Energy-saving Cross-layer Optimization of Big Data Transfer Based on Historical Log Analysis

Lavone Rodolph, MD S Q Zulkar Nine, Luigi Di Tacchio, and Tevfik Kosar
Department of Computer Science and Engineering, University at Buffalo, Buffalo, New York 14260, USA
Email: {lrodolph, mdsqzulk, luigidit, tkosar}@buffalo.edu

***Abstract*—With the proliferation of data movement across the Internet, global data traffic per year has already exceeded the Zettabyte scale. The network infrastructure and end-systems facilitating the vast data movement consume an extensive amount of electricity, measured in terawatt-hours per year. This massive energy footprint costs the world economy billions of dollars partially due to energy consumed at the network end-systems. Although extensive research has been done on managing power consumption within the core networking infrastructure, there is little research on reducing the power consumption at the end-systems during active data transfers. This paper presents a novel cross-layer optimization framework, called Cross-LayerHLA, to minimize energy consumption at the end-systems by applying machine learning techniques to historical transfer logs and extracting the hidden relationships between different parameters affecting both the performance and resource utilization. It utilizes offline analysis to improve online learning and dynamic tuning of application-level and kernel-level parameters with minimal overhead. This approach minimizes end-system energy consumption and maximizes data transfer throughput. Our experimental results show that Cross-LayerHLA outperforms other state-of-the-art solutions in this area.**

***Index Terms*—energy-efficient data transfers, cross-layer optimization, dynamic parameter tuning, historical log analysis.**

I. Introduction

With the vast amount of data produced by scientific and engineering applications, social media, cloud computing, and the emerging Internet of Things (IoT), it is estimated that by 2022 the global Internet traffic will exceed 4.8 zettabytes per year caused by a projected 28.5 billion network-connected devices [1]. The energy consumption of telecommunication networks has already 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]. This massive energy footprint has ignited immense research in power-aware networking, focusing on reducing power consumption at both the hardware and systems-levels, including network devices.

A large portion of the existing energy-efficient networking research focuses on reducing power consumption within 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], power-aware networking protocols designed to consider energy consumption while routing data [4], and putting idle components to sleep [5]. On the other hand, many of the existing core infrastructure solutions have shortcomings. For example, putting idle components to sleep can be detrimental to throughput. Also, replacing existing network switches and network protocols with power-aware switches and energy-efficient network protocols are expensive solutions and not practical in the short term. This paper presents a cost-effective, energy-efficient, practical end-system solution, called Cross-LayerHLA, which is an optimization framework that combines offline historical log analysis with dynamic online tuning to minimize end-system energy consumption and maximize data transfer throughput.

The major contributions of Cross-LayerHLA include:

- To the best of our knowledge, it is the first to integrate cross-layer adaptive online tuning with offline analysis using machine learning techniques to reduce energy consumption at the end systems and improve data transfer throughput performance.
- It is the first to dynamically tune both application-level and kernel-level data transfer parameters while obtaining optimal parameter values from offline analysis based on real-time network conditions.
- Compared to state-of-the-art solutions, it reduces end-system energy consumption during data transfers considerably while increasing the data transfer throughput at the same time.

The rest of this paper is organized as follows: Section II describes the related work in the field; Section III describes how we model and optimize energy consumption and throughput performance based on historical log analysis and dynamic online tuning; Section IV evaluates Cross-LayerHLA and compares it to other state-of-the-art solutions in this area; and Section V concludes the paper.

II. Related Work

Prior work on application level tuning of transfer parameters mostly proposed static or non-scalable solutions to the problem with some predefined values for the subset of the problem space [6], [7]. The main problem with such solutions is that they do not consider the dynamic nature of the network links and the background traffic in the intermediate nodes. Kasparan et al. [8] analyzed how pipelining affects throughput in local area networks and high-speed downlink packet access networks. Natarajan et al. [9] showed that using a single stream for transferring independent web objects is very inefficient in high latency networks. Using concurrent channels for delivering such objects increases download rates and decreases browser response times by enabling concurrent rendering.

Robert et al. [10] used parallel file requests to increase usage of available bandwidth during Internet video streaming. Li et al. [11] presented a parallel streaming method instead of traditional adaptive sequential streaming. In distributed network environments, fetching media segments by parallel channels increases streaming performance and also copes with inefficient usage of resources. Alan et al. [12] analyzed the effects of different application-layer protocol parameters such as TCP pipelining, parallelism and concurrency levels on end-to-end throughput versus total energy consumption.

In our prior work [13], we combined application-level (pipelining, parallelism, and concurrency) and kernel-level parameters (number of active cores, CPU frequency level) to minimize energy consumption at the end systems (source and destination nodes) during active data transfers. In that work, we heuristically performed the optimization without considering historical log analysis, but its convergence to optimal values was slow. In this work, we optimize data transfers and reduce energy consumption by jointly employing offline historical analysis, machine learning techniques, and online dynamic real-time tuning.

III. Modeling and Optimizing Transfer Throughput and Energy Consumption

To accurately model the data transfer throughput and end-system energy consumption, we consider and ensure that: (1) our model is representative and characteristic of real-world data transfers; (2) our model accounts for fluctuating networking conditions, external background traffic, and external network loads; and (3) our model considers dataset characteristics (dataset size, average file size within a dataset, and file size distribution). The dataset characteristics would have a significant impact on transfer throughput performance and end-system energy consumption [12]. For example, transferring a dataset comprised of multiple small files (such as text files) can benefit significantly from high concurrency and pipelining, and transferring a single large file would benefit from parallelism. To convey the impact of the kernel-level and application-level transfer parameters on end-system energy consumption and throughput performance, we express the relationships as the following functions:

$$E = f_1(cpu_{num}, cpu_{freq}, cc, p, pp, data, net) \quad (1)$$

$$T = f_2(cpu_{num}, cpu_{freq}, cc, p, pp, data, net) \quad (2)$$

where E is the energy consumption, T is the throughput performance, cpu_{num} is the number of active CPU cores, cpu_{freq} is the frequency level of each active CPU core, cc is the concurrency level, p is the parallelism level, pp is the pipelining level, $data$ is the dataset characteristics and net is the network characteristics.

To address the factors above, we chose to: (1) accumulate and store real-world historical log data transfer information; (2) utilize machine learning to perform clustering on the historical log files, grouping/stratifying similar log entries based on network conditions, dataset meta-information, and network characteristics to uncover hidden relationships; and (3) derive an appropriate interpolation or regression model based on the rendered data characteristics. These steps are explained in the offline optimization subsection below.

A. Offline Analysis

Step 1: Accumulate and cache historical log data: During a data transfer, we accumulate the following information: i) throughput performance; ii) energy consumption; iii) network characteristics; iv) end-system resource usage; v) dataset characteristics; and vi) kernel-level and application-level parameter configuration. Kernel-level and application-level parameter configuration includes: i) the number of active CPU cores; ii) the frequency level in which the active CPU cores operate; iii) concurrency; iv) parallelism; and v) pipelining. Historical log files are accumulated periodically and cached in a log server.

Step 2: Stratify historical log data: After accumulating historical log data, we stratify the log entries based on similar data characteristics. We utilize Hierarchical Agglomerative Clustering (HAC) [14] with the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [15] to achieve this. Also, we enforce artificial boundaries based on data transfers' distribution characteristics to ensure accurate stratification. Boundaries are based on the mean, and standard deviation of the achieved throughput and energy consumption of multiple collective data transfer runs. We utilize a bottom-up approach to cluster data in tiers. For tier 1, we cluster log entries based on external network characteristics. For tier 2, we apply agglomerative hierarchical clustering within each of the tier 1 clusters, further grouping log entries based on dataset meta-info (dataset size, file distribution, average file size and number of files in dataset). Since high external network loads decreases throughput performance and increases end-system energy consumption we further stratify the strata/data into interval ranges based on the data transfer logs' external network load distribution characteristics. This is necessary as current optimal parameters may become suboptimal as external network traffic changes. For tier 3, we further group tier 2 clusters based on network characteristics such as source and destination nodes and their corresponding bandwidth and latency.

Step 3: Derive interpolation/regression model: After data is accumulated, stratified, and categorized, we analyze and extract the hidden relationships within them. Since estimated energy consumption and throughput can be modeled as polynomial surfaces, we utilized piece-wise cubic spline interpolation [16] to derive the interpolants for each cluster/stratum. Second-order (quadratic) polynomial interpolation and third-order (cubic) polynomial interpolation based on Newton's or Lagrange's methods [17] do not fit the data as smoothly as piece-wise cubic polynomials, which assures smoothness up to the second derivative. Since each data transfer parameter has distinct qualities, we model them independently. We first model a 2-dimensional cubic spline interpolation for energy consumption utilizing the number of CPU cores, denoted as

cpu, as the abscissas and the end-system energy consumption values, denoted as e, as the ordinates.

$$e_i = s_i(cpu_i) \tag{3}$$

Given a cluster/stratum of discrete points which we will refer to as knots in a 2-dimensional space $\{(cpu_i, e_i)\}$, $i = 1, ..., N$, we formulate the interpolant $e_i = s(cpu_i)$ by using the piece-wise polynomial $s_i(cpu)$ as a nexus bridging together the pair of consecutive knots (cpu_i, e_i) and (cpu_{i+1}, e_{i+1}). This allows us to derive natural relaxed piece-wise cubic polynomials, where each e_i have zero second derivatives at the end knots and assures curvature smoothness by restricting its coefficients. We can define each cubic polynomial piece as:

$$s_i(cpu_i) = a_{i,0} + a_{i,1}cpu + a_{i,2}cpu^2 + a_{i,3}cpu^3 \forall cpu \in [cpu_i, cpu_{i+1}] \tag{4}$$

Since we have $N-1$ piece-wise cubic polynomials, we have $4(N-1)$ unknown coefficients $a_{i,j}$ where $j = 0, 1, 2, 3$ of piece-wise cubic polynomial $s_i(cpu)$. We also have N continuity conditions/stipulations for all given knots evaluated by the corresponding piece-wise cubic polynomial as specified by:

$$s_i(cpu_i) = e_i, \quad i = 1, .., N \tag{5}$$

Furthermore, we have $N-2$ continuity conditions/stipulations for all interior knots specified as:

$$s_i(cpu_{i+1}) = s_{i+1}(cpu_{i+1}), \quad i = 1, .., N-2 \tag{6}$$

Utilizing Leibniz's notation for differentiation [18], we can specify $N-2$ slope continuity conditions/stipulations for the interior knots as:

$$\frac{ds_i}{d(cpu)}(cpu_{i+1}) = \frac{ds_{i+1}}{d(cpu)}(cpu_{i+1}) \quad i = 1, .., N-2 \tag{7}$$

This produces $N-2$ quadratic polynomial continuity stipulations. We also enforce $N-2$ additional continuity stipulations for the interior knots utilizing the second derivative as follows:

$$\frac{d^2 s_i}{d^2(cpu)}(cpu_{i+1}) = \frac{d^2 s_{i+1}}{d^2(cpu)}(cpu_{i+1}) \quad i = 1, .., N-2 \tag{8}$$

This produces $N-2$ Linear polynomial stipulations. Since we are constructing a natural relaxed piece-wise cubic spline polynomial the second derivative at the end knots are zero and may be specified as follows:

$$\frac{d^2 s}{d^2(cpu)}(cpu_1) = \frac{d^2 s}{d^2(cpu)}(cpu_n) = 0 \tag{9}$$

By solving the system of linear equations we obtain the coefficients.

End-system energy consumption also depends on the other four data transfer parameters: frequency (freq), concurrency (cc), pipelining (pp), and Parallelism (p). We extend the above example to formulate three multivariate piece-wise cubic spline polynomials corresponding to cubic polynomial surfaces. These include: $e_1 = s(cpu, freq)$, $e_2 = s(cc, p)$ and $e_3 = s(pp)$. For each cubic spline polynomial we obtain the optimal parameter values based on data transfer service level agreements (SLA) and perform uniform sampling matching the criteria. Utilizing piece-wise cubic-spline surface interpolation, we were able to stitch together multiple cubic functions and predict/estimate both energy consumption and throughput performance of the previously missing parameters. This was necessary as optimal transfer parameter values needed to meet a particular SLA may not be present in the historical log data. To test the accuracy of our cubic spline interpolation method, we used 70% of the logs to perform the interpolation and the remaining 30% as the test data. We used the standard Root Mean Squared Error (RMSE) [19] to calculate the accuracy.

B. SLA Optimization

We developed two categories of service-level agreements (SLAs) as our optimization goals: (1) energy-constrained SLAs and (2) throughput assurance/guarantee SLAs. Based on the SLA type and the SLA specifications, we must select an optimal combination of the tunable parameters to satisfy the SLA requirements. To find the optimal parameter values for minimum end-system energy consumption, our energy-constrained optimization model determines the local minima in each of the three multivariate piece-wise cubic spline interpolation formulas and selects parameters corresponding to the minimum of the minima. This can be achieved by performing the second partial derivative test on all local minima, by calculating the Hessian matrix and determining if the matrix is positive definite. For throughput optimization, we perform the reverse, find all local maxima, calculate the Hessian matrix, and determine if the matrix is negative definite. We extend our previous optimization model [20] to include kernel-level parameters and express it as:

$$\begin{aligned} \min_{parameters(kernel,app)} \quad & (E) \\ \text{subject to} \quad & T \geq T_{sla} \end{aligned} \tag{10}$$

$$\begin{aligned} \max_{parameters(kernel,app)} \quad & \int_{T_s}^{T_f} T \\ \text{subject to} \quad & E \leq E_{sla} \end{aligned} \tag{11}$$

The energy-constrained model allows a user to select optimal data transfer parameters that maximize throughput while ensuring the achieved energy consumption does not exceed the threshold specified by the SLA. The throughput guarantee model allows a user to select optimal data transfer parameters that minimizes energy consumption while ensuring the achieved throughput does not fall below the throughput threshold specified in the SLA. We utilize a Matlab solver to obtain optimal transfer parameters based on the specified SLA.

After the offline analysis is performed, we store optimized kernel-level and application-level transfer parameters in our custom data structure for our dynamic online program to use. Each data structure instance corresponds to the optimal parameter configuration of a cluster satisfying the SLA.

Algorithm 1: Dynamic Energy Constraint Tuning

```
datasets = ClusterFiles()
for Timeout do
    calculateDeltaEnergy()
    calculateWeightedThroughput()
    calculateDeltaRtt().
    t = dataSize / Tavg.
    E_pred = P_avg * t
    calculateExternalNetworkPercentage()
    startWithLightExtNetworkLoadSurface()
    /* If Energy Increased                                   */
    if (ΔE_last + E_pred > (1 + β) * pastE_pred) ||
      (ΔE_last + E_pred > SLA) then
        /* External Network Load Increased                   */
        if Ext_Net > (1 + β) * Ext_refNet then
            /* Obtain Cluster/Surface with Higher
               External Network Load                         */
            surface_Highθ ← NewOptParams
        end
    end
    // Energy Decreased & Ext Network Load
       Decreased
    else if refExtNet < (1 − α) * Ext_NetLast then
        surface_Lowθ ← NewOptParams
    end
end
```

Algorithm 2: Dynamic Throughput Tuning

```
datasets = ClusterFiles()
for Timeout do
    calculateWeightedThroughput()
    calculateDeltaRtt().
    calculateExternalNetworkPercentage()
    startWithLightExtNetworkLoadSurface()
    /* If Throughput Decreased                               */
    if T_avg < (1 − α) * T_Last || (T_avg < SLA) then
        /* External Network Load Increased                   */
        if Ext_Net > (1 + β) * refExtNet then
            /* Obtain Cluster/Surface with Higher
               External Network Load                         */
            surface_Highθ ← NewOptParams
        end
    end
    else
        // Throughput Increased
        if // External Network Load decreased
        refExtNet < (1 − α) * Ext_NetLast then
            /* Obtain Cluster/Surface with Lower
               External Network Load                         */
            surface_Lowθ ← NewOptParams
        end
    end
end
```

C. Online Dynamic Energy Constraint Tuning

Cross-LayerHLA employs a dynamic online energy constraint tuning algorithm shown in Algorithm 1 derived from the offline energy constraint optimization model. Based on the energy constraint SLA, it periodically monitors the instantaneous power consumption at specified regular time intervals. If the instantaneous power consumption exceeds the threshold specified in the SLA, it obtains new optimal parameters within the confidence range/sampling region by obtaining a new energy polynomial surface that is closest to the current measured external network load and energy consumption range. If the measured instantaneous power consumption decreased from the last interval check, it obtains the closest energy polynomial surface and retrieves the associated optimal data transfer parameters. This is done approximately three times as continuously changing parameters are expensive.

D. Online Dynamic Throughput Guarantee Constraint Tuning

Cross-LayerHLA employs a dynamic online throughput constraint tuning algorithm shown in Algorithm 2 derived from the offline throughput constraint optimization model. In this algorithm, it periodically monitors the instantaneous throughput at specified regular time intervals. If the instantaneous throughput falls below the threshold specified in the SLA, it obtains new optimal parameters within the confidence range/sampling region by obtaining a new polynomial throughput surface that is closest to the current measured external network load. If the measured instantaneous throughput increased or decreased from the last interval check, it obtains the closest throughput polynomial surface and retrieves the associated optimal data transfer parameters for the associated external network load. Obtaining optimal parameters from the polynomial throughput surfaces is done at a maximum of three times since it is expensive to modify the data transfer parameters. Afterward, if necessary, data transfer parameters are adjusted heuristically.

E. Testing Online Dynamic Algorithms

In order to fairly compare the efficiency of our dynamic energy constraint algorithm and throughput guarantee algorithm with other transfer tools/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/power consumption by utilizing joint optimal kernel-level and application-level parameters. We call this SLA policy the minimum energy consumption SLA. To test our throughput guarantee algorithm, we use an SLA policy that enforces our algorithm to transfer data with the maximum throughput rate achievable, utilizing joint optimal kernel-level and application-level parameters.

IV. Experimental Evaluation

We collected experimental data by performing data transfers on three different wide-area network testbeds over a one week period. Our testbeds include (1) Chameleon Cloud, server located at the University of Chicago and client located at

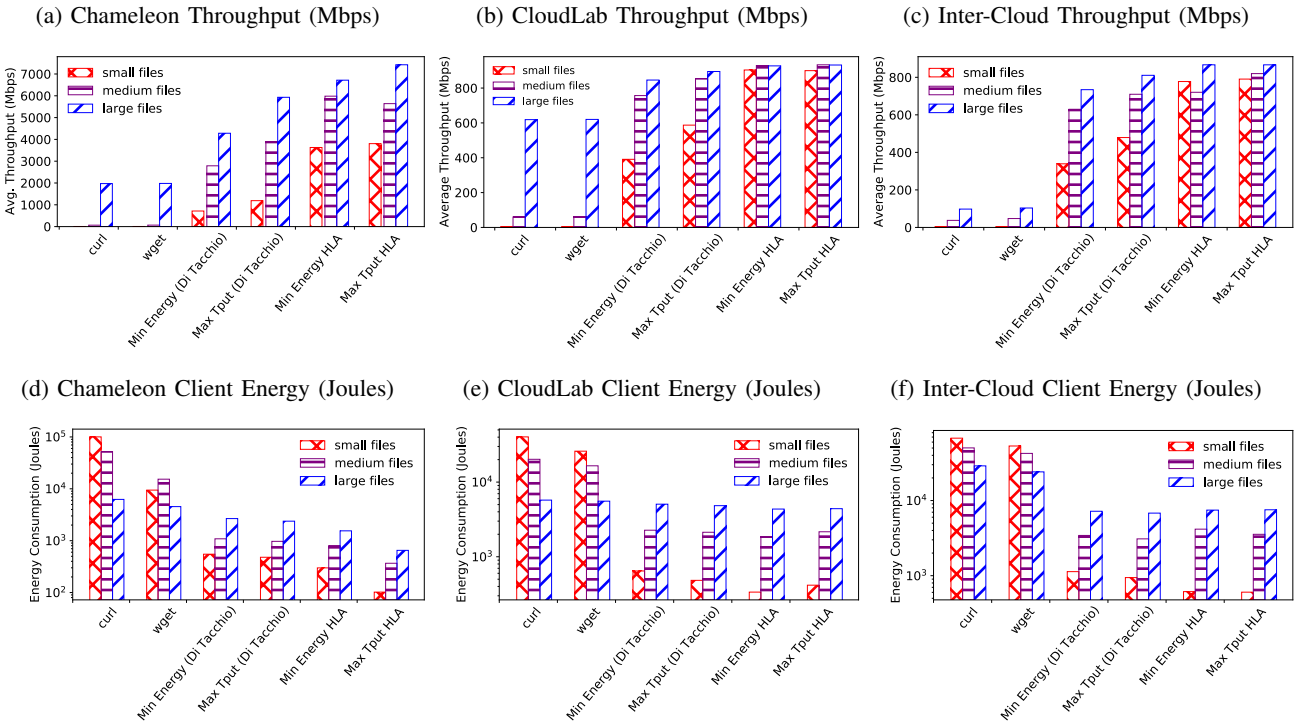

Fig. 1: Achieved throughput (Mbps) and energy consumption (Joules) over 3 diverse testbeds

Testbed	Bandwidth	RTT	BDP	CPU Architecture
Chameleon	10 Gbps	32 ms	40 MB	Haswell (server) Haswell (client)
CloudLab	1 Gbps	36 ms	4.5 MB	Haswell (server) Broadwell (client)
Inter-Cloud	1 Gbps	48 ms	6 MB	Haswell (server) Bloomfield (client)

TABLE I: Characteristics of testbeds

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

TABLE II: Dataset Characteristics

the Texas Advanced Computing Center; (2) CloudLab, 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 Wisconsin (part of CloudLab). Both the Chameleon nodes run on a Dell PowerEdge R630 containing 24 CPU cores distributed in dual-socket Intel Xeon E5-2670 v3 "Haswell" processors, each containing 12 cores and 128 GiB of RAM [21]. 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 containing 64 GiB of RAM. The server within the CloudLab testbed as well as 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 containing eight cores plus hyper-threading and 128 GiB of RAM [22]. The server within the Inter-Cloud testbed shares the same specifications as the Chameleon Cloud server. A specification overview is provided in table I. Experimental data transfers were performed during both peak hours 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 [23]; (2) the medium file size dataset consisting of 5,000 image files derived from Flickr [24]; (3) the large file size dataset consisting of 128 video files from Jiku [25]. Complete dataset characteristics are specified in table II. We collected and stored data transfer meta-information on our log servers. We analyzed, grouped, and performed optimizations on our historical log data based on the target SLAs.

On all of our testbeds we measure the client's power 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. [26] and Hähnel et al. [27] 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 readings. In addition, to mitigate the disk-to-disk and I/O-to-disk transfer latency, we performed memory-to-memory data transfers.

To the best of our knowledge there are three related state-of-the-art solutions in this area. These include the algorithms proposed by Alan et al. [12], Di Tacchio et al. [13] and

Nine et al. [20]. Alan et al. [12] implemented energy-aware algorithms to optimize HTTP data transfers and reduce power consumption based on heuristics that lacked real-time tuning, adversely affecting throughput performance and end-system energy consumption during fluctuating network conditions. Di Tacchio et al. developed real-time tuning heuristics to optimize throughput and minimize energy consumption by tuning both application-level data transfer parameters and kernel-level parameters during HTTP transfers. However, at times, convergence to near optimal parameters was slow causing data transfers to utilize suboptimal parameter combinations for extended time periods during the tuning process. Slow convergence adversely affects throughput performance and increases end-system energy consumption. Over-estimating data transfer parameters and over- provisioning compute resources can increase energy cost. Under-estimating data transfer parameters and under-provisioning compute resources can reduce throughput performance. Nine et al. [20] proposed dynamic tuning algorithms for GridFTP data transfers utilizing offline historical log analysis and online tuning mechanisms; however, they did not consider jointly scaling pertinent kernel-level parameters with data transfer application-level parameters. Resources such as the number of active cores and CPU frequency determine the number of Instructions per Second (IPS) that can be executed and affects throughput performance and end-system energy consumption. Dynamically tuning application-level parameters without tuning kernel-level parameters can adversely affect throughput performance and end-system energy consumption.

Since we developed and implemented algorithms to dynamically tune HTTP data transfer parameters we compared our algorithms to those of Di Tacchio et al. [13]. 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 fair comparison between all algorithms and baseline tools we utilized the datasets specified in table II 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 (Max Tput HLA) and (2) minimum achievable energy consumption (Min Energy HLA).

Figure 1 displays a comparison of throughput performance and energy consumption across three diverse testbeds: 1.) Chameleon Cloud, 2.) CloudLab and 3.) Inter-Cloud. As anticipated, curl and wget produced subpar results across all testbeds for all data transfers due to the absence of parameter optimization. Di Tacchio et al. algorithms performed better than all the baseline tools. However, executing dataset transfers utilizing Di Tacchio et al. algorithms (Max Tput (Di Tacchio) and Min Energy (Di Tacchio) on the large Bandwidth-Delay-Product (BDP) testbed (Chameleon Cloud) illustrated a few shortcomings: i) Heuristically tuning transfer parameters without using regression analysis or offline estimation techniques based on past data transfer history may cause slow convergence to optimal parameter values. This may be further exacerbated by fluctuating external network conditions which may cause the heuristic to overestimate application-level transfer parameters and compute resources such as the number of CPU cores. Additionally, dynamic external network conditions can cause the heuristic to underestimate application-level data transfer parameters and compute resources. ii) The cost of over-estimating or under-estimating parameter values and compute resources is expensive. Overestimating parameter values and compute resources can increase energy consumption. Conversely, underestimating parameter values and compute resources can degrade throughput performance. Utilizing machine learning techniques and interpolation 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. This is clearly observed in figure 1(a) and 1(d) which compares achieved throughput performance and data transfer energy across all algorithms and datasets. Max Tput HLA outperformed Max Tput (Di Tacchio) across all datasets increasing throughput of HTML data transfers by 69%, image data transfers by 31% and video data transfers by 20%. Furthermore, Max Tput HLA achieved the lowest power consumption due to reduced data transfer time. Extended data transfer time increases both static and non-static power consumption. Additionally, Min Energy HLA algorithm decreased energy consumption by an additional 46% for HTML data transfers, 26% for image data transfers and 42% for video data transfers, with respect to Min Energy (Di Tacchio). Further throughput improvement was observed when we compared Max Tput HLA algorithms against the baseline tools. Max Tput HLA improved throughput performance for both HTML and image data transfers by up to 99% when compared to curl and wget. For video data transfers, Max Tput HLA increased throughput by 73%. Furthermore, Min Energy HLA decreased energy consumption by an additional 99% for HTML data transfers, 98% for image data transfers and 75% for video data transfers. Moreover, for all algorithms we enhanced throughput performance and minimized end-system energy consumption utilizing approximately 67% of the available CPU cores.

Data transfers executed on the CloudLab network, a lower BDP network, demonstrated that Max Tput HLA outperformed Max Tput (Di Tacchio) by up to 35% for HTML transfers. Max Tput HLA further increased throughput by 14% for image data transfers and 8% for video transfers. For lower BDP networks, both Max Tput (Di Tacchio) and Min Energy (Di Tacchio) were able to converge faster to optimal parameters than on larger BDP networks. Nevertheless, the heuristic tuning mechanism has the propensity to either overestimate or underestimate data transfer parameters and compute resources when network conditions fluctuate. On the otherhand, Min Energy HLA decreased energy consumption by an additional 48% for HTML data transfers, 18% for image data transfers and 9% for video transfers compared to Min Energy (Di Tacchio). With respect to the baseline tools, Max Tput

HLA further increased throughput by up to 99% for HTML transfers, 94% for image transfer and up to 34% for video transfers. Furthermore, Min Energy HLA decreased energy consumption by an additional 99% for HTML transfers and up to 24% for video transfers. Experiments performed on the Inter-Cloud network demonstrated that the Max Tput HLA algorithm increased throughput by an additional 40% for HTML data transfer, 13% for image data transfers and 8% for video data transfers with respect to Max Tput (Di Tacchio). Furthermore, Min Energy HLA decreased energy consumption by an additional 36% for HTML data transfers, but slightly increased energy consumption for image and video transfers. Compared to curl, Max Tput HLA increased throughput for HTML data transfers by 99%, 95% for image data transfers and 89% for video data transfers. In addition, Min Energy HLA decreased energy consumption by an additional 99% for HTML data transfers, 92% for image data transfer and 74% for video data transfers. Compared to wget, Max Tput HLA increased throughput by 99% for HTML data transfers, 95% for image data transfers and 89% for video data transfers. Furthermore, Min Energy HLA decreased energy consumption by an additional 99% for HTML data transfers, 93% for image data transfers and 88% for video data transfers. Additionally, for all algorithms we enhanced performance by approximately using only 80% of the available CPU cores.

.

V. Conclusion

In this paper, we have introduced a cross-layer optimization framework that combines offline analysis with adaptive online tuning to minimize end-system energy consumption and maximize data transfer throughput performance. We presented novel algorithms that dynamically tune both application-level and kernel-level parameters based on historical log analysis and current network conditions to meet the requirements set by the service-level agreements (SLAs). Our detailed experimental analysis and results show that our proposed Cross-LayerHLA algorithms outperforms the state-of-the-art solutions in this area, reducing energy consumption considerably while increasing data transfer throughput.

Acknowledgements

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

References

[1] "Cisco visual networking index: Forecast and Trends, 2017022 White Paper," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html, accessed: 2020-02-07.

[2] B. Labs. (2020) G.w.a.t.t.: New bell labs application able to measure the impact of technologies like sdn and nfv on network energy consumption. [Online]. Available: http://media-bell-labs-com.s3.amazonaws.com/pages/20150114_1907/$GWATT_WhitePaper.pdf$

[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] 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.

[7] 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.

[8] D. Kaspar, K. Evensen, P. Engelstad, and A. F. Hansen, "Using http pipelining to improve progressive download over multiple heterogeneous interfaces," in *2010 IEEE International Conference on Communications*. IEEE, 2010, pp. 1–5.

[9] P. Natarajan, F. Baker, and P. Amer, "Multiple tcp connections improve http throughput-myth or fact?" in *2009 IEEE 28th International Performance Computing and Communications Conference*, 2009.

[10] R. Kuschnig, I. Kofler, and H. Hellwagner, "Improving internet video streaming performance by parallel tcp-based request-response streams," in *2010 7th IEEE Consumer Communications and Networking Conference*. IEEE, 2010, pp. 1–5.

[11] C. Liu, I. Bouazizi, and M. Gabbouj, "Parallel adaptive http media streaming," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, pp. 1–6.

[12] I. Alan, E. Arslan, and T. Kosar, "Energy-aware data transfer algorithms," in *In Proc. of SC'15*. IEEE, 2015, pp. 1–12.

[13] L. Di Tacchio, M. S. Z. Nine, T. Kosar, M. F. Bulut, and J. Hwang, "Cross-layer optimization of big data transfer throughput and energy consumption," in *IEEE CLOUD*, 2019, pp. 25–32.

[14] F. Murtagh and P. Legendre, "Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion?" *Journal of classification*, vol. 31, no. 3, pp. 274–295, 2014.

[15] I. Gronau and S. Moran, "Optimal implementations of upgma and other common clustering algorithms," *Information Processing Letters*, vol. 104, no. 6, pp. 205–210, 2007.

[16] W. Bickley, "Piecewise cubic interpolation and two-point boundary problems," *The Computer Journal*, vol. 11, no. 2, pp. 206–208, 1968.

[17] D. Kincaid, D. R. Kincaid, and E. W. Cheney, *Numerical analysis: mathematics of scientific computing*. American Mathematical Soc., 2009, vol. 2.

[18] H. Thurston, "Leibniz's notation," *The Mathematical Gazette*, vol. 57, no. 401, pp. 189–191, 1973.

[19] N. Wiener, "The wiener rms (root mean square) error criterion in filter design and prediction," 1949.

[20] M. S. Z. Nine, L. Di Tacchio, A. Imran, T. Kosar, M. F. Bulut, and J. Hwang, "Greendataflow: Minimizing the energy footprint of global data movement," in *IEEE Big Data*, 2018, pp. 335–342.

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

[22] D. Duplyakin, R. Ricci, A. Maricq, and et al., "The design and operation of CloudLab," in *Proceedings of USENIX ATC'19*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

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

[24] "Yahoo flickr creative commons," https://webscope.sandbox.yahoo.com/ catalog.php?datatype=idid=67., 2020.

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

[26] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *ACM/IEEE ISLPED*, Aug 2010, pp. 189–194.

[27] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using rapl," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 3, pp. 13–17, 2012.