Information Sciences 476 (2019) 413–428

Contents lists available at ScienceDirect

Information Sciences

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

Energy-aware data throughput optimization for next generation internet

Tevfik Kosar[a,*], Ismail Alan[a], M. Fatih Bulut[b]

[a] University at Buffalo (SUNY), 338 Davis Hall, Buffalo, New York 14260, United States
[b] IBM T.J. Watson Research Center, Yorktown Heights, New York 10598, United States

ARTICLE INFO

Article history:
Received 4 December 2017
Revised 24 September 2018
Accepted 30 September 2018
Available online 5 October 2018

Keywords:
Application-level protocol tuning
Energy efficiency
Throughput optimization
HTTP
Transfer algorithms

ABSTRACT

According to recent statistics, more than 1 zettabytes of data is moved over the Internet annually, which consumes several terawatt hours of electricity, and costs billions of US dollars to the world economy. Hypertext Transfer Protocol (HTTP) is used in the majority of these data transfers, accounting for 70% of the global Internet traffic. We claim that HTTP transfers, and the services based on HTTP, can become more energy efficient without any performance degradation by application-level tuning of certain protocol parameters. In this paper, we analyze several application-level parameters that affect the throughput and energy consumption in HTTP data transfers, such as the level of parallelism, concurrency, and pipelining. We introduce novel service-level-agreement (SLA) based algorithms which can decide the best combination of these parameters considering user-defined energy efficiency and performance criteria. Our experimental results show that up to 80% energy savings can be achieved at the client and server hosts during HTTP data transfers, while increasing the end-to-end transfer throughput at the same time.

1. Introduction

The number of devices connected to Internet will be three times as high as the world population in 2021, and the global IP traffic will reach 3.3 zettabytes per year[46]. The increased number of users and data rates do not only require increased network bandwidth and achievable data transfer throughput, but also result in increased energy footprint. The annual electricity consumed by the global data movement is estimated to be more than 100 terawatt hours at the current rate, costing more than 20 billion US dollars per year [22,41,46]. This fact has resulted in considerable amount of work focusing on power management and energy efficiency in hardware and software systems [12,16,26,43,45,48] as well as on power-aware networking [19,20,27,38].

Most 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. [22] were amongst 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 [23], which were later extended by other researchers. Nedevshi et al. proposed adapting the rate at which switches forward packets depending on the traffic [40]. 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

* Corresponding author.
E-mail address: tkosar@buffalo.edu (T. Kosar).

https://doi.org/10.1016/j.ins.2018.09.065
0020-0255/

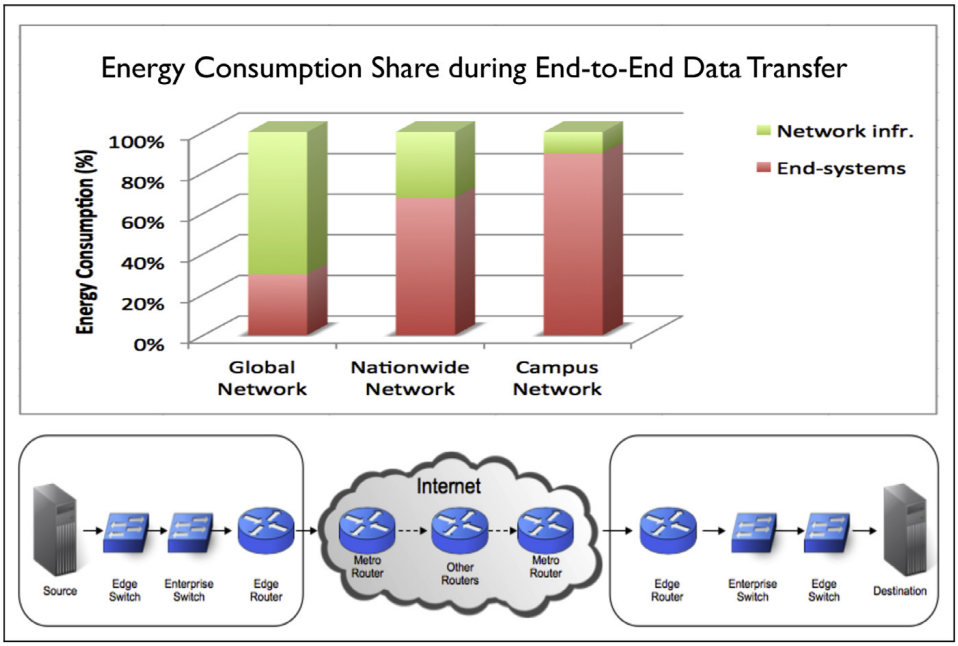

Fig. 1. Energy consumption share of the end systems vs network infrastructure during end-to-end data transfer. This ratio depends on the number of network devices (i.e., routers, switches, etc.) between the end systems, and how much power each device consumes.

to low power mode when there is no network traffic. Other related research in power-aware networking has focused on architectures with programmable switches [21], switching layers that can incorporate different policies [28], and power-aware network protocols for energy efficiency in network routing [13].

Notwithstanding the broad range of research in power management techniques for the networking infrastructure, there has been little work focusing on saving data transfer energy at the end systems (i.e., sender and receiver nodes). Approximately 25% of total electricity consumption during the end-to-end data transfers occur at the end-systems on a global (intercontinental) network, whereas this number goes up to 60% on a nationwide network, and up to 90% on a local area network [7]. This ratio depends on the number of network devices (i.e., routers, switches, hubs, etc.) between the sender and receiver nodes, and how much power each device consumes, as illustrated in Fig. 1. On any of these networks, decreasing the end-system power consumption would result in significant energy savings considering exabytes of data are moved, and terawatts of electricity is consumed in worldwide data movement every year.

While achieving energy efficiency, we should avoid penalizing the performance, as empirical data suggests performance directly impacts revenue. For example, Google reported 20% revenue loss due to a specific experiment that increased the time to display search results by as little as 500 milliseconds; and Amazon reported a 1% sales decrease for an additional delay of as little as 100 milliseconds [30]. We argue that in certain cases, energy savings and an increase in performance can be achieved simultaneously.

In this paper, we analyze the performance versus energy consumption trade-offs in Hypertext Transfer Protocol (HTTP), which is the de-facto transport protocol for Internet and related services ranging from file sharing to media streaming. Studies analyzing the Internet traffic [14,44] show that in recent years HTTP holds the largest share and accounts for 70% of the global Internet traffic. Recent studies argue that HTTP will become the narrow waist of the future Internet, meaning the vast majority of Internet traffic is expected to run over HTTP regardless of the underlying transport protocol [42]. We introduce service-level agreement (SLA) based HTTP data transfer algorithms which can decide the best combination of these parameters based on user-set energy efficiency and performance criteria. To the best of our knowledge, this is the first work proposing energy-aware data transfer algorithms for high-performance HTTP data transfers with a quantification of the key practical tradeoffs between energy reduction and QoS/performance of data transfers impacting customer SLAs.

The presented SLA-based HTTP transfer algorithms let the end users to define their throughput and energy requirements. While keeping the quality of service (transfer throughput in this case) at the desired level, they keep the power consumption at the minimum possible level via tuning of the application-level transfer parameters. With the help of our SLA-based energy-efficient data transfer algorithms, the Internet service providers will be able to minimize the energy consumption during data transfers without compromising the SLA with the customer in terms of the promised performance level, but still execute the transfers with minimal energy levels given the requirements. Such a capability will be crucial for almost all big IT companies which provide cloud-hosted, web-based, or Internet-of-Things (IoT) related services.

IT companies can benefit from our novel approach in different ways. First, they can provide HTTP-based energy-efficient transfer service in their cloud offerings. Second, IT companies can integrate our novel approach to their existing applications that are already adapted by many users. Especially mobile and IoT related applications, which depend on battery power

and need energy-efficiency most, can immensely benefit from our approach with minimum change in their implementation. Experimental results show that our proposed approach can save up to 80% energy at the end systems during HTTP data transfers, and it can increase the end-to-end data transfer throughput at the same time.

The rest of this paper is organized as follows. Section 2 presents background information on energy-aware tuning of HTTP and discusses the related work in this area. Section 3 analyzes the effects of different protocol parameters on HTTP performance as well as on end-system energy consumption. Section 4 presents our SLA-based data transfer algorithms. Section 5 discusses the experimental results; and Section 6 concludes the paper.

2. Energy-aware HTTP tuning

One common way to increase the data transfer throughput at the application level is through the tuning of protocol parameters such as pipelining, parallelism, and concurrency.

Pipelining targets the problem of transferring a large number of small files over the network [17,18,49]. In a regular TCP transfer, acknowledgement for the previous transfer is waited before starting the next transfer, which may cause a delay of more than one Round-Trip-Time (RTT) between individual transfers. With pipelining, multiple unacknowledged transfers can be active in the network pipe at any time and the delay between individual transfers is minimized.

Parallelism sends different chunks of the same file using different data channels (i.e., TCP streams) at the same time and achieves high throughput by mimicking the behavior of individual streams and getting a higher share of the available bandwidth [25,33,36]. On the other hand, using too many simultaneous connections congests the network and the throughput starts dropping down. Predicting the optimal parallel stream number for a specific setting is a very challenging problem due to the dynamic nature of the interfering background traffic.

Concurrency refers to sending multiple files simultaneously through the network using different data channels at the same time. Most studies in this area do not take the data size and the network characteristics into consideration when setting the concurrency level [31,50]. Liu et al. [35] adapt the concurrency level based on the changes in the network traffic, but do not take into account other bottlenecks that can occur on the end systems.

When used wisely, these techniques have a potential to improve the transfer performance at a great extent, but improper use of these parameters can also hurt the performance of the data transfers due to increased load at the end systems and congested links in the network. For this reason, it is crucial to find the best combination for these parameters with the least intrusion and overhead to the system resource utilization and power consumption.

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 [10,24,37]. 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. [29] analyzed how HTTP pipelining affects throughput in wireless local area networks (WLAN) and high-speed downlink packet access (HSDPA) networks. Results showed enabling pipelining eliminates the waiting time between successive requests and effectively increases the aggregated throughput. The fine-grained segmentation of data and using pipelining to fill the bandwidth-delay product (BDP) results in a natural adaption to network heterogeneity and increases interruption-free data transfers.

Natarajan et al. [39] showed that using a single HTTP stream for transferring independent web objects is very inefficient in high latency networks. Using concurrent HTTP channels for delivering such objects increases download rates and decreases browser response times by enabling concurrent rendering. Their work showed that the performance of using aggressive number of parallel HTTP flows depends on several factors such as available bandwidth, path latency, HTTP response size, and network congestion.

Robert et al. [32] used parallel HTTP requests to increase usage of available bandwidth during Internet video streaming. They analyzed video streaming performance in terms of video quality at various packet loss rates. Their results showed that at high packet loss rates using multiple HTTP channels with large sized chunks perform much better than a single HTTP channel without compromising TCP-friendliness. Li et al. [34] presented a parallel HTTP 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.

In prior work [6] ,we have analyzed the effects of different protocol parameters such as TCP pipelining, parallelism and concurrency levels on end-to-end throughput versus total energy consumption in the context of the GridFTP protocol [9], which is a fast, reliable and secure extension of FTP and widely used in the scientific computing community. We introduced three novel data transfer algorithms which achieve high data transfer throughput using GridFTP while keeping the energy consumption during the transfers at the minimal levels [7]. In [8], we have performed a preliminary study on the effects of these protocol parameters on the energy efficiency of HTTP data transfers.

In this paper, we introduce novel SLA-based HTTP data transfer algorithms which can decide the best combination of these parameters based on user-set energy efficiency and performance criteria. None of the existing work in this area proposed any energy-aware data transfer algorithms for high-performance HTTP data transfers with a quantification of the key practical tradeoffs between energy reduction and end-to-end performance of data transfers impacting user SLAs.

Table 1
Characteristics of the datasets used in the analysis.

Dataset	Data format	Avg. file size	Min–Max	Std. dev.
HTML	TEXT	102 KB	50 KB–150 KB	29.14 KB
Image	JPEG	2.4 MB	2 MB–3 MB	0.28 MB
Video	AVI	222 MB	202 MB–249 MB	14.78 MB

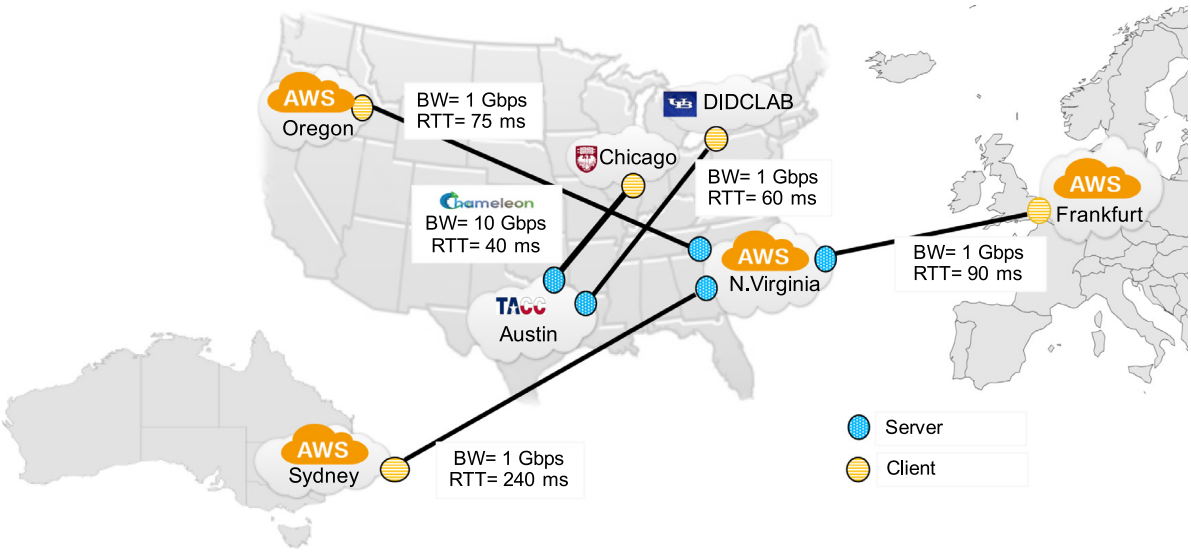

Fig. 2. Network map and specifications of the test environment.

3. Analysis of protocol parameter effects

Measuring the energy consumption of data transfers is a challenging task due to lack of hardware power monitoring options for most of the devices. Even though one can measure power consumption of a specific device using a power meter, it is not always possible to hook up power meters to all systems involved in a particular study, especially when they are remote servers operated by other entities.

In this work, we used two power models similar to those presented by Alan et al. [6] to estimate the server power consumption during data transfers for two different cases of access privileges: *(i)* the fine-grained power model requires access to utilization information of four system components: CPU, memory, disk and NIC; *(ii)* the CPU-based power model only needs utilization information of the CPU in the target system. Our approach resembles Mantis [15] in predicting power consumption of data transfers based on operating system metrics. It is non-intrusive, models the full-system power consumption, and provides real-time power prediction. It requires a one-time model-building phase to extract power consumption characteristics of the system components. For each system component (i.e. CPU, memory, disk, and NIC), we measure the power consumption values for varying load levels. Then, linear regression is applied to derive the coefficients for each component metric. The derived coefficients are used in our power model to predict the total power consumption of the data transfers.

These power models were used to analyze the power consumption of HTTP transfers at different levels of pipelining (pp), parallelism (p), and concurrency (cc) levels. Three different representative datasets were used during experiments in order to capture the throughput and power consumption differences based on the dataset type: *(i)* the HTML dataset is a set of raw HTML files from the Common Crawl project [2]; *(ii)* the image dataset is a set of photo files from Flicker [4]; and *(i)* the video dataset is a set of video file from Jiku [3]. The details of these datasets are presented in Table 1.

Apache HTTP server with default configuration and a custom HTTP client based on Apache HTTP Component libraries (which enables modification of protocol parameters) are used in the experiments. We also compared our client performance to two widely used command-line HTTP clients, wget and curl. Unfortunately these clients do not allow tuning of the protocol parameters. The experiments are conducted using two different server locations, five different client locations, and five different network settings. One of the servers is located at the Chameleon Cloud node in Austin, Texas (USA) and it is serving the clients at the DIDCLab in Buffalo, New York (USA) and at the University of Chicago, Illinois (USA). The second server is located at the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) node in North Virginia (USA) and it serving clients located at the AWS nodes in Oregon (USA), Frankfurt (Germany), and Sydney (Australia). The locations and specifications of the used servers and clients as well as the capacities of the links between them are presented in Fig. 2.

Fig. 3 presents the results of the data transfers between a web server in Virginia and a web client located in Sydney. We conducted these experiments between two AWS nodes where network bandwidth is 1 Gbps and RTT is 240 ms. We

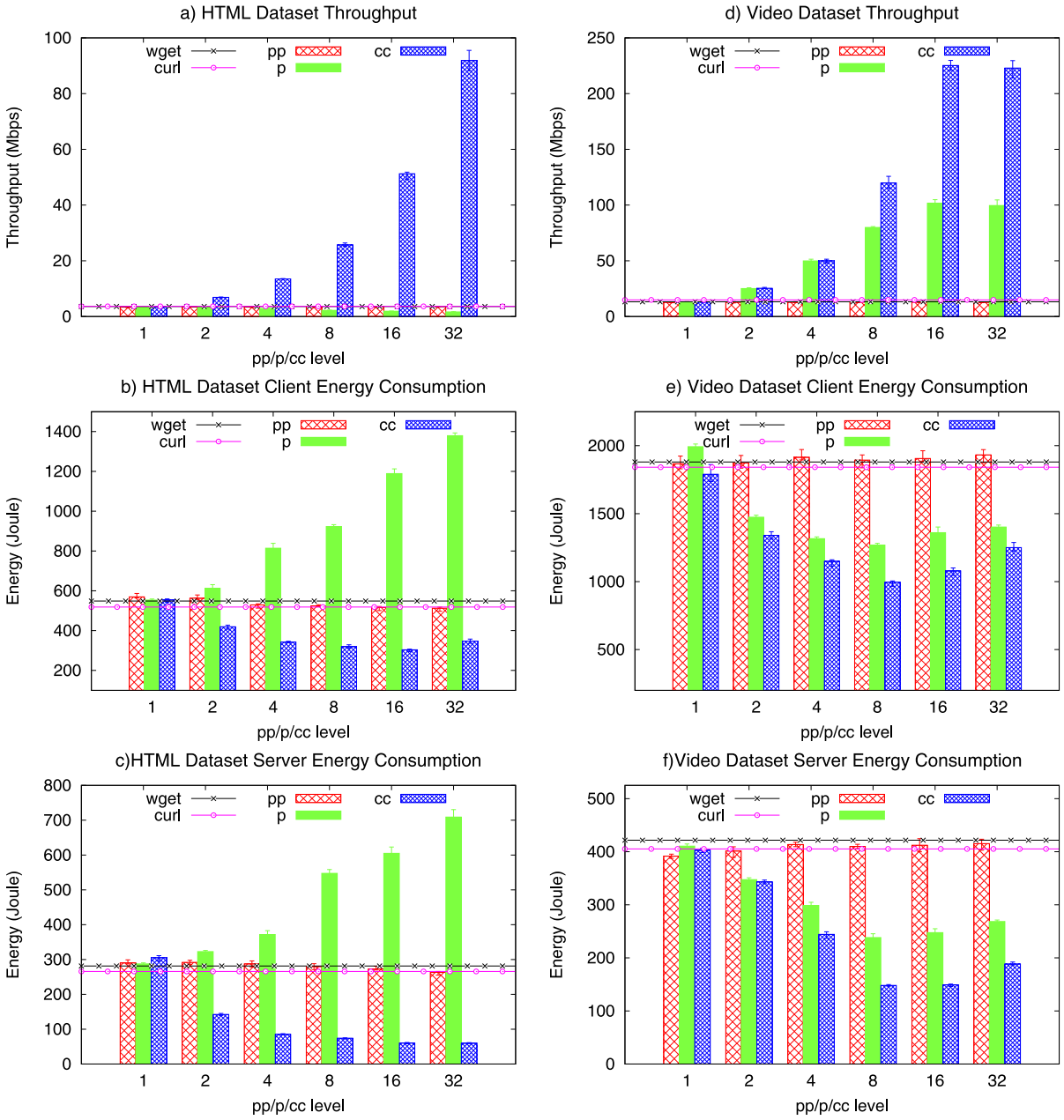

Fig. 3. Performance vs energy consumption trade-offs of HTTP transfers from a web server in Virginia to a client in Sydney.

transferred each dataset only changing one parameter (i.e., pipelining, parallelism, or concurrency) at a time to observe the individual effect of each one on the end-to-end throughput versus the end-system (client and server) energy consumption. Energy consumption of a data transfer is calculated using the two power models we mentioned at the beginning of this section. We repeated each experiment at least five times at various times and took average of them. Since there is no dedicated network between the source and destination servers, and a shared internet connection with long RTT is utilized, it is hard to achieve the maximum attainable network bandwidth. However, our results showed that we can achieve up to 240 Mbps throughput for the large (video) dataset.

Increased levels of pipelining and concurrency improve the end-to-end HTTP data transfer throughput and decrease end-system energy consumption for the small (HTML) and medium (image) sized datasets, which show very similar characteristics. Due to space limitations, we do not present the graphs for the image dataset. The results for the HTML dataset is presented in Fig. 3(a)–(c). Even though pipelining and concurrency lead to an increase in the instantaneous power consumption, decrease in the data transfer time overcomes and the overall energy consumption decreases. Pipelining can increase the throughput around 7% and 95% while decreasing energy consumption by 10% and 19% respectively from level 1 to 32, since BDP of this network is much bigger than the average file size of these datasets. For the video dataset, pipelining has almost no effect on throughput and energy consumption. Parallelism does not increase the throughput for the HTML and

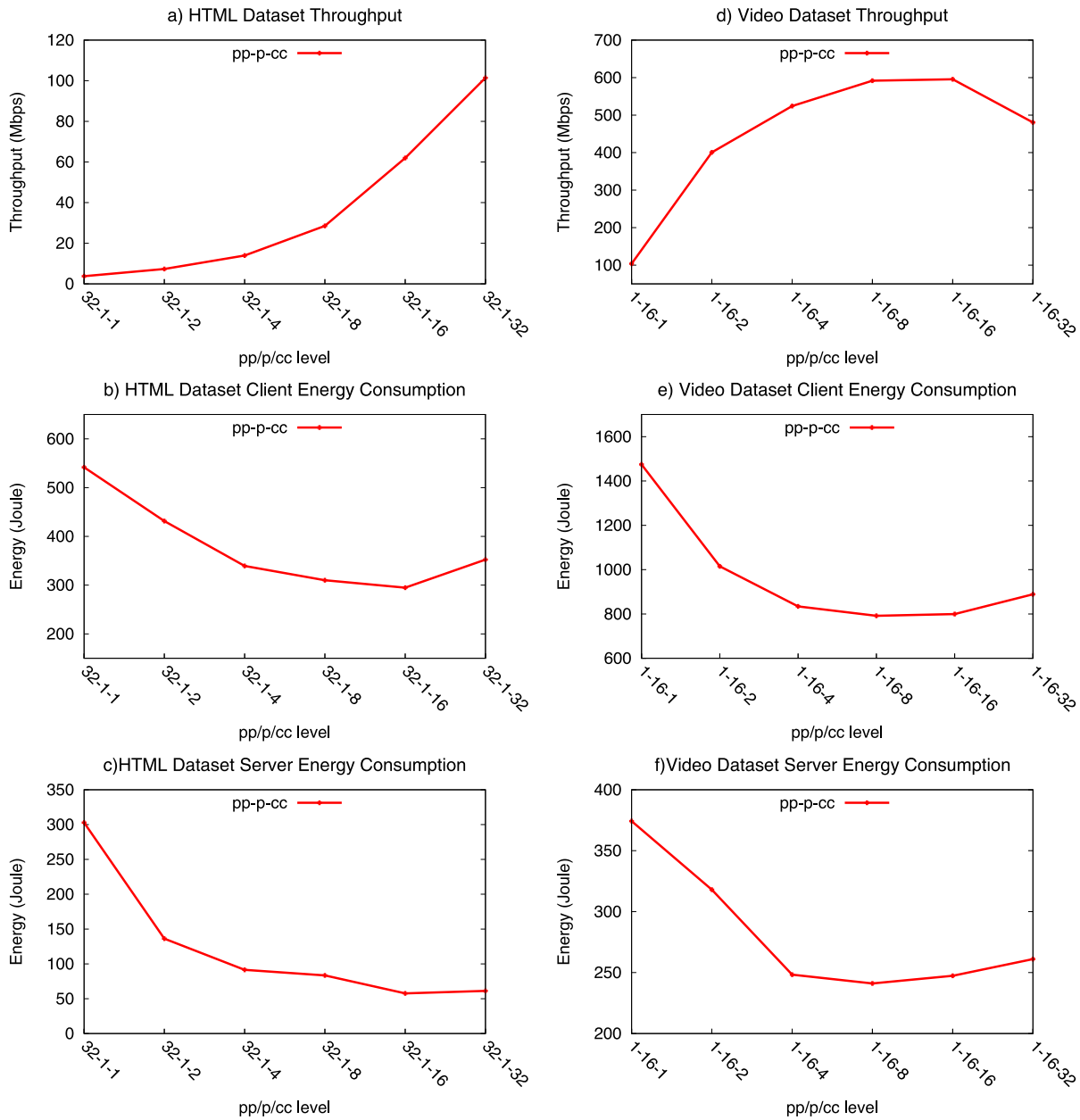

Fig. 4. Transfers with combined parameters between Sydney and Virginia.

image datasets in this network condition, but it increases the energy consumption by almost 2X. However for the video dataset, it increases the throughput and decreases the total energy consumption until reaching its optimal values. It increases the throughput by 6x while decreasing the energy consumption by 55% from level 1 to 8. We can conclude that, parallelism can improve the throughput and decrease the energy consumption of large-file dominated dataset transfers if it can be set to the optimal value. Same as the previous experiments, concurrency is the most effective parameter. As opposed to pipelining and parallelism, concurrency can increase the throughput and decrease the energy consumption considerably for all datasets. We reached the lowest energy consumption at level 16 for the HTML dataset with 45% energy gain, at level 8 for the image dataset with 28% energy gain, and at level 8 for the video dataset with 48% energy gain.

We also run transfers by using a combination of different parameters at the same time as shown in Fig. 4. The values of pipelining and parallelism are fixed to the values which returned the best throughput/energy ratio. This energy efficiency ratio quantifies how much data can be transferred at the cost of unit energy consumption. The transfer throughput significantly increases when a combination of different parameters are used compared to the case when they are used individually. For the HTML dataset, the combined parameter optimization increases the throughput by up to 20X while decreasing the energy consumption by up to 80%. For the image dataset, the throughput is increased by up to 10X, and the energy consumption is reduced by up to 50%. Finally, for the video dataset, the throughput is increased by up to 6X, and the energy

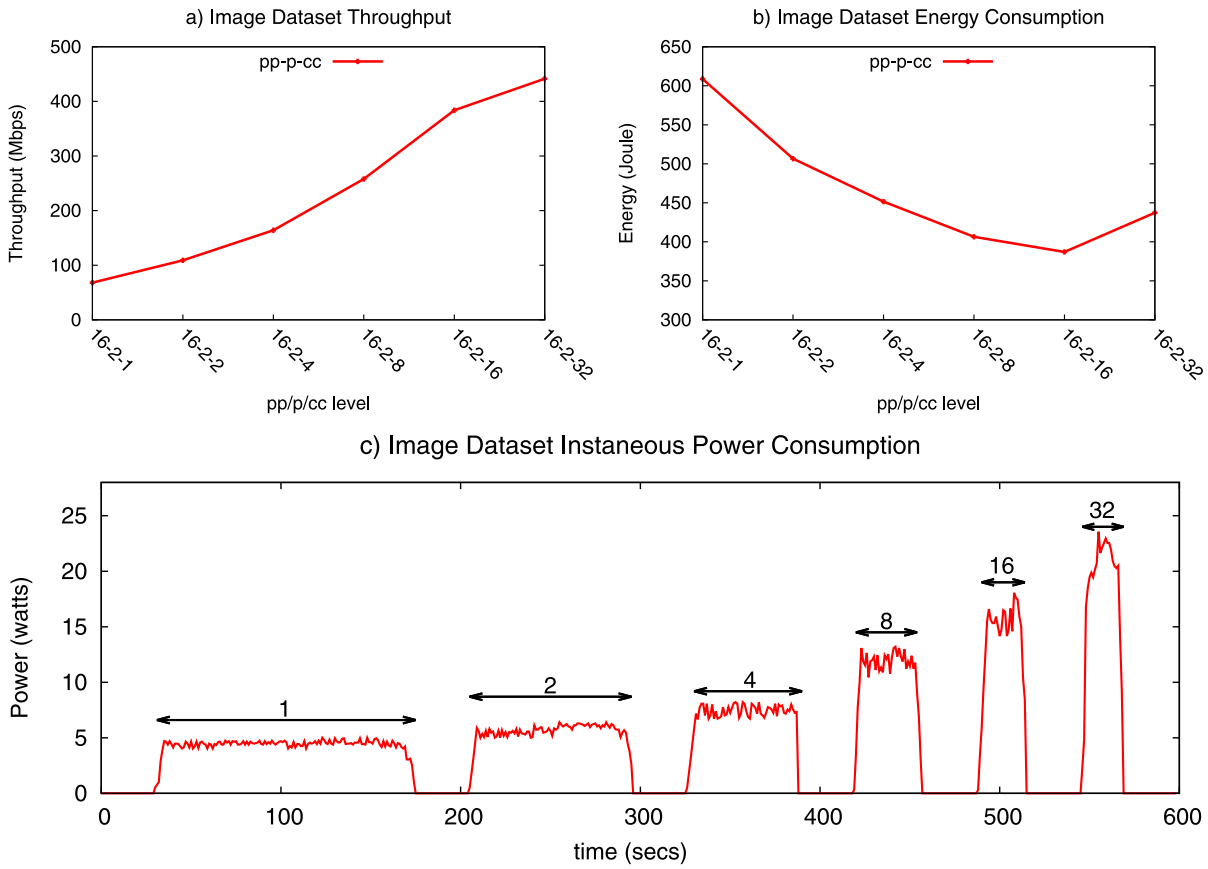

Fig. 5. Effect of combined parameters on (a) end-to-end throughput and (b) total energy consumption (for the image dataset transfers between Oregon and Virginia AWS EC2 nodes); (c) concurrency level versus instantaneous power consumption.

consumption is reduced by up to 40%. These results show that using a good combination for these three transfer parameters increases the end-to-end HTTP data transfer throughput drastically while reducing the total energy consumption at the sender and receiver nodes significantly.

In Fig. 5, we can see how the instantaneous power consumption changes during the image dataset transfer between Oregon and Virginia with combined parameters. We included only the image dataset transfer results in this case due to space limitations as the results for other datasets were quite similar. In this experiment, the pipelining level is fixed to 16, and the parallelism level is fixed to 2 (since these are the best values for this specific setting), whereas the concurrency level is changed between 1 and 32. When we increase the level of concurrency from 1 to 32, the average instantaneous power consumption increases from 4.2 W to 19.5 W, whereas the transfer time decreases from 145 seconds to 23 seconds. Thus overall energy consumption substantially decreases from 608 J to 437 J. This graph helps us to understand the nice balance between the increased instantaneous power consumption and the decreased transfer time, since both affect the total energy consumption of the system. As long as the energy gain due to the decreased transfer time is more than the loss due to the increased instantaneous power consumption, then we save energy at this system while increasing the throughput. But this is not always the case. In some transfers, we observe that although the throughput continues to increase, the total energy consumption would not continue to decrease, instead come to a balance and start increasing again. This is also what we observe in Fig. 5(b), when the concurrency level is increased from 16 to 32. The main reason for this is the server and network components are typically not energy proportional [11].

Lessons learned. The lessons learned from these experiments that should be harnessed while developing energy-aware data transfer algorithms can be summarized as follows: *(1)* Concurrency is the most effective and helpful parameter in terms of increasing the throughput and decreasing the total energy consumption, but the optimal concurrency value can be different depending on the dataset characteristics, end-system resources, and network conditions. Increasing the concurrency level does not always lead to energy savings and throughput gain, and estimating that fine point is a challenging task. *(2)* Pipelining leads to energy savings depending on the file size and BDP. If the average file size is less than BDP, pipelining increases the throughput and leads to energy savings. For larger file sizes, pipelining does not effect the throughput or energy consumption much. *(3)* High levels of parallelism can cause energy lost and throughput decline for small files but it is very effective for transferring large files such as the video dataset in our experiments.

4. SLA-based energy-aware throughput optimization algorithms

Service Level Agreements (SLA) have become more popular with the rapidly advancing and growing pay-as-you-go and low cost cloud computing services. As the usage of such systems increases, the guaranteed reliability and quality of the provided services become more important, which requires mutually agreed SLAs. Hence, we devised various HTTP data transfer algorithms based on different SLA requirements considering the lessons learned from our previous experiments.

4.1. SLA: Minimum energy consumption

The aim of this algorithm is to minimize the energy consumption of data transfers by tuning the analyzed transfer parameters. The SLA: Minimum Energy (MinE) algorithm transfers files with minimum energy consumption without considering any performance criteria. In other words, we offer an option for the users who do not need to finish the transfers in the time pressure. We believe this approach will be helpful for background transfers and update procedures which do not require any time limitations.

Instead of using the same parameter combination for the whole data set, which can increase the power consumption unnecessarily, we initially group files into three subgroups; *Small, Medium* and *Large* based on the file sizes using a clustering algorithm such as DBSCAN [47]. Then, we calculate the best possible parameter combinations for each subgroup in which the Bandwidth-Delay-Product (BDP), average file size, and TCP buffer size are taken into consideration. Pipelining and concurrency are the most effective parameters for small file transfers, so it is especially important to choose the best pipelining and concurrency values for such transfers. Pipelining value is calculated by dividing BDP to the average file size of the group which returns large values for *Small* files. By setting pipelining to relatively high values, we are transferring multiple data packets back-to-back, which in turn prevents idleness of the network and system resources, and decreases the energy consumption. As the average file size of the subgroups increases, pipelining value is set to smaller values since it does not further improve the data transfer throughput. It could even cause a performance degradation and redundant power consumption by poorly utilizing the network and system resources. Moreover, we assign most of the available data channels to the *Small* subgroup which multiplies the impact of energy saving when combined with pipelining. Besides pipelining, minimum energy algorithm also tries to keep the concurrency level of *Large* files at minimum since using more concurrent channels for large files causes more power consumption. For the parallelism level, we again consider TCP buffer size, BDP, and average file size. The equation will return small values for *Small* files which will avoid creating unnecessarily high number of threads and thus prevents redundant power consumption. The parallelism level for *Medium* and *Large* subgroups will be high if the system buffer size is not large enough to fill the network channel bandwidth.

4.2. SLA: Maximum achievable throughput

In the SLA: Maximum achievable throughput (MaxThr) algorithm, the focus is mainly maximizing the overall throughput without considering power consumption. As we mentioned in Section 3, even after choosing the best parameter combination for each file, the throughput obtained during the transfer of the small files is significantly lower compared to the large files due to the high overhead of reading too many files from disk and under utilization of the network pipe. Hence, keeping the balance between large and small files in the transfer of heterogeneous datasets is important for achieving the maximum throughput.

Similar to MinE, MaxThr algorithm groups files into three subgroups according to the file size and calculates the best possible parameter values for pipelining and parallelism using BDP, TCP buffer size and average file size of the subgroup. The algorithm distributes data channels among subgroups using round-robin algorithm in the order of Large-Medium-Small. After channel distribution is completed, algorithm transfers subgroups concurrently using the calculated concurrency level for each subgroup. The algorithm continues to check the channel distribution until completion of all file transfers. When the transfer of all files in a subgroup is completed, the channels of the subgroup are scheduled for another subgroup based on same round-robin order.

4.3. SLA: Maximum energy efficiency

Maximum energy efficiency is defined as using the least amount of energy possible to provide the same level of service. When we refine this general definition to the data transfer service, energy efficient data transfer means transferring data with less energy providing the same or higher throughput rates compared to the regular transfers. Hence, the main purpose of SLA: Maximum Energy Efficiency algorithm (EE) is finding the best possible parameter configuration to reach maximum achievable throughput with minimum energy consumption. The algorithm does not only focus on minimizing the energy consumption or maximizing the throughput, rather it aims to find high performance and low power consumption concurrency levels within defined transfer channel range. While the minimum value for the concurrent number of the transfer channels is 1 for all environments, the maximum value might be different due to variability of end-system resource capacities and fairness concerns. Thus, we let the user to be able to decide the maximum acceptable number of concurrent channels for a data transfer. We have chosen concurrency to leverage the throughput and energy consumption, since in our

previous experiments we have observed that concurrency is the most influential transfer parameter for all file sizes in most of the experimented settings.

After estimating the optimal pipelining and parallelism values with previous equations, we also calculated proportions for each subgroup based on total size and the number of files. Given the maximum allowed channel count, proportions are used to determine the number of channels to be allocated for each subgroup. For example, if we have a dataset dominated by small files, then assigning equal number of channels to subgroups would cause sub-optimal transfer throughput since large files will be transferred faster than smaller files and the average transfer throughput will be subjected to small subgroup's throughput. To solve this problem, we initially calculate the proportions of the subgroups and then allocate channels to the subgroups accordingly.

The EE algorithm starts to transfer files with one active channel, and then increases the channel count until reaching the user-defined maximum channel count. Instead of evaluating the performance of all concurrency levels in the search space, EE decreases the search space by incrementing the concurrency level by four each time. Each concurrency level is run for five-second time intervals and then the power consumption and throughput of each interval are calculated. Once EE examines throughput/energy ratio of the concurrency levels tested in the search space, it picks the concurrency level with maximum throughput/energy ratio to transfer the rest of the files.

4.4. SLA: Flexible throughput by energy efficient approach

In addition to the previous algorithms, the users may want to transfer data within flexible throughput requirements by using an energy efficient approach. Hence, we devised the SLA: Flexible Throughput (FlexibleThr) algorithm which lets the users to define their minimum throughput requirements as a percentage of the maximum achievable throughput in a given transfer environment. The FlexibleThr algorithm takes the desired minimum throughput value as input and aims to achieve that target throughput with minimum energy consumption possible by tuning the transfer parameters. While keeping the quality of service (transfer throughput in this case) at the desired level, FlexibleThr algorithm uses a technique similar to the MinE algorithm to keep the power consumption at the minimum possible level.

The FlexibleThr algorithm initiates the transfer with one channel, and if the throughput is less than the target throughput, it increases the required concurrency level by comparing the current and target throughput values. FlexibleThr continues to check the throughput every five seconds, and if the throughput is still less than the target throughput at the estimated concurrency level, then additive increase is applied to reach the target throughput until it reaches or exceeds the target. When assigning the transfer channels to the subgroups, FlexibleThr gives priority to the small files similar to the MinE algorithm due to the energy consumption concerns.

5. Experimental results

Similar to the analysis of protocol parameter experiments, the developed algorithms were tested on the Chameleon Cloud, DIDCLab and AWS EC2 instances as presented in Fig. 2. In the experiments, we used two different datasets due to the different capacities of the utilized networks. For the 10 Gbps links, a 32 GB dataset was used; and for the 1 Gbps links, a 7 GB dataset was used. For both datasets, the file sizes range between 150 KB and 250 MB. The same Apache HTTP server and the custom HTTP client configurations were used during the experiments. We compared the performance of our algorithms with energy-agnostic wget and curl clients as well as our Apache web client at default setting (without any tuning).

We evaluated the performance of MinE and MaxThr algorithms at different concurrency levels defined by the user. On the other hand, EE algorithm takes upper bound for the concurrency level but finds the optimal level itself during its search phase. Since wget and curl tools do not allow to open multiple channels, their performance are independent of the user-defined maximum level of concurrency. For our custom HTTP client, we tested it on default (w/o tuning) mode and considered it as a base performance level for a given data transfer. Thus, its performance is also independent of the concurrency level.

Fig. 6 shows the results of the algorithms running on the Chameleon Cloud testbed. The web server is located at an instance in Austin, TX and the web client is located at an instance in Chicago, IL. The network bandwidth between the server and client is 10 Gbps, and the round trip time is 40 ms. The algorithms are compared based on their achieved throughput, energy consumption, and energy efficiency numbers. In accordance with their objective, MaxThr always achieves highest throughput and MinE achieves lowest energy consumption almost at all concurrency levels. Interestingly, while the performance of MinE is five times greater than wget, curl, and w/o tuning mode at concurrency level 8, it also consumes around 60% less energy than these tools as shown in Fig. 6. This strengthens the idea that in some cases the energy consumption can be decreased and the data transfer throughput can be increased at the same time. Even with concurrency level 1, our algorithms perform better than other tools. The achievement stems from using optimal pipelining and parallelism values starting from initial configuration. It also shows that even concurrency is the most influential parameter, parallelism and pipelining can still make a difference in the throughput and energy consumption, thus should be tuned carefully. EE aims to find the sweet spot where the throughput/energy (energy efficiency) ratio is maximized. It searches the concurrency level in the search space, bounded by 1 and *maxChannels*, after which the throughput increase is surpassed by the energy consumption increase. As given in Algorithm 3, it evaluates the throughput/energy ratio of multiple concurrency levels and picks the concurrency level with the highest throughput/energy ratio and runs the rest of the transfer with it. Compared to

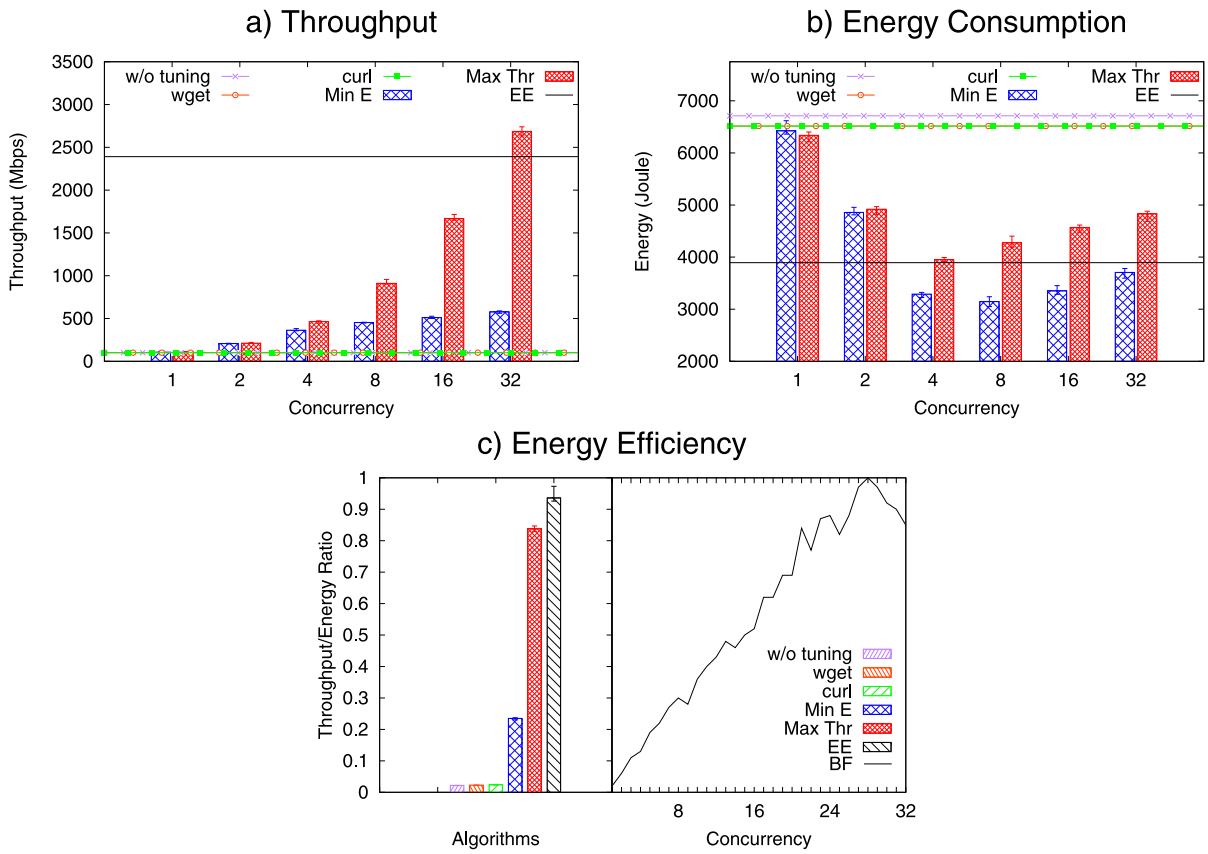

Fig. 6. HTTP transfers from a web server in Austin to a client in Chicago.

Algorithm 1 — SLA: Minimum energy consumption.

Require: *ChannelCount*
  $GroupFiles(filelist, BDP)$
  for *each group small :: large* **do**
    $pipelining = \left\lceil \frac{BDP}{avgFileSize} \right\rceil$
    $parallelism = Min(\left\lceil \frac{BDP}{bufSize} \right\rceil, \left\lceil \frac{avgFileSize}{bufSize} \right\rceil)$
    $concurrency = Min(\left\lceil \frac{BDP}{avgFileSize} \right\rceil, \lceil \frac{ChannelCount+1}{2} \rceil)$
    $ChannelCount = ChannelCount - concurrency$
  end for
  $startTransfer(groups)$

Algorithm 2 — SLA: Maximum achievable throughput.

Require: *ChannelCount*
  $GroupFiles(filelist, BDP)$
  $calculateOptimalParameters()$
  for $i = 0$ to *ChannelCount* **do**
    if $i\% = 0$ **then**
      $large\ concurrency++$
    else if $i\% = 1$ **then**
      $medium\ concurrency++$
    else
      $small\ concurrency++$
    end if
  end for
  $startTransfer(groups)$

Algorithm 3 — SLA: Maximum-energy-efficiency.

Require: *MaximumChannelCount*
  GroupFiles(*filelist*, *BDP*)
  calculateOptimalParameters()
  calculateProportions{Calculate proportional weight of each subgroup}
  Channel = 1
  while *Channel* <= *MaximumChannelCount* **do**
    transfer(*Channel*)
    energyConsumption = *calculate*(*SystemMetrics*)
    throughput = *calculateThroughput*(){Calculate energy and throughput for 5 secs period.}
    $energyEfficiency[i] = \frac{throughput}{energyConsumption}$
    Channel+ = 4
  end while
  transfer(*EnergyEfficientConcurrencyLevel*)

Algorithm 4 — SLA: Flexible throughput by energy efficient approach.

Require: *TargetThroughput*
  GroupFiles(*filelist*, *BDP*)
  calculateOptimalParameters()
  startTransfer()
  Throughput = *calculateThroughput*(){Calculate throughput for every 5 secs period.}
  if *Throughput* <= *targetThroughput* **then**
    concurrency = *targetThroughput*/*actThroughput*
  end if
  while *Throughput* <= *targetThroughput* **do**
    concurrency + +
  end while

MaxThr, EE consumes 25% less energy in trade off 10% less throughput for concurrency level 32 at which MaxThr achieves the highest throughput. At concurrency level 4, MaxThr consumes similar amount of energy while obtaining 4 times less throughput compared to EE which can justify the argument of consuming less amount of energy without sacrificing the data transfer throughput.

In order to compare the performance of the EE algorithm to the ideal case, we ran a brute-force search (BF) algorithm to find the concurrency level which maximizes the throughput/energy ratio. BF is a revised version of the EE algorithm in a way that it skips the search phase and runs the transfer with pre-defined concurrency levels. BF search range is bounded by 32 since the throughput/energy ratio follows a steadily decreasing pattern after around a value of 28 as shown in Fig. 6 (c). The concurrency level which yields the highest throughput/energy ratio is considered as the best possible value under these conditions, and the throughput/energy ratio of all other algorithms are compared to this value. As a result, we observe that the concurrency level chosen by EE can yield as much as 94% throughput/energy efficiency compared to the best possible value obtained by BF. On the other hand, while MinE is successful in consuming the least amount of energy for each concurrency level, it can only reach around 25% of the best possible throughput/energy ratio.

We also tested our algorithms using Amazon's EC2 nodes located in Frankfurt and Virginia as shown in Fig. 7. Between these two nodes, the network bandwidth is 1 Gbps, and the round trip time is 90 ms. The wget, curl, and w/o tuning mode again yield the lowest throughput and consume most energy due to lack of parameter tuning. Additionally, MaxThr achieves highest throughput and MinE achieves lowest energy consumption almost at all concurrency levels. MinE consumes minimum energy at concurrency level 16 and decreases energy consumption 75% while increasing throughput three times with respect to the base level. Although the energy consumption of MaxThr and EE are close to each other at concurrency level 8, the throughput values differ considerably and EE achieves 2 times more throughput compared to MaxThr. While MaxThr's throughput increases as the concurrency level increases, its power consumption curve follows a parabolic pattern and reaches the minimum point at concurrency level 4. One of the reasons for this observed pattern is that the transfer instances used on AWS have four cores and energy consumption per core decreases as the number of active cores increases [5]. When concurrency level goes above 4, then each core starts running more number of threads and this leads to an increase in energy consumption per core. When we look at the energy efficiency graph, EE algorithm can reach as much as 92% of the optimal value while MaxThr reaches 83%, MinE reaches 26%, and other tools can not exceed 10% level.

Finally, we evaluated our SLA based algorithms between a web server at the Chameleon Cloud node in Austin,TX and the web client at DIDCLab in Buffalo,NY as presented in Fig. 8. Similar to the previous environments, MinE and MaxThr realize their goals. MaxThr algorithm increases throughput from 37 Mbps to 290 Mbps despite the shared low-bandwidth network connection between the server and client; and MinE algorithm decreases energy consumption 85% with respect to

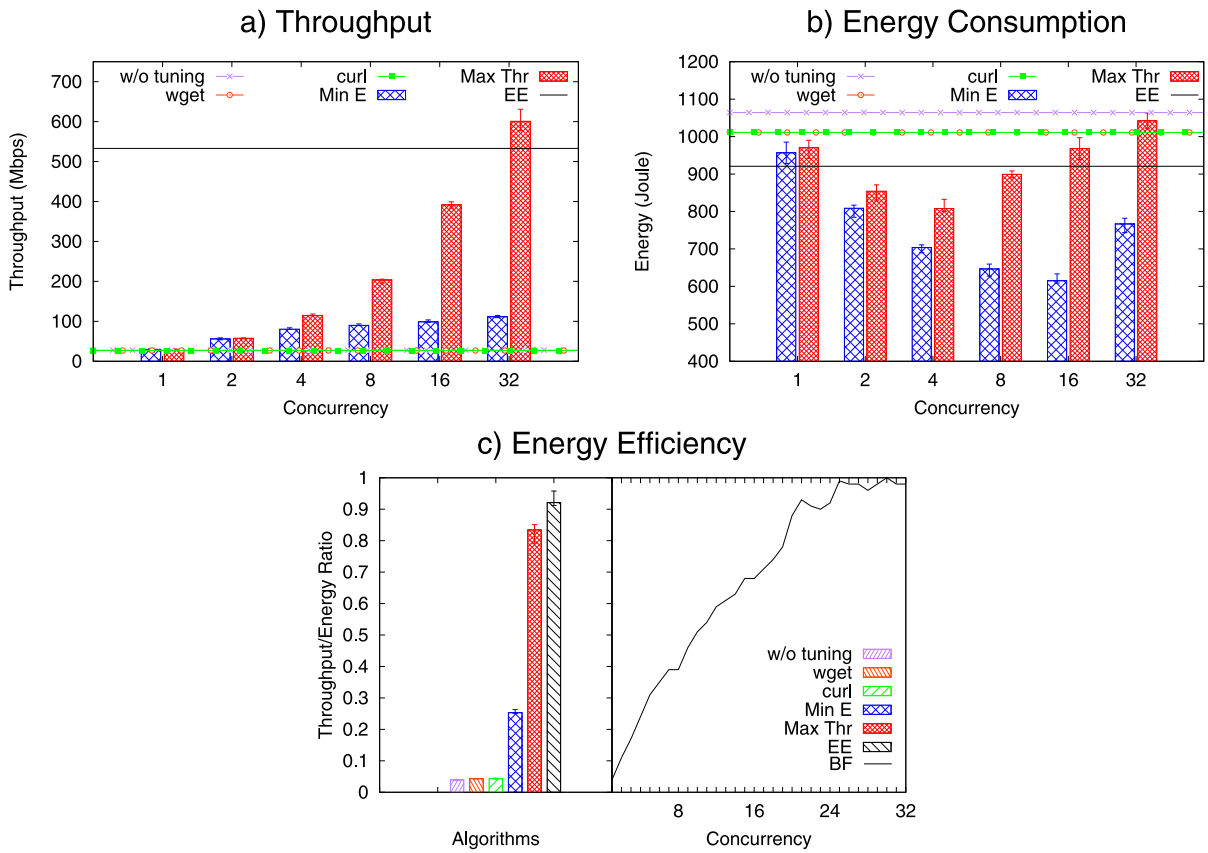

Fig. 7. HTTP transfers from a web server in Virginia to a client in Frankfurt.

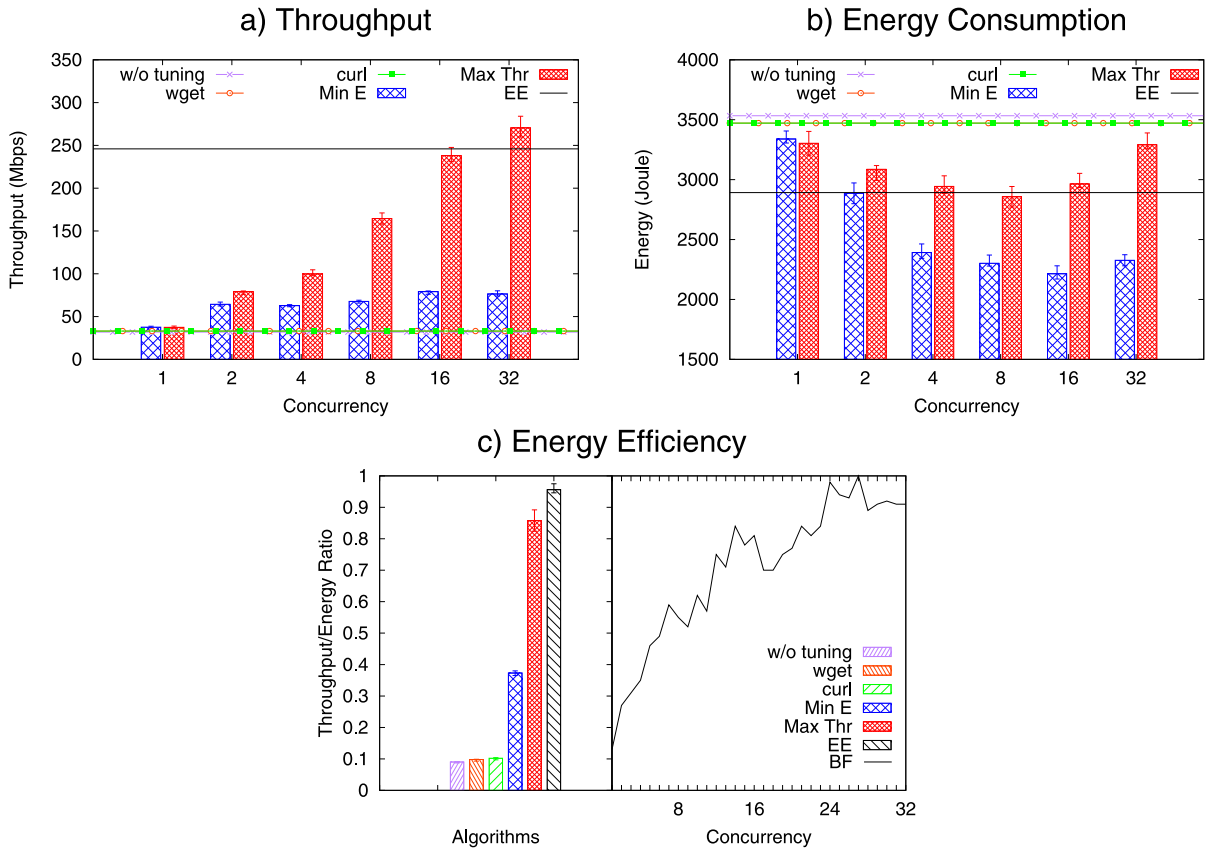

Fig. 8. HTTP transfers from a web server in Austin to a client in Buffalo.

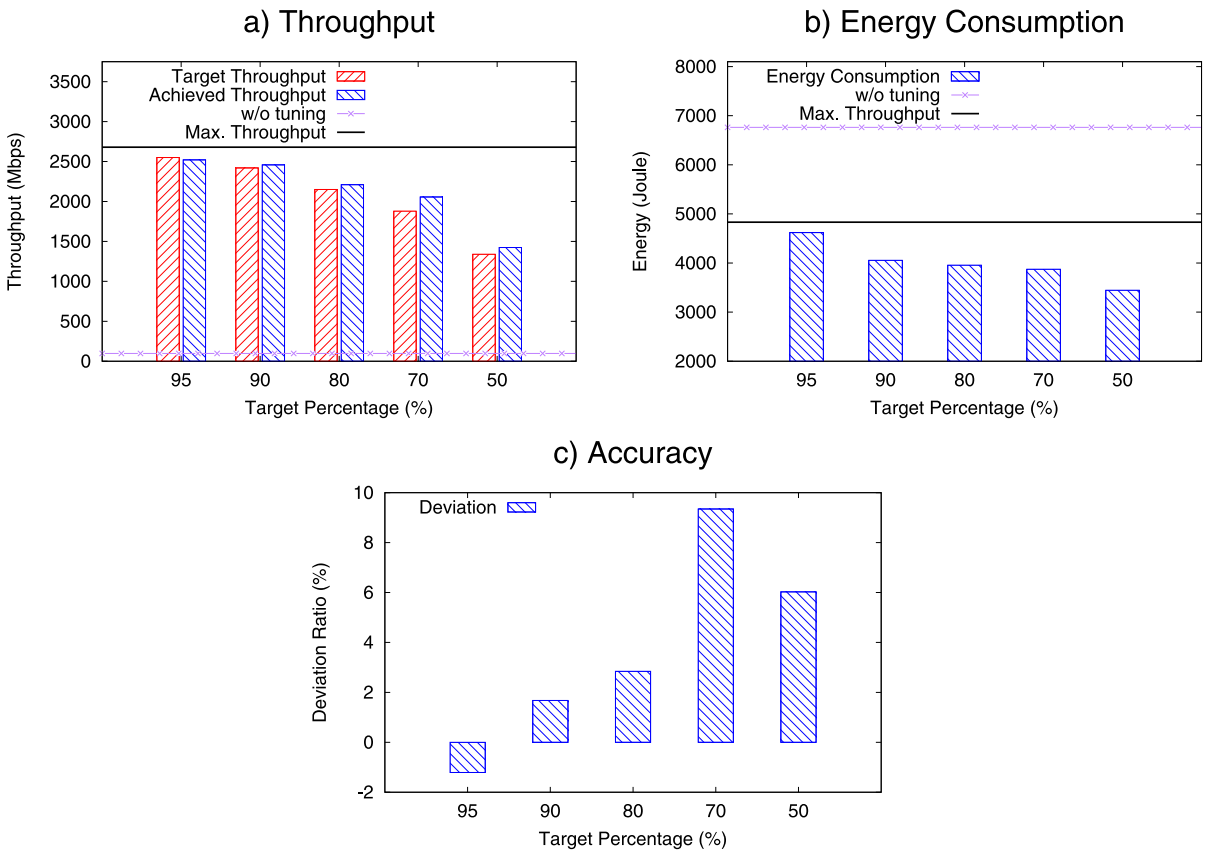

Fig. 9. Flexible throughput experiments from a web server in Austin to a client in Chicago.

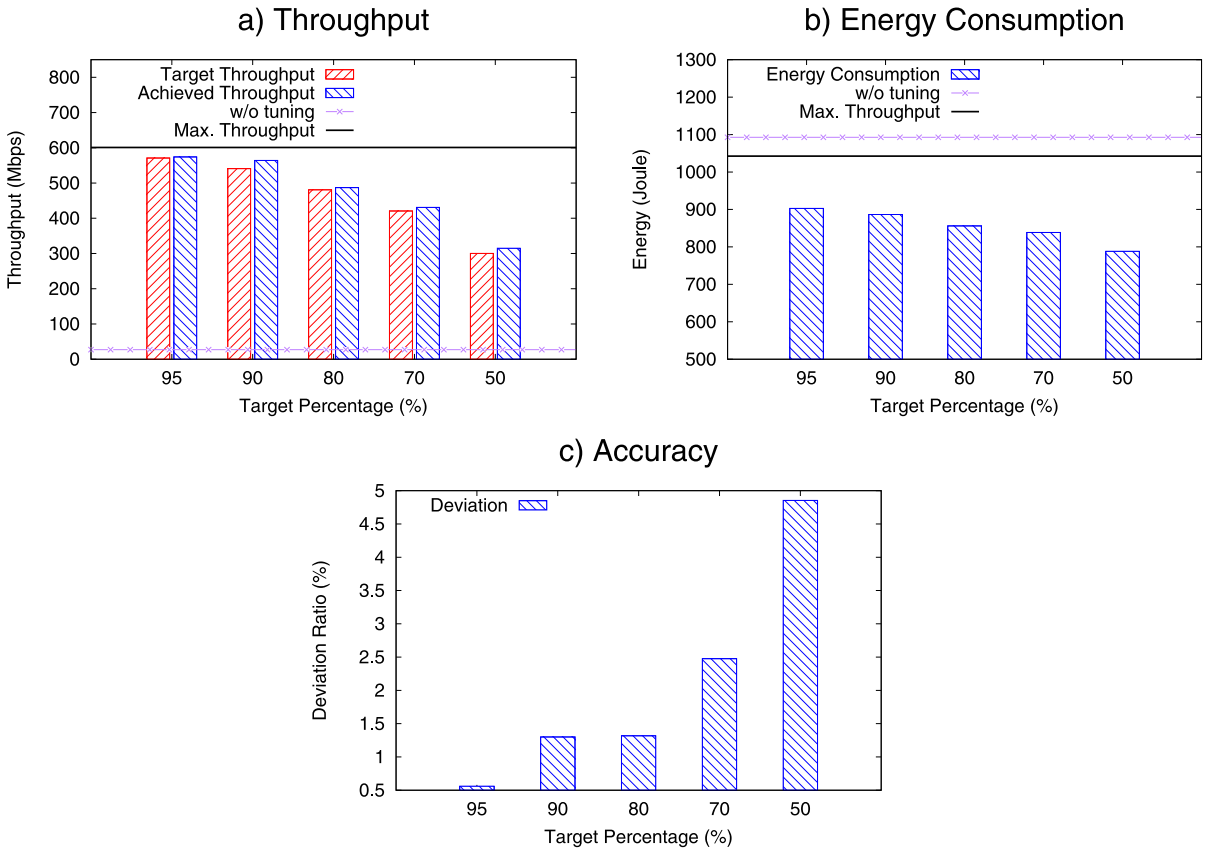

Fig. 10. Flexible throughput experiments from a web server in Virginia to a client in Frankfurt.

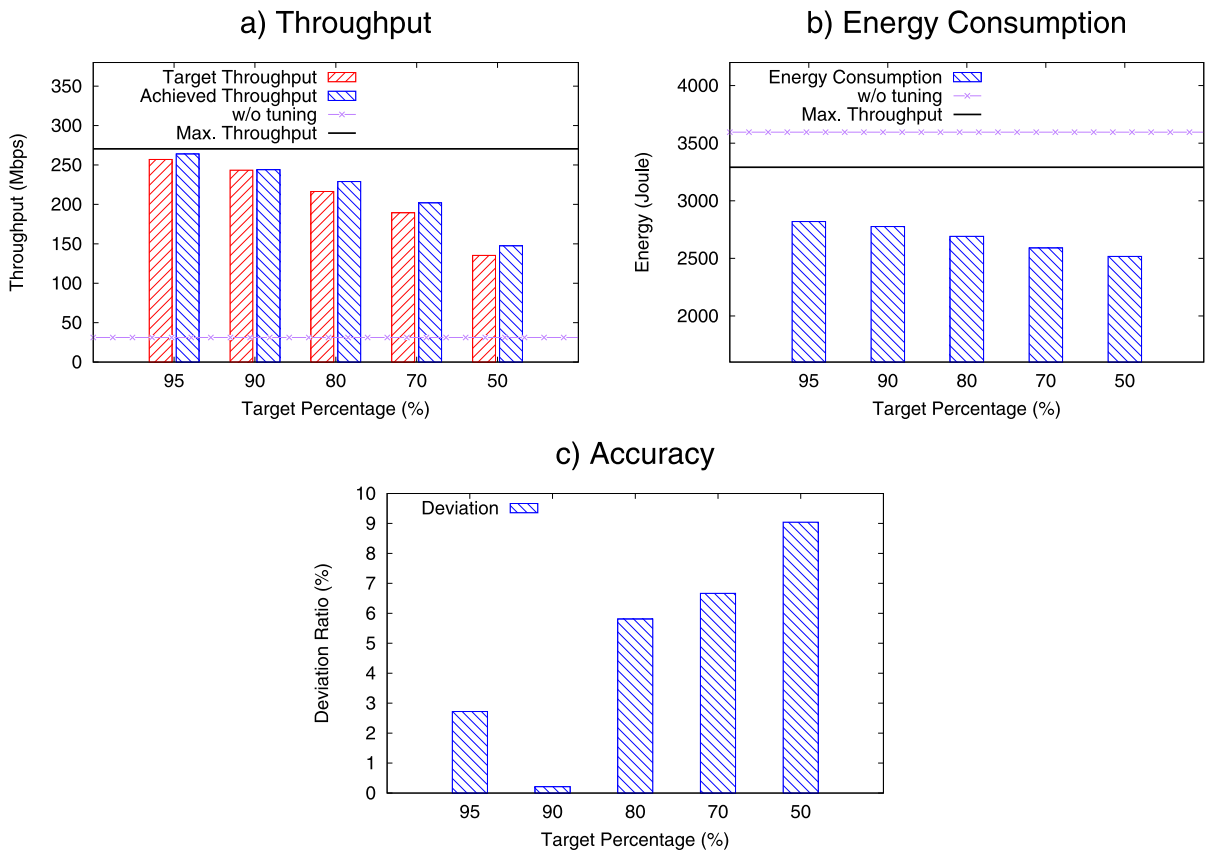

Fig. 11. Flexible throughput experiments from a web server in Austin to a client in Buffalo.

the base power consumption. Unlike the other experiments, MinE and MaxThr achieve their best throughput/energy ratio at comparatively low concurrency levels in this setting. This strengthens the motivation behind EE as it is designed to capture the network-specific sweet spots in which throughput/energy ratio is maximized.

We tested the FlexibleThr algorithm with different levels of throughput targets proportioned to the maximum throughput achieved by the MaxThr algorithm in this specific environment. In FlexibleThr, x% target percentage means that the algorithm tries to achieve a transfer throughput of at least x% of the maximum throughput possible (i.e., with at most 100% – x% performance loss). For example, maximum throughput achieved by MaxThr in Chameleon Cloud network with 10G connection is around 2600 Mbps, so 95% target percentage corresponds to at least 2470 Mbps transfer rate.

FlexibleThr is able to deliver all various throughput requests except 95% target throughput percentage at the Chameleon Cloud with 10G network since FlexibleThr is unable to reach the requested throughput on this network even after reaching the maximum level of concurrency. As presented in Fig. 6(b), 7(b) and 8(b) after some specific concurrency level, energy consumption starts to increase. Thus, achieving target throughput with minimum possible concurrency level will minimize energy consumption. Figs. 9(a), 10(a) and 11(a) show how close our FlexibleThr algorithm gets to the target throughput specified by the user under different network conditions. FlexibleThr is able to achieve all throughput expectations within 10% deviation rate as shown in Figs. 9(c), 10(c) and 11(c). Since the effect of each new channel creation on throughput is higher when the total number of channels is small, accuracy decreases as FlexibleThr is expected to provide low transfer throughput. Figs. 9(b), 10(b) and 11(b) depict energy consumption comparison between MaxThr, w/o tuning mode, and FlexibleThr algorithm. FlexibleThr can deliver requested throughput while decreasing the energy consumption by up to 50% for Chameleon Cloud with 10G interconnection, up to 38% for AWS transfers, and up to 42% for Chameleon Cloud and DIDCLab transfers respectively. Finally, if customers are flexible in transferring their data with some reasonable delay, FlexibleThr algorithm helps the service providers to save from the energy consumption considerably and also the service providers can possibly offer low-cost and flexible data transfer options to their customers in return of delayed transfers.

6. Conclusion

In this paper, we analyzed different application-layer parameters that affect both the throughput and the energy consumption of HTTP, which is the de-facto transport protocol for Web services. Different parameters such as pipelining, parallelism and concurrency levels play an important role in the achievable network throughput and the total energy consumption. We introduced SLA-based algorithms which can decide the best combination of these parameters based on user-set

energy efficiency and performance criteria. Our experimental results show that significant amount of energy savings (up to 80%) can be achieved at the sending and receiving nodes during data transfers with no or minimal performance penalty if the correct parameter combination is used. In some cases, the throughput can be maximized and the energy consumption can be minimized at the same time. With the help of our SLA-based energy-efficient data transfer algorithms, the Internet service providers will be able to minimize the energy consumption during data transfers without compromising the SLA with the customer in terms of the promised performance level, but still execute the transfers with minimal energy levels given the requirements.

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 OCR-W1771224. We also would like to thank Chameleon Cloud and AWS for letting us use their resources for some of the experiments presented in this paper.

References

[1] IEEE energy efficient ethernet standards, 2010, (doi:10.1109/IEEESTD.2010.5621025).
[2] Common crawl, 2018, (http://commoncrawl.org).
[3] Jiku mobile video dataset, 2018, (http://www.jiku.org/datasets.html).
[4] Yahoo flickr creative common images, 2018, (http://webscope.sandbox.yahoo.com/).
[5] I. Alan, E. Arslan, T. Kosar, Energy-aware data transfer tuning, in: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, IEEE, 2014a, pp. 626–634.
[6] I. Alan, E. Arslan, T. Kosar, Energy-Performance trade-offs in data transfer tuning at the end-Systems, Sustain. Comp. Inform. Syst.J. 4:4:318-329 (2014b).
[7] I. Alan, E. Arslan, T. Kosar, Power-aware data scheduling algorithms, in: Proc. IEEE/ACM SC'15, 2015.
[8] I. Alan, T. Kosar, Energy-aware http data transfers, in: Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on, IEEE, 2016, pp. 37–42.
[9] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, The globus striped gridftp server, in: Proc. IEEE SC'05, 2005, p. 54.
[10] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, S. Tuecke, Software as a service for data scientists, Communications of the ACM 55:2 (2012) 81–88.
[11] L.A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer (Long Beach Calif) (12) (2007) 33–37.
[12] D. Brooks, V. Tiwari, M. Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, in: Proc. of ISCA'00, ACM, New York, NY, USA, 2000, pp. 83–94.
[13] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, S. Wright, Power awareness in network design and routing, in: In Proc. of IEEE INFOCOM, April, 2008, 2008.
[14] J. Czyz, M. Allman, J. Zhang, S. IekelJohnson, E. Osterweil, M. Bailey, Measuring ipv6 adoption, SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 87–98.
[15] D. Economou, S. Rivoire, C. Kozyrakis, P. Ranganathan, Full-system power analysis and modeling for server environments, in: Proc. of Workshop on Modeling, Benchmarking, and Simulation, 2006.
[16] X. Fan, W.-D. Weber, L.A. Barroso, Power provisioning for a warehouse-sized computer, ACM SIGARCH Comp. Archit. News 35 (2) (2007) 13–23.
[17] K. Farkas, P. Huang, B. Krishnamurthy, Y. Zhang, J. Padhye, Impact of tcp variants on http performance, Proc.High Speed Netw. 2 (2002).
[18] N. Freed, SMTP service extension for command pipelining, 2018, (http://tools.ietf.org/html/rfc2920).
[19] W. Fu, T. Song, A frequency adjustment architecture for energy efficient router, ACM SIGCOMM Comp. Comm. Rev. 42 (4) (2012) 107–108.
[20] E. Goma, M.C.A.L. Toledo, N. Laoutaris, D. Kosti, P. Rodriguez, R. Stanojev, P.Y. Valentin, Insomnia in the access or how to curb access network related energy consumption, in: Proc. of ACM SIGCOMM 2011, 2011.
[21] A. Greenberg, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta., Towards a next generation data center architecture: Scalability and commoditization, in: In ACM PRESTO, pages 57-2, 2008, 2018.
[22] M. Gupta, S. Singh, Greening of the internet, in: ACM SIGCOMM, pages 19-6, 2003, 2003.
[23] M. Gupta, S. Singh, Energy conservation with low power modes in ethernet lan environments, in: IEEE INFOCOM (MiniSymposium) 2007, 2007.
[24] T.J. Hacker, B.D. Noble, B.D. Atley, The end-to-end performance effects of parallel tcp sockets on a lossy wide area network, in: Proc. of IPDPS '02, IEEE, 2002, p. 314.
[25] T.J. Hacker, B.D. Noble, B.D. Atley, Adaptive data block scheduling for parallel streams, in: Proc. of HPDC '05, ACM/IEEE, 2005, pp. 265–275.
[26] K. Hasebe, T. Niwa, A. Sugiki, K. Kato, Power-saving in large-scale storage systems with data migration, in: IEEE CloudCom 2010, 2010.
[27] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: Saving energy in data center networks, in: Proc. of NSDI 2010, 2010.
[28] D.A. Joseph, A. Tavakoli, I. Stoica, A policy-aware switching layer for data centers, in: SIGCOMM CCR 38(4):51-62, 2008, 2008.
[29] D. Kaspar, K. Evensen, P. Engelstad, A.F. Hansen, Using http pipelining to improve progressive download over multiple heterogeneous interfaces, in: Proc. of ICC, IEEE, 2010, pp. 1–5.
[30] R. Kohavi, R.M. Henne, D. Sommerfield, Practical guide to controlled experiments on the web: Listen to your customers not to the hippo, in: In Proceedings of KDD 2007, 2007.
[31] T. Kosar, M. Balman, A new paradigm: data-aware scheduling in grid computing, Future Generat. Comp. Syst. 25 (4) (2009) 406–413.
[32] R. Kuschnig, I. Kofler, H. Hellwagner, Improving internet video streaming performance by parallel tcp-based request-response streams, in: Proc. of CCNC, IEEE, 2010, pp. 1–5.
[33] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, S. Tuecke, Applied techniques for high bandwidth data transfers across wide area networks, in: Proc. of CHEP 2015, 2015.
[34] C. Liu, I. Bouazizi, M. Gabbouj, Parallel adaptive http media streaming, in: Proc. of ICCCN, IEEE, 2011, pp. 1–6.
[35] W. Liu, B. Tieman, R. Kettimuthu, I. Foster, A data transfer framework for large-scale science experiments, in: Proc. of DIDC Workshop, 2010.
[36] D. Lu, Y. Qiao, P.A. Dinda, Characterizing and predicting tcp throughput on the wide area network, in: Proc. of ICDCS '05, 2005a.
[37] D. Lu, Y. Qiao, P.A. Dinda, F.E. Bustamante, Modeling and taming parallel tcp on the wide area network, in: Proc. of IPDPS, 2005b, p. 68b.
[38] P. Mahadevan, P. Sharma, S. Banerjee, P. Ranganathan, A power benchmarking framework for network devices, in: Proc. of IFIP Networking, May 2009, 2009.
[39] P. Natarajan, F. Baker, P. Amer, Multiple tcp connections improve http throughput myth or fact, in: IEEE IPCCC, 2009.
[40] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, D. Wether-all, Reducing network energy consumption via rate-adaptation and sleeping, in: Proc. Of NSDI, April 2008, 2008.
[41] U. of Minnesota, Minnesota internet traffic studies (mints), 2012,
[42] L. Popa, A. Ghodsi, I. Stoica, Http as the narrow waist of the future internet, in: Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, 2010, p. 6.

[43] F. Rawson, I. Austin, Mempower: a simple memory power analysis tool set, IBM Austin Res. Lab. (2004).
[44] P. Richter, N. Chatzis, G. Smaragdakis, A. Feldmann, W. Willinger, Distilling the Internet's Application Mix from Packet-sampled Traffic, in: J. Mirkovic, Y. Liu (Eds.), Passive and Active Measurement, Lecture Notes in Computer Science, 8995, 2015, pp. 179–192.
[45] S. Rivoire, P. Ranganathan, C. Kozyrakis, A comparison of high-level full-system power models., In Proc. of HotPower'08 (2008).
[46] C. Systems, Visual networking index: Forecast and methodology, 2016 – 2021, 2017,
[47] T.N. Tran, K. Drab, M. Daszykowski, Revised {DBSCAN} algorithm to cluster data with dense adjacent clusters, Chemomet. Intell. Lab. Syst. 120 (0) (2013) 92–96.
[48] S.V. Vrbsky, M. Galloway, R. Carr, R. Nori, D. Grubic, Decreasing power consumption with energy efficient data aware strategies, FGCS 29 (5) (2013) 1152–1163.
[49] E. Yildirim, E. Arslan, J. Kim, T. Kosar, Application-level optimization of big data transfers through pipelining, parallelism and concurrency, IEEE Trans. Cloud Comput. 4 (1) (2016) 63–75.
[50] E. Yildirim, T. Kosar, End-to-end data-flow parallelism for throughput optimization in high-speed networks, J. Grid Comp. (2012) 1–24.