset [18]. The historical log for data transfer suffers partial observation problems as computing clusters and network usually does not give real-time performance metrics for computation or network system resources. For example, storage servers utilization metric and queue size of devices in shared networks are some metrics that play paramount importance in transfer performance but these parameter metrics are private. The lack of these important real-time performance metrics from all components of end-to-end transfers across multiple clusters and network domains makes the historical dataset incomplete and introduces an aleatoric type of uncertainty.

In our work, the uncertainty is captured in equation 3 and 4 of our model as the contending transfers l_{ctd} parameter is not deterministic, the search result using the decision tree has aleatoric uncertainty associated with it. The decision search tree uncovers relationships between throughput and θ, l_{ctd} and the level of the aleatoric type of uncertainty could be reduced

by combining the search results made by an ensemble of decision trees and performing the throughput or energy surface construction on the combined search result logs.

An ensemble approach uses the combined output of a set of decision trees (e.g., decision search trees in our case) to improve on the search result accuracy offered by any one of its members [19] [20]. In our example, we build two trees as shown in section III(A) for sample log shown in Table I. For an incoming data transfer request with attributes parameters as $f_{avg}, n, rtt, buf_{size}, b$ respectively, the request matches with node $N1$ in first tree and with node $N2$ in second tree. We combine the historical logs from node $N1$ in tree-1 and logs from node $N2$ in tree-2 to construct the combined logs throughput and energy surfaces. Finally, these surfaces are then used to find maximum throughput or minimum energy points and associated optimum application and kernel parameter settings for that incoming data transfer request.

D. Constructing Decision Search Tree

The attribute selection procedure is novel in the proposed decision search tree building process where we consider diversity index and standard deviation as metrics from the historical log attribute set to choose the attribute at which we perform node cutting while building the tree.

1) Ranking the attributes based on Diversity Index: While building the search trees, we expect to build trees with minimum depth (i.e., traversing the tree is faster) and group the historical logs so that the grouped logs are as sparse as possible. To meet both of these objectives, choosing the appropriate attribute to cut a tree node is important. For example, we could choose any of the five attributes from the root node to derive the first layer (i.e., depth=1) nodes. The attribute should be chosen so that it maximally differentiates the historical logs. Diversity Index is such a metric that could be used to rank the attributes, and the ordered attributes could then be used to choose one attribute while building the tree [21]. The derivation of $diversity\ index$ for an attribute from a history log is derived as follows:

If S is a parameter value set of an attribute $S = \{c_1, c_2, c_3 \cdot \cdot \cdot \cdot \cdot c_n\}$ and $C_{max} = \max\{c_1, c_2, c_3 \cdot \cdot \cdot \cdot \cdot c_n\}$, $S_{normalized} = \left\{\frac{c_i}{C_{\max}}\right\}$ for i=1 to i=n. So, $S_{normalized} = \{c_{1-norm}, c_{2-norm}, c_{3-norm} \cdot \cdot \cdot \cdot \cdot c_{n-norm}\}$ and $S_{n-no-duplication} = \{c_{1-n-nd}, c_{2-n-nd} \cdot \cdot \cdot \cdot \cdot c_{n-n-nd}\}$ where, $S_{n-no-duplication} \subseteq S_{normalized}$ if $C_{max-norm} = \max\{c_{1-norm}, c_{2-norm}, c_{3-norm} \cdot \cdot c_{n-norm}\}$ $C_{min-norm} = \min\{c_{1-norm}, c_{2-norm}, c_{3-norm} \cdot \cdot c_{n-norm}\}$, then $diversity\ index$ (DI) could be calculated by following equation:

$$DI = (C_{max-norm} - C_{min-norm}) \times \sum_{i=c_{1-n-nd}}^{i=c_{n-n-nd}} \frac{1}{freq.\ of\ C_i} \tag{8}$$

The first part of Equation 8 captures how much range a particular attribute (for example, $FileSize$) has, and the second part captures how unique the values for that specific attribute are. The second part of Equation 8 is constructed to penalize multiple occurrences of a particular value.

2) Ranking the attributes based on Standard Deviation: Standard deviation is another metric that could be used to rank the attributes, and the ordered attributes could be used to build the tree while cutting a tree node. Standard deviation measures the relative spread of the values for each attribute field. The standard deviation for a particular attribute value is bigger when the differences of those attribute values are more spread out regarding the distribution mean. For a set of N numbers $\{x_1, x_2, \cdots, x_N\}$, if the mean of the collection is $\bar{x}$ then standard deviation, SD is as follows:

$$SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{9}$$

The variance was also considered to measure the spread of the fields in a ruleset, and described as follows:

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{N-1} \tag{10}$$

where S^2 = sample variance, x_i = the value of the one observation, $\bar{x}$ = the mean value of all observations, and N = the number of items in the set.

3) Band of trees: We build two trees for a given historical transfer log; one with diversity index as attribute selection metric and another with standard deviation as selection metric. Once tree-1 and tree-2 are built, the system traverses both trees to find the matching nodes from each tree for an incoming data transfer request with five attributes $(f_{avg}, n, rtt, buf_{size}, b)$. After traversing and matched nodes are found in the respective trees, application and kernel parameters are obtained as described in Section III(C).

E. Searching θ for Incoming Data Transfer Request

Given a historical log "L", the decision search tree leaf nodes cluster/group the logs based on combination of five attributes $(f_{avg}, n, rtt, buf_{size}, b)$. The clusters are $c_1, c_2, c_3 \cdot \cdot \cdot \cdot \cdot c_n$ where n is the number of clusters and is equal to the number of leaf nodes in a search tree. For each incoming transfer request, corresponding five attributes $(f_{avg}, n, rtt, buf_{size}, b)$ along with target throughput T_{es} or target energy $E_{es}$are used to traverse the search trees and find corresponding matching leaf nodes. Algorithm 2 describes the tree traversing and optimal parameter finding steps.

From a matched node containing historical logs, we construct throughput as a polynomial surface which is a function of the application and kernel parameters θ =$cc, p, pp, cpu_{num}, cpu_{freq}$. Incoming transfer request corresponding attributes $f_{avg}, n, rtt, buf_{size}, b$ matches one of the leaf node in the decision search tree and that node corresponding logs could construct multiple surfaces. The matched surface could then be decomposed into multiple components (i.e., binned surface components) based on the different binned values of throughput (i.e., throughput 0-100 could be binned to bin 100, 101-200 to bin 200, and so on) so that expected

Algorithm 2: Optimal parameter discovery algorithm

input : β =Incoming transfer request attributes $b, rtt, f_{avg}, buf_{size}, n$, decision tree,targetThroughput T_{es} or targetEnergy E_{es} as SLA

// targetThroughput is the Throughput the algorithm is targeting to achieve based on SLA and targetEnergy is the transfer energy budget based on SLA

output θ =cc,p,pp,cpu_num,cpu_freq

:

1 allTheNodesOfTreeList=getAllTheNodesAsList(decision tree);
2 **while** *allTheNodesOfTreeList is not empty* **do**
3 currentNode=Pop(allTheNodesOfTreeList);
4 **if** *currentNode has* β **then**
5 matchedNode=currentNode;
6 break;
7 **end**
8 **end**
9 **if** *SLA==MAXThroughput* **then**
10 surfaces= decomposedThroughputSurface(matchedNode);
 // matchedNode throughput surface is decomposed into multiple components with different throughput level
11 finalThroughputSurface=getFinalSurface(surfaces,T_{es});
12 θ=maxSurfacePoint(finalThroughputSurface);
13 **end**
14 **else**
15 surfaces= decomposedEnergySurface(matchedNode);
 // matchedNode energy surface is decomposed into multiple components with different energy level
16 finalEnergySurface=getFinalSurface(surfaces,E_{es});
17 θ=minSurfacePoint(finalEnergySurface);
18 **end**
return θ ;

throughput could be matched with a binned surface component.

In line 10, the algorithm finds the closest throughput surface component from expected throughput T_{es}. In line 11 and 12 and calculate the maximum throughput point in the selected surface component, and return the maximum throughput point corresponding to application and kernel parameters as final θ=$cc, p, pp, cpu\_num, cpu\_freq$. Similar is true for finding min energy SLA corresponding θ as shown in line 15,16 and 17. Using algorithm 2, we pre-compute and store optimized kernel-level and application-level transfer parameters in our custom data structure for our dynamic online program to use.

F. Online Dynamic Maximum Throughput Parameter Tuning

We cluster historical log entries based on network characteristics, network conditions, and file characteristics during the offline analysis phase. Using algorithm 2, we pre-compute and store optimized kernel-level and application-level transfer parameters for an exhaustive combinations of scenarios in our custom data structure for our dynamic online program to use. Table II shows the offline analysis time for three different testbed with different number of historical logs. The offline analysis consist of two phases: decision tree construction time and decision tree traversing time while generating the custom data structure for our dynamic online program. Algorithm 3 shows the steps to use the pre-computed data structure from section III(E) for maximum throughput optimization SLA. Before a data transfer task starts, our online dynamic tuning algorithm for throughput first measures the RTT for 3 seconds which is shown in line 3 of algorithm 3. Our algorithm utilizes the measured average RTT and the associated expected throughput entry along with other file characteristics as the search key to retrieve initial transfer parameters (θ) from the offline stage pre-computed data structure. Using the initial transfer parameters, the algorithm performs the data transfer for approximately 10 seconds before obtaining new transfer parameters from the decision search tree utilizing the measured average delta RTT and the instantaneous delta throughput. This is necessary to ensure previously retrieved parameters (theta) have not become sub-optimal. Since network conditions may fluctuate in a shared network, we periodically check and adjust transfer parameters for the reasons mentioned above. We developed three periodic time intervals to check and adjust transfer parameters based on network conditions and dataset characteristics. For small, medium, and large datasets, we review and adjust parameters every 10, 20, and 30 seconds respectively as shown in line 7 to 10 of algorithm 3.

TABLE II: Characteristics of testbeds

Testbed	Bandwidth	RTT	BDP	CPU Architecture
Chameleon	10 Gbps	34 ms	40 MB	Haswell (server) Haswell (client)
CloudLab	1 Gbps	38 ms	4.5 MB	Haswell (server) Broadwell (client)
Inter-Cloud	1 Gbps	45 ms	4.5 MB	Haswell (server) Bloomfield (client)

Algorithm 3: Dynamic Throughput Tuning

1 datasets=$ClusterFiles()$;
2 $Timeout = getTimeout(Dataset.AvgFileSize)$
3 δ rtt = $measureRtt(time\_3sec)$
4 θ = $obtainInitParams(\delta$ rtt)
5 $startTransfer(\theta$, Dataset)
6 **for** $Timeout$ **do**
7 δ throughput = $measuredInstTrhoughput()$
8 δ rtt = $measuredRtt()$
9 θ = $SearchTreeParams(\delta rtt, \delta Throughput)$
10 $UpdateParameters(\theta$, Dataset)
11 **end**

G. Online Dynamic Minimum Energy Parameter Tuning

Based on the SLA, if the given SLA is minimum energy, a dynamic online energy constraint tuning algorithm shown in Algorithm 4 derived from the offline energy constraint optimization model. Based on the energy constraint SLA, parameter tuner periodically monitors the instantaneous energy consumption at specified regular time intervals as shown in line 7 in Algorithm 4. It then uses this instantaneous energy consumption and transfer elapsed time (i.e., line 8) to approximate the energy consumption of the transfer. With the approximate energy value and current measured delta rtt (i.e., line 3), the dynamic tuner obtains the updated transfer parameters (i.e., line 10).

Algorithm 4: Dynamic Energy Tuning

```
1  datasets=ClusterFiles();
2  Timeout = getTimeout(Dataset.AvgFileSize)
3  δ rtt = measureRtt(time_3sec)
4  θ = obtainInitParams(δrtt)
5  startTransfer(θ, Dataset)
6  for Timeout do
7      δ Energy =measuredInstEnergy()
8      elapsedTime=getElapsedTime()
9      approximateEnergy =
         enrgyApproximation(δEnergy, elapsedTime)
10     θ =
         SearchTreeParams(δrtt, approximateEnergy)
11     Transfer(θ, Dataset)
12 end
```

IV. Experimental Evaluation

We performed over 45,000 data transfers across three diverse wide-area network testbeds, collecting valuable experimental data encapsulating network conditions and characteristics. Testbeds utilized include (1) Chameleon Cloud [22], server located at the University of Chicago and client located at the Texas Advanced Computing Center; (2) CloudLab [23], server located at the University of Wisconsin and client located at the University of Utah; (3) Inter-Cloud, server located at the Texas Advanced Computing Center (part of Chameleon Cloud) and client located at the University of Utah (part of CloudLab). Both the Chameleon nodes run on either a Dell PowerEdge R630 containing 24 CPU cores distributed in dual-socket Intel Xeon E5-2670 v3 "Haswell" processors, each containing 12 cores, or on a PowerEdge R740 containing two Intel Xeon Skylake CPUs (each with 12 cores / 24 threads) [22]. The client within the CloudLab architecture runs on an HPE ProLiant XL170r server containing 10 CPU cores plus hyper-threading distributed in an Intel E5-2640v4 "Broadwell" processor having 64 GiB of RAM. The server within the CloudLab testbed and the client within the inter-cloud testbed run on Cisco's UCS SFF 220 M4 and UCS LFF 240 M4, respectively. Both contain 2 Intel E5-2630 "Haswell" processors, each having eight cores plus hyper-threading and 128 GiB of RAM [23]. The server within the Inter-Cloud testbed shares the same specifications as the Chameleon Cloud server. A specification overview is provided in Table III.

TABLE III: Required time for offline analysis

TestBed (#of logs)	Cloudlab (9841)	Chameleon (18325)	InterCloud (11228)
Average Decision tree Construction Time (Second)	4.5397	10.41378	4.3038
Hash Table construction time from tree traversing time (Second)	9.025355	13.0955	20.253139

TABLE IV: Dataset Characteristics

Dataset	Num Files	Total Size	Avg. File Size	Std. Dev.
Small	20,000	1.94 GB	101.92 KB	29.06 KB
Medium	5,000	11.70 GB	2.40 MB	0.27 MB
Large	128	27.85 GB	222.78 MB	15.19 MB

Experimental data transfers were performed during peak and non-peak hours, utilizing three diverse datasets containing different characteristics. These include: (1) the small file size dataset consisting of 20,000 HTML files derived from the common crawl project [24]; (2) the medium file size dataset consisting of 5,000 image files derived from Flickr [25]; (3) the large file size dataset consisting of 128 video files from Jiku [26]. Complete dataset characteristics are specified in Table IV.

On all of our testbeds, we measure the client's energy consumption using Intel's Running Average Power Limit (RAPL), which uses a software model to accurately estimate power consumption based on hardware performance counters and I/O models. David et al. [27] and Hahnel et al. [28] highlighted RAPL's precision in measuring both memory and CPU power consumption. To distinguish data transfer power consumption from total system power consumption, we subtracted the system baseline power consumption from the total power. In order to fairly compare the efficiency of our dynamic minimum energy algorithm and maximum throughput algorithm with other transfer solutions, we utilize the extreme use cases. To achieve this, we use an SLA policy that informs our dynamic energy constraint algorithm to transfer data with the least amount of energy consumption by utilizing cross-layer optimal kernel-level and application-level parameters. We call this SLA policy the minimum energy decision tree (i.e., Min Energy D tree) SLA. To test our maximum throughput optimization algorithm, we use an SLA policy maximum throughput SLA (i.e., Max Throughput D tree) that enforces our algorithm to transfer data with the maximum throughput rate achievable, utilizing cross-layer optimal kernel-level and application-level parameters.

The experimental results in Figure 2 show how our proposed decision tree-based max throughput and min energy algorithms compare to Di Tacchio's algorithm implemented in [29]. Di Tacchio et al. developed real-time tuning heuristics to opti-

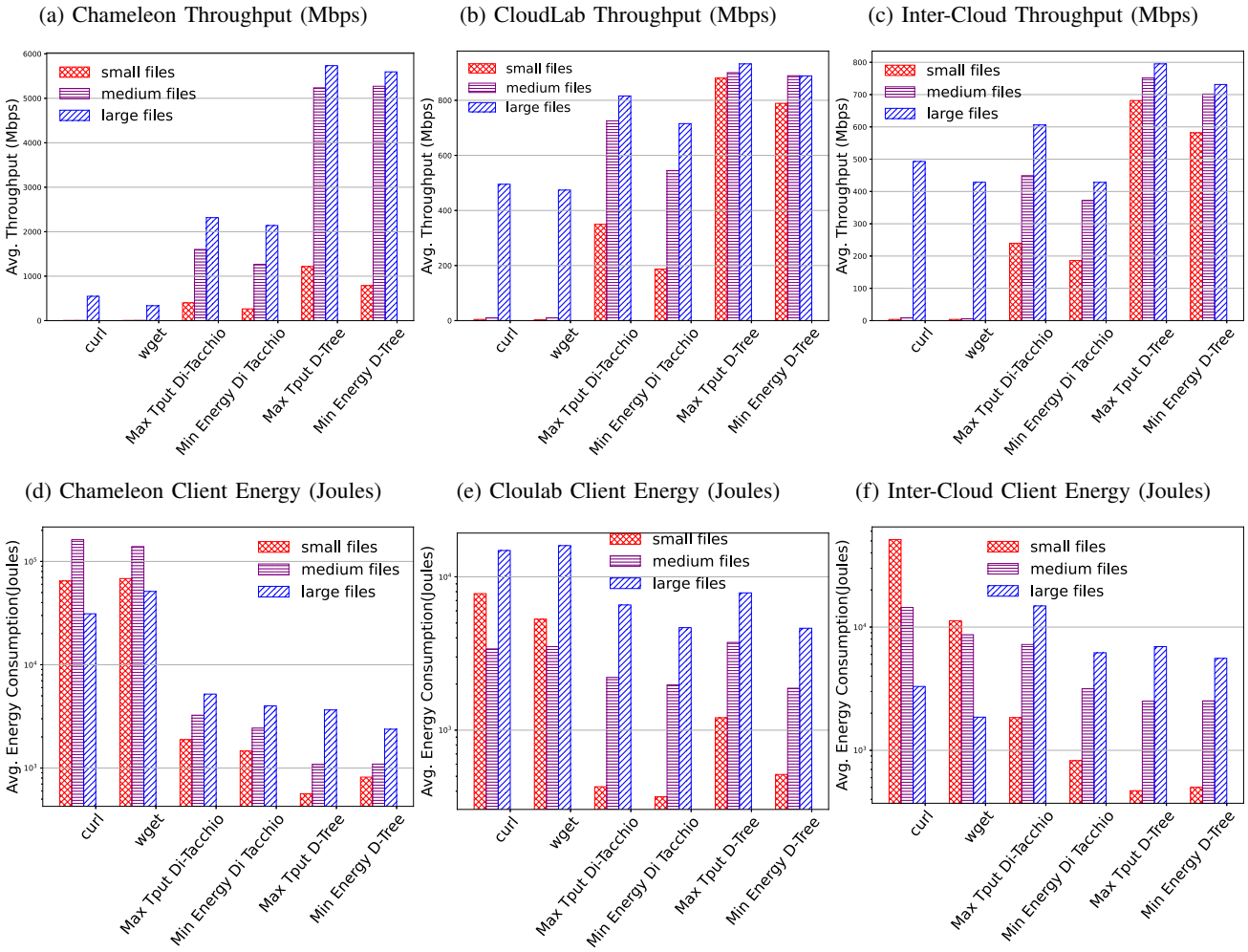

Fig. 2: Achieved throughput (Mbps) and energy consumption (Joules) over 3 diverse testbeds.

mize throughput and minimize energy consumption by tuning both application-level data transfer parameters and kernel-layer parameters during HTTP transfers. Furthermore, we compared our algorithms to two common data transfer baseline tools: (1) curl, an open-source tool used to transfer data, and (2) wget, a free command-line tool used to retrieve files from the web. For a fair comparison between all algorithms and baseline tools, we utilized the datasets specified in table III when performing experimental data transfers. In addition, since the baseline tools did not support service-level agreements (SLAs), we set the SLAs of our models/algorithms to two diametrical cases: (1) maximum achievable throughput and (2) minimum achievable energy consumption.

Figure 2 compares throughput performance and energy consumption across three diverse testbeds: (1) Chameleon, (2) CloudLab, and (3) Inter-Cloud. As anticipated, curl and wget produced sub-par results across all testbeds for all data transfers due to the absence of parameter optimization. Di Tacchio et al.'s algorithms performed better than all the baseline tools. As demonstrated in sub-figures, 2(a) and 2(d), our algorithm outperforms all other algorithms in both throughput performance and energy consumption cases within a high bandwidth and high Bandwidth Delay Product (BDP) network environment (Chameleon Cloud). Di Tacchio's heuristic does not take advantage of past data transfer history and must adjust parameters strictly on fluctuating network conditions.

Figures 2(b) and 2(e) show how our dynamic max throughput and min energy algorithm outperforms all other algorithms in lower bandwidth and lower BDP network environments (Cloudlab testbed). Sub-figures 2(c) and 2(f) demonstrate how our algorithms outperform all other algorithms and baseline tools for datasets containing small size HTML files, datasets containing medium size image files, and a dataset containing large video files in the intercloud testbed. Decision tree max throughput outperformed Max throughput (Di Tacchio) across all datasets in throughput improvement of HTML data transfers by 180%, image data transfers by 120%, and video data transfers by 52%. Additionally, the decision tree minimum energy algorithm decreased energy consumption by an additional 37% for HTML data transfers, 23% for image data transfers, and

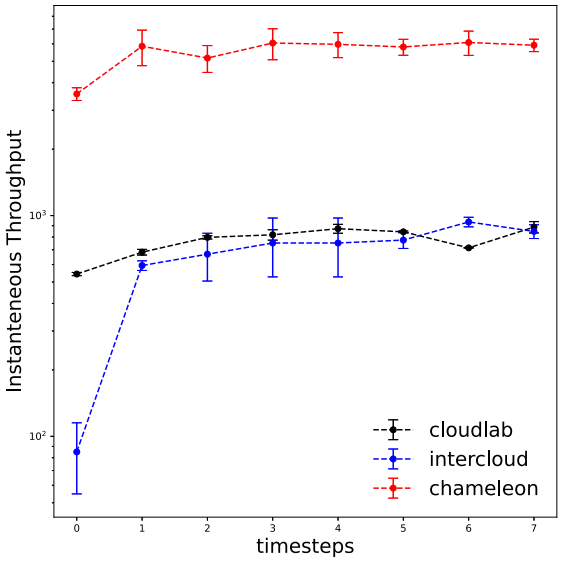

Fig. 3: The convergence of achieved throughput with decision tree max throughput algorithm for all three testbeds. The x axis shows the timesteps and y axis is instantaneous throughput in log scale.

6% for video data transfers, with respect to the minimum Energy(Di Tacchio) algorithm. The cost of overestimating or underestimating parameter values and compute resources are expensive. Overestimating parameter values and compute resources can increase energy consumption.

Conversely, underestimating parameter values and compute resources can degrade throughput performance. Utilizing decision-search-tree based clustering techniques on past data transfer history logs allows our algorithms to accurately estimate near-optimal data transfer parameters for a given SLA based on the current network conditions. This causes our algorithms to converge faster to optimal parameters. Our dynamic maximum throughput and min energy algorithms converge quickly to optimal cross-layer parameter values based on real-time network feedback and historical log analysis. As shown in figure 3, with the decision tree approach, the convergence to optimal throughput takes the minimal number of timesteps[1]. This faster convergence is because the decision tree combines the current network condition and knowledge from offline historical analysis to provide faster convergence towards maximally achievable throughput.

V. Related Work

Improving data transfer throughput in wide-area high-speed networks is a challenging task that depends on many dynamic factors, including but not limited to: network conditions, network settings, data transfer parameter configuration, and dataset characteristics. Therefore, it is difficult to develop predictive models to accurately capture hidden throughput patterns in fluctuating network environments. Several works on application-level parameter tuning during data transfer mostly propose non-scalable and static solutions to the problem with some predefined values for the subset of the problem space [30], [31], [32], [33].Earlier research focused on developing predictive models based on three methodologies: analytical, empirical, and model-free [33], [34]. Hacker et al. [31] developed an analytical model correlating throughput with various network parameters. Yildirim et al. [35] developed an empirical approach by applying Newton's method to find the optimal number of TCP streams via parallelism that will maximize throughput. In our prior work [29] and [36], we developed a combination of model-free and empirical algorithms to maximize throughput by utilizing both machine learning techniques and online dynamic parameter tuning. This paper extends our prior work by using a novel decision tree-based model capable of dealing with the aleatoric uncertainty presented in the historical logs.

[1]For each data point in the line, it denotes the mean of all the instantaneous throughput at that timestep and the vertical bars at each data point shows the distribution of all the instantaneous throughput at that timestep. From the plot it could be deduced that decision tree throughput algorithm converges to the maximum achievable throughput within one timestep.

VI. Conclusion

Augmenting data transfer throughput and minimizing energy consumption during data transfer in end hosts over high-speed, long-distance networks is becoming progressively challenging. Numerous factors such as fluctuating network conditions, limitations of underlying transfer protocols, dataset characteristics, network characteristics, and data transfer parameter settings must be considered to achieve optimal or close to optimal performance. A multifaceted approach to improving data transfer throughput and minimum energy consumption involves optimally and dynamically adjusting application-layer and kernel-layer transfer parameters based on real-time network conditions and dataset characteristics. In this paper, we presented a novel two-phase dynamic throughput predictive optimization model that utilizes offline decision-search-tree based learning techniques to encapsulate and categorize historical data transfer log information and an online search optimization algorithm to find optimal transfer parameters. Furthermore, we also explore ensemble methods to tackle the uncertainty while carrying out the offline phase of analyzing historical log data and building a decision search tree. The experimental evaluation shows that our decision tree-based model outperforms state-of-the-art solutions in this area by achieving 117% higher throughput on average while consuming 19% less energy at the end systems during active data transfers.

Acknowledgements

This project is in part sponsored by the National Science Foundation (NSF) under award numbers CCF-2007829, OAC-1842054, OAC-1724898. We also would like to thank the Chameleon Cloud and CloudLab for letting us use their resources in our experiments.

References

[1] Cisco, "Cisco annual internet report (2018–2023) white paper," *Online](accessed March 26, 2021) https://www. cisco. com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/whitepaper-c11-741490. html*, 2020.

[2] L. B. G.w.a.t.t., "Measuring energy consumption for short code paths using rapl," 2020.

[3] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: scalability and commoditization," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, 2008, pp. 57–62.

[4] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power awareness in network design and routing," in *IEEE INFOCOM*, 2008, pp. 457–465.

[5] M. Gupta and S. Singh, "Greening of the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 19–26.

[6] L. Castin and B. Ffenay, "Clustering with decision trees: Divisive and agglomerative approach," 2018.

[7] B. Liu, Y. Xia, and P. S. Yu, "Clustering via decision tree construction," 2004.

[8] 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*, ser. HPDC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 717–724. [Online]. Available: https://doi.org/10.1145/1851476.1851582

[9] T. Kosar, "Dynamically tuning level of parallelism in wide area data transfers," in *Ph.D. thesis*. University of Wisconsin–Madison, 2005.

[10] E. Yildirim and T. Kosar, "End-to-end data-flow parallelism for throughput optimization in high-speed networks," *J. Grid Comput.*, vol. 10, no. 3, p. 395–418, sep 2012. [Online]. Available: https://doi.org/10.1007/s10723-012-9220-9

[11] T. Hacker, B. Noble, and B. Athey, "Adaptive data block scheduling for parallel tcp streams," in *HPDC-14. Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005.*, 2005, pp. 265–275.

[12] E. Yildirim, M. Balman, and T. Kosar, "Dynamically tuning level of parallelism in wide area data transfers," in *Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing*, ser. DADC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 39–48. [Online]. Available: https://doi.org/10.1145/1383519.1383524

[13] E. Yildirim, D. Yin, and T. Kosar, "Balancing tcp buffer vs parallel streams in application level throughput optimization," in *Proceedings of the Second International Workshop on Data-Aware Distributed Computing*, ser. DADC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 21–30. [Online]. Available: https://doi.org/10.1145/1552280.1552283

[14] B. K. Y. Z. J. P. K. Farkas, P. Huang1, "Impact of tcp variants on http performance," *Proc. High Speed Netw. 2*. [Online]. Available: http://www.hit.bme.hu/~farkask/publications/hot_HSNWKS2002.pdf

[15] F. N., "Smtp service extension for command pipelining," *RFC Editor*.

[16] J. Kim, E. Yildirim, and T. Kosar, "A highly-accurate and low-overhead prediction model for transfer throughput optimization," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 787–795.

[17] G. J. K. Bilal M. Ayyub, "Uncertainty modeling and analysis in engineering and the sciences (1st ed.)." in *Chapman and Hall/CRC. https://doi.org/10.1201/9781420011456*, 2006.

[18] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. W. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenkov, and S. Nahavandi, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *CoRR*, vol. abs/2011.06225, 2020. [Online]. Available: https://arxiv.org/abs/2011.06225

[19] M. C. Darling and D. J. Stracuzzi, "Toward uncertainty quantification for supervised classification," 1 2018. [Online]. Available: https://www.osti.gov/biblio/1527311

[20] S. K. Mahjour, L. O. Mendes da Silva, L. A. A. Meira, G. P. Coelho, A. A. de Souza dos Santos, and D. J. Schiozer, "Evaluation of unsupervised machine learning frameworks to select representative geological realizations for uncertainty quantification," *Journal of Petroleum Science and Engineering*, vol. 209, p. 109822, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920410521014418

[21] H. Jamil and N. Weng, "Multibit tries packet classification with deep reinforcement learning," in *2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR)*, 2020, pp. 1–6.

[22] K. Keahey, J. Anderson, Z. Zhen, and et al., "Lessons learned from the chameleon testbed," in *Proceedings of USENIX ATC'20*, July 2020.

[23] D. Duplyakin, R. Ricci, A. Maricq, and et al., "The design and operation of CloudLab," in *Proceedings of the USENIX ATC*, Jul. 2019.

[24] "Common crawl," http://commoncrawl.org, 2020.

[25] "Yahoo flickr creative commons," https://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67., 2020.

[26] "Jiku mobile video dataset," http://www.jiku.org/datasets.html, 2020.

[27] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," p. 189–194, 2010. [Online]. Available: https://doi.org/10.1145/1840845.1840883

[28] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using rapl," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, p. 13–17, jan 2012. [Online]. Available: https://doi.org/10.1145/2425248.2425252

[29] L. Di Tacchio, M. S. Q. Z. Nine, T. Kosar, M. F. Bulut, and J. Hwang, "Cross-layer optimization of big data transfer throughput and energy consumption," in *2019 IEEE CLOUD*, 2019.

[30] K. T. K. C. DEELMAN, E. and M. LIVNY, "What makes workflows work in an opportunistic environment?" vol. 18, no. 10, 2006, p. 1187–1199.

[31] T. J. Hacker, B. D. Noble, and B. D. Atley, "The end-to-end performance effects of parallel tcp sockets on a lossy wide area network," in *Proceedings of IPDPS '02*. IEEE, April 2002, p. 314.

[32] G. Kola, T. Kosar, J. Frey, M. Livny, R. Brunner, and M. Remijan, "Disc: A system for distributed data intensive scientific computing," 2004.

[33] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, "Modeling and taming parallel tcp on the wide area network," in *Proceedings of IPDPS '05*. IEEE, April 2005, p. 68.2.

[34] D. Yin, E. Yildirim, S. Kulasekaran, B. Ross, and T. Kosar, "A data throughput prediction and optimization service for widely distributed many-task computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 899–909, 2011.

[35] E. Yildirim, D. Yin, and T. Kosar, "Prediction of optimal parallelism level in wide area data transfers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2033–2045, 2011.

[36] L. Rodolph, M. S. Q. Zulkar Nine, L. Di Tacchio, and T. Kosar, "Energy-saving cross-layer optimization of big data transfer based on historical log analysis," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–7.