To send or not to send: Reducing the cost of data transmission

Leana Golubchik*, Samir Khuller†, Koyel Mukherjee†, Yuan Yao*
*University of Southern California, Los Angeles, CA 90089
Email: {leana, yuanyao}@usc.edu
†University of Maryland, College Park, MD 20742
Email: {samir, koyelm}@cs.umd.edu

***Abstract*—Frequently, ISPs charge for Internet use not based on peak bandwidth usage, but according to a percentile (often the 95th percentile) cost model. In other words, the time slots with the top 5 percent (in the case of 95th percentile) of data transmission volume do not affect the cost of transmission. Instead, we are charged based on the volume of traffic sent in the 95th percentile slot. In such an environment, by allowing a short delay in transmission of some data, we may be able to reduce our cost considerably.**

We provide an optimal solution to the offline version of this problem (in which the job arrivals are known), for any delay $D > 0$. The algorithm works for any choice of percentile. We also show that there is no efficient deterministic online algorithm for this problem. However, for a slightly different problem, where the maximum amount of data transmitted is used for cost accounting, we provide an online algorithm with a competitive ratio of $\frac{2D+1}{D+1}$. Furthermore, we prove that no online algorithm can achieve a competitive ratio better than $\frac{2D+1}{D+F(D)}$ where $F(D) = \sum_{i=1}^{D+1} \frac{i}{D+i}$ for any $D > 0$ in an adversarial setting.

We also provide a heuristic that can be used in an online setting where the network traffic has a strong correlation over consecutive accounting cycles, based on the solution to the offline percentile problem. Experimental results are used to illustrate the performance of the algorithms proposed in this work.

I. Introduction

A huge amount of data is transferred over the Internet every day. For example, Google, Amazon and other operators of large data centers that host cloud computing applications need to provide services and synchronize processed data across multiple locations. Multimedia content providers such as Akamai need to replicate video clips over servers located in different regions for optimized content delivery. Huge data chunks are transmitted over high speed links, and cost the senders a significant amount of money on a daily basis.

Cost accounting for data transfer is performed differently from other utilities such as energy, where companies are billed by the total volume of utility consumed. For network bandwidth, the cost in large scale computing systems, the 95th percentile rule is widely used [10]. The rule can be described as follows: the average utility usage is sampled every w time units. Thus, in an accounting cycle of length L, a total of $\frac{L}{w}$ samples are taken. (A typical number for w is 5 minutes and L is 24 hours.) Then the 95th percentile of these samples is taken as the actual utility usage in that accounting cycle.

The performance requirements of such data transfers typically include transmission delay, as the data chunks have to be delivered within some time period. Often, such delay requirements are not stringent. For example, video replications over servers at different locations do not have to be carried out exactly at 9:00PM daily. All that is needed is that they are completed, for instance, before midnight. This provides opportunities for data senders to optimize data transfers by attempting to reduce the 95th percentile usage. We can view data chunks as jobs, and links (or senders) as servers. Then, the problem can be viewed as a scheduling problem aimed as reducing the 95th percentile usage. In fact, given some slack in job scheduling, we can consider any percentile as defining our metric.

Consider the following simple example: Suppose there is a server that serves jobs of the same size. Jobs arrive at the server in each time slot. The server has a capacity of serving up to 5 jobs in one slot. Over an accounting period of 5 time slots, the number of jobs arriving at the server is $[3, 3, 3, 3, 0]$. Our goal is to minimize the 80th percentile (second largest in this case) of the number of jobs served in one time slot. If all jobs are served within the time slot in which they arrive, with no delay, then the service sequence is also $[3, 3, 3, 3, 0]$. Note that according to the 80th percentile rule, the billed output bandwidth usage is also 3. However by delaying each job by at most one time slot, the server can instead choose the service sequence $[2, 4, 2, 2, 2]$, thus reducing the billed bandwidth usage to 2, since the slot with the maximum load is not counted. Note that the trade-off here is that four jobs are delayed. One is delayed from slot 1 to slot 2, one from slot 3 to slot 4, and two from slot 4 to slot 5.

The above example looks straightforward. But the general problem of percentile minimization can be difficult, due to the following: 1) jobs arrive with delay requirements and 2) the job arrival patterns might not be available ahead of time. This paper considers this problem, as well as a closely related problem from an algorithmic prospective. We focus on reducing the 95th percentile because it is the value based on which the bandwidth usage is currently billed; however, our approach can be applied to other percentile values[1]. We provide job scheduling algorithms and analysis of the problem of minimizing the 95th percentile of job service, subject to

[1]For the problem of minimizing the 100th percentile (maximum value), the complexity of our approach is lower than the complexity for other percentile values.

delay constraints. We focus on uniform delay requirements, where all jobs arrive with the same delay requirement. Our study covers both the offline problem, where all the job arrivals are known in advance, and the online problem, where no knowledge of job arrivals is available a priori.

While scheduling problems have been investigated for decades [12], the metrics considered in prior work have more to do with minimizing the maximum load, or scheduling a given fraction of jobs, while minimizing maximum load [6]. Our model represents a new class of problems, and is based on the charging mechanism in use.

The bounded delay buffer management problem, first introduced by Kesselman et al. [9] is a somewhat similar problem that has been widely studied in literature. This is an online problem, where jobs have weights and deadlines, and the goal is to maximize the weight of jobs served within their deadline, where the processor can only process one job at a time. Though the problem seems related to the problems we consider in this work, it is not obvious that their techniques can be directly applied to our problem. In Section X we provide some more references to subsequent works in this area.

The contributions of this work are summarized as follows:

- We analyze the offline problem of minimizing the 95th percentile of job service where the jobs can be delayed by at most one time slot, i.e., $D = 1$, and provide OPT, a dynamic programming algorithm that obtains an optimal service schedule in $O(N^2 \log C)$ time, where N is the number of time slots and C is the link capacity (see Section III). The algorithm can be modified to consider any percentile.
- We extend our approach to general uniform delay requirements. (see Section IV).
- We show that the online problem is hard and no algorithm achieves constant competitive ratio (see Section V).
- We study a closely related problem where the maximum, instead of the 95th percentile is minimized. We provide an efficient online algorithm that achieves close to optimal performance for this problem (see Section VI). We validate this by showing a lower bound that any deterministic algorithm can achieve for a general uniform delay.
- We provide a simple heuristic aimed at reducing the 95th percentile of job service based on OPT. The heuristic can be used in an online manner (see Section VII).
- Extending the original problem to the case where jobs have variable sizes makes it NP-hard (See Section VIII).
- We evaluate the performance of all algorithms using synthetic workload datasets and show that our heuristic achieves up to 13% reductions of the 95th percentile of the number of jobs served (see Section IX).
- We provide pointers to future work and conclude in Section XI.

II. Problem Definition

Consider a system with a single server, where time is slotted. Let N be the total number of time slots in an accounting cycle. Let $A(i)$ denote the number of jobs arriving in time slot i. Each job is of unit work. Jobs arrive at the beginning of each time slot and some number of them, $S(i)$, are served by the end of that time slot. Jobs that are served in the time slot in which they arrived incur no delay. Unserved jobs are carried over for service in future time slots; each time a job is carried over, it incurs an additional delay of one time slot.

The goal is to choose $S(i), i = 1, 2, 3, ..., N$, such that the 95th percentile of $S(i)$ is minimized. To make this problem realistic (and potentially interesting) the following constraints are introduced:

- The server has a capacity of C. Thus, $S(i) \leq C, \forall i$. To simplify the problem, we assume that $A(i) \leq C, \forall i$.
- Each job can be delayed by at most D time slots ($0 < D \gg N$).

This problem has an offline and an online version, depending on whether job arrivals, $A(i)$, are known in advance or not.

III. Solution to Offline Problem for $D = 1$

In this section we give an algorithm to solve the proposed offline problem, where $A(i), i = 1, 2, ..., N$ is known a priori. To simplify the problem, we first assume delay $D = 1$. We will later extend our solution to accommodate general D values. We call our approach OPT.

A solution to this problem is useful in scenarios like bulk data transfer [10], where workload is pre-determined but can be scheduled with some flexibility. According to the 95th percentile accounting rule, the number of jobs served in the top 5% of the total time slots are not included in accounting. These time slots can be viewed as "free" because the jobs served in these time slots do not change the 95th percentile result. Let $T = 5\% \times N$ be the number of "free" time slots (we assume T is an integer). Assume that in "free" time slots, up to C jobs can be served, whereas in the other $N - T$ time slots, at most $H (H \leq C)$ jobs can be served. In this way, H serves as the upper bound of the 95th percentile of the number of jobs served in one time slot.

The goal is to find the smallest H, so that there is a schedule to serve jobs, such that all but T slots serve at most H jobs. We make a guess for H, and then verify if our guess leads to a feasible solution. We can do a doubling search, combined with a binary search for H.

We first discuss the solution for unit delay ($D = 1$). At step i, the number of arriving jobs is $A(i)$. Each job should be served in the same slot, or in the next slot. The load on any single slot cannot exceed C (the maximum capacity).

$$OPT(1, 0) = \max\{A(1) - H, 0\}$$

$$OPT(1, 1) = 0$$

$$OPT(i, t) = \min \begin{cases} \max\{OPT(i-1, t-1) + A(i) - C, 0\} \\ \max\{OPT(i-1, t) + A(i) - H, 0\} \text{ if } OPT(i-1, t) \leq H \end{cases}$$

Since this is a standard dynamic program, we do not further discuss how to implement it. The key point is that if a

feasible schedule exists with the chosen value of H, then $OPT(N,T) = 0$. If $OPT(N,T) > 0$ then this implies that there is no schedule with the guessed value of H.

We assume that each instance has the property that $A(i) \leq C$. From this it is easy to see (by induction) that $OPT(i,t) \leq C$. In other words, since we never receive more than C jobs, we never pass on more than C unfinished jobs.

Theorem 1. *$OPT(i,t)$ minimizes the number of unserved jobs in the first i slots, assuming that at most t slots can have load $> H$, and all the other slots are* required *to have load at most H.*

Proof: The proof is by induction on (i,t). In other words, we assume that we have proven the claim for $(i-1,j)$ for all $j \leq i-1$ and wish to prove it now for (i,t) for all $t \leq i$. In fact if $t \geq i$, $OPT(i,t) = 0$ since all the slots are "free". If we assume that a solution exists for the given value H then clearly by time N, all the jobs are handled and thus $OPT(N,T) = 0$.

For the general case note that in an optimal solution the load in slot i is either $> H$ or $\leq H$. In the former case, we have a capacity of C in time slot i. If an optimal solution exists that passes on p jobs, by induction our solution for $(i-1,t-1)$ passes on at most p jobs. As a result, if the load at time i is at most C then with fewer jobs it is at most C. Since at most C jobs are passed on from slot $i-1$, we can handle them all in time slot i. In the latter case, we use an optimal solution for the sub-problem $(i-1,t)$, but it is important to note that if more than H jobs are passed on, then we cannot feasibly handle more than H delayed jobs in time slot i since there is an upper bound on the delay of $D = 1$. ■

IV. Extending OPT to General Uniform Delay

We assume that all arriving jobs can be delayed by at most $D \geq 1$ time slots. The goal is to minimize the 95th percentile of the number of jobs served. Our algorithm in Section III solves the $D = 1$ case. We now extend this algorithm to the general D case. When $D > 1$, the jobs left unfinished by time slot i consist of jobs that have been delayed for $1, 2, ..., D$ time slots. Intuitively, the service priority for these jobs should be different. In general, let $\boldsymbol{U(i)} = (U_1(i), ..., U_D(i))$ be the vector of unfinished jobs at *the end* of time slot i. That is, an element $U_j(i)$ represents the number of jobs that have been delayed for j time slots, including time slot i. For instance, $U_1(i)$ is the number of jobs that arrived in time slot i but did not get served in that time slot. Similarly, $U_D(i)$ represents the number of jobs that have already been delayed for a maximum allowed D time slots; thus, they all have to be served in time slot $i+1$. Let us define an augmented vector $\boldsymbol{U^A(i)} = (A(i), \boldsymbol{U(i-1)})$, where $U_0^A(i) = A(i)$ and $U_k^A(i) = U_k(i-1)\ \forall\ k \geq 1$. Let $\boldsymbol{S(i)} = (S_0(i), ..., S_D(i))$ be the service vector.

The service vector $\boldsymbol{S(i)}$ represents the amount of work done in time slot i. Specifically, $S_j(i)$ represents the quantity of jobs $U_j^A(i)$ served in time slot i.

In this scenario, H being feasible means that $\sum_{j=0}^{D} S_j(i) \leq H$ for an "accounted" slot and $\sum_{j=0}^{D} S_j(i) \leq C$ for a "free" slot. Moreover, $0 \leq S_j(i) \leq U_j^A(i)$.

We show that an algorithm similar to the one given in Section III can solve the general problem. We define a dynamic program to calculate $OPT(i,t)$. Since jobs can be delayed for more than one time slot now, our algorithm now needs to (1) provide a service schedule vector, (2) calculate the vector of unfinished jobs, and (3) decide whether a certain H value is feasible.

For a particular $\boldsymbol{U^A(i)}$, there are multiple feasible $\boldsymbol{S(i)}$ vectors that satisfy the above constraints. For example, when $D = 2$, suppose $\boldsymbol{U(i-1)} = (2,1)$, $A(i) = 3$ and $H = 3$, $\boldsymbol{S(i)}$ can be either $(1,1,1)$, $(0,2,1)$, or $(2,0,1)$.

We choose $\boldsymbol{S(i)}$ in a greedy manner. Specifically, jobs that arrive earlier will be served before those arriving later. This is intuitive because jobs that have been delayed longer are more *urgent* and should be prioritized over those that just arrived. Given this, the service schedule vector we choose for the example above is $(0,2,1)$.

To check feasibility of assigning an "accounted" time slot or "free" slot, we need to compare H with $U_D(i)$ to check if the corresponding schedule is feasible.

We define $OPT(i,t), \{i = 1,2,3,...,n; t = 0,1,2,...,T\}$ as the minimum number of total jobs left unfinished by the end of time slot i, using t "free" time slots. We have $OPT(i,t) = 0$ if $i \leq t$, as we assume that $A(i) \leq C, \forall i$.

For every (i,t), we have associated vectors $\boldsymbol{U(i,t)}$ and $\boldsymbol{S(i,t)}$, where the former is the vector of unserved jobs after time slot i when t free slots have been used, and the latter is the vector of served jobs in time slot i when t free slots have been used. These vectors are stored along with the $OPT(i,t)$ entry.

$OPT(i,t)$ is chosen to be one of the following values, depending on whether i is a free slot or an accounted slot.

If i is a free slot, then we can transmit up to C jobs. This is distributed among the unserved jobs as follows. First, we define the augmented vector $\boldsymbol{U^A(i,t)} = (A(i), \boldsymbol{U(i-1,t-1)})$. Let k be the smallest index such that $\sum_{j=k}^{D} U_j^A(i,t) \leq C$. Since $A(i) \leq C\ \forall i$, k is well-defined.

Let $\hat{U}_{k-1}^A(i,t) = C - \sum_{j=k}^{D} U_j^A(i,t)$. Then $\boldsymbol{S(i,t)} = (0,\ldots,0,\hat{U}_{k-1}^A(i,t), U_k^A(i,t), \ldots, U_D^A(i,t))$. We define $\boldsymbol{U^C(i,t)}$ to be the first D elements of $\boldsymbol{U^A(i,t)} - \boldsymbol{S(i,t)}$. This is because the last element of the difference of the vectors is 0 since k is well-defined.

In this case, define $OPT_1(i,t) = \sum_{j=1}^{D} U_j^C(i,t)$.

If i is an accounted slot, then we can transmit up to H jobs. This is again distributed among the unserved jobs as follows. First, we again define the augmented vector for this case as $\boldsymbol{U^A(i,t)} = (A(i), \boldsymbol{U(i-1,t)})$. Let k be the smallest index such that $\sum_{j=k}^{D} U_j^A(i,t) \leq H$.

If there exists no such k, that means $U_D^A(i,t) = U_D(i-1,t) > H$, hence this value of H is infeasible. In this case, define $OPT_2(i,t) = \infty$.

Otherwise, let $\hat{U}_{k-1}^A(i,t) = H - \sum_{j=k}^{D} U_j^A(i,t)$. Then $\boldsymbol{S(i,t)} = (0,\ldots,0,\hat{U}_{k-1}^A(i,t), U_k^A(i,t), \ldots, U_D^A(i,t))$. Again, $\boldsymbol{U^H(i,t)}$ is defined to be the first D elements of

$\boldsymbol{U^A(i,t) - S(i,t)}$. This is because the last element of the difference of the vectors is 0 since k is now well-defined.

In this case, define $OPT_2(i,t) = \sum_{j=1}^{D} U_j^H(i,t)$.

$OPT(i,t)$ is chosen to be the minimum of $OPT_1(i,t)$ and $OPT_2(i,t)$. The corresponding vector $\boldsymbol{U^C(i,t)}$ or $\boldsymbol{U^H(i,t)}$ is retained as vector $\boldsymbol{U(i,t)}$.

Therefore, for every (i,t), $OPT(i,t) = \sum_{j=1}^{D} U_j(i,t)$ by definition. Hence, we can now define $OPT(i,t)$ more compactly as follows.

$$OPT(1,0) = \max(A(1) - H, 0)$$

$$OPT(1,1) = 0$$

$$OPT(i,t) = 0 \ \forall \ t \geq i$$

$$OPT(i,t) = \min \begin{cases} \max\{OPT(i-1,t-1) + A(i) - C, 0\} \\ \max\{OPT(i-1,t) + A(i) - H, 0\} \text{ if } U_D(i-1,t) \leq H \end{cases}$$

Theorem 2. *$OPT(i,t)$ minimizes the number of unserved jobs in the first i slots, assuming that at most t slots can have load $> H$, and all the other slots are* required *to have load at most H.*

Proof: The proof is by induction on (i,t). Given the base cases defined in our dynamic program, we assume that we have proven the claim for $(i-1,j)$ for all $j \leq i-1$ and wish to prove it now for (i,t) for all $t \leq i$. For $t = i$, $OPT(i,t) = 0$ since all the slots are "free". If a solution exists for a given H, then obviously $OPT(N,T) = 0$.

In the general case, in an optimal solution the load in slot i is either $> H$ or $\leq H$. The former case is similar to $D = 1$. In the latter case, we use an optimal solution for the sub-problem $(i-1,t)$, but if $U_D(i-1,t) > H$, then we cannot feasibly handle more than H delayed jobs in time slot i since these jobs have reached their maximum delay. Hence we need to prove that $U_D(i-1,t) \leq U_D^O(i-1,t)$ for all $t \leq i-1$ for any algorithm O.

We first prove the following claim:

***Claim* 1.** *For any (i',t'), if $U_j(i',t') > 0$ for any $1 < j \leq D$, then $U_k(i',t') = A(i'-k+1)$ for all $k < j$.*

Proof: This claim follows from the greedy manner in which we choose the service schedule. Specifically, at any (i',t'), if $U_j(i',t') > 0$ for some $1 < j \leq D$, then the jobs with lesser delay will not be processed at all. The jobs with delay less than j would be the jobs arriving at time slots $(i'-k+1)$ where $k < j$. ■

Proof of Theorem 2 contd. If $U_D(i-1,t) = 0$ then $U_D(i-1,t) \leq U_D^O(i-1,t)$. Otherwise, if $U_D(i-1,t) > 0$, then from the above claim $U_k(i-1,t) = A(i-k)$ for all $k < D$. However, $OPT(i-1,t) = \sum_{j=1}^{D} U_j(i-1,t)$ by definition. By the induction hypothesis, $OPT(i-1,t) \leq O(i-1,t)$ for any algorithm O ($O(i-1,t)$ is defined similarly to $OPT(i-1,t)$ for an algorithm O). Hence, $\sum_{j=1}^{D} U_j(i-1,t) \leq \sum_{j=1}^{D} U_j^O(i-1,t)$. Since $U_k(i-1,t) = A(i-k) \geq U_k^O(i-1,t)$ for all $k < D$, therefore, $U_D(i-1,t) \leq U_D^O(i-1,t)$ for any O. This completes the proof of Theorem 2. ■

V. Online Problem: Hardness

So far we have studied an *offline* version of the 95th percentile problem, where it is assumed that the sequence $A(i), i = 1,2,...,N$ is known before OPT is carried out.

In this section, we relax this assumption and assume that $A(i)$ cannot be observed until time slot i. This relaxation makes the problem more realistic because in many scenarios, it is difficult to predict the volume of future job arrivals. However, it also makes the problem much more difficult. In fact, for the 95th percentile problem, performance of any online algorithm can be quite bad, as stated in Theorem 3.

For simplicity, we use $\mathcal{N}_A^X$ to denote the 95th percentile of the service schedule generated by algorithm X for arrival sequence $A(1)$ through $A(N)$.

Theorem 3. *For any online algorithm there exists an arrival series $A(i), i = 1,2,...,N$ for unit delay, such that $\frac{\mathcal{N}_A^{OL}}{\mathcal{N}_A^{OPT}}$ cannot be bounded by any constant.*

To prove the theorem, we show that an adversary is able to construct an arrival sequence, such that no matter how the jobs are being served, the ratio of the optimal solution and the one achieved by the online algorithm goes to infinity.

Proof: The adversary constructs the arrival sequence as follows: Let $A(1) = x$, and then $A(i) = \alpha x$, where $\alpha > 1$ for $i = 2,3,....$ Generate such arrivals until the number of time slots with non-zero job service ($S(\cdot)$) is exactly T for the online algorithm. Suppose this happens at time slot T^*. Clearly, $T^* \leq 2T$.

If $T^* < 2T$, then let $A(T^*+1) = x$, $A(T^*+2), ..., A(n) = 0$. In this case, we can construct $S^{OPT}(2k-1) = 0$ and $S^{OPT}(2k) = A(2k-1) + A(2k)$ for $k = 1,2,...,T^*/2$. Since $T^* + 1 \leq 2T$, all these non-zero services are not accounted. Thus $\mathcal{N}_A^{OPT} = 0$. On the other hand, $\mathcal{N}_A^{OL} > 0$. The reason is that in time slot 1 through $T^* + 2$ there are at least $T + 1$ time slots with non-zero job service for the online algorithm. Therefore, $\frac{\mathcal{N}_A^{OL}}{\mathcal{N}_A^{OPT}}$ can be arbitrarily large.

On the other hand, if $T^* = 2T$, then it must be the case that, $S^{OL}(1) = 0$, $S^{OL}(2) = (1+\alpha)x$, $S^{OL}(2k) = 2\alpha x$ and $S^{OL}(2k-1) = 0$ for $k = 2,3,...,T$. Now let $A(T^*+1) = 4\alpha x$. $A(T^*+2), ..., A(n) = 0$, and we have $\mathcal{N}_A^{OL} \geq \min\{(1+\alpha)x, 2\alpha x\} \geq (1+\alpha)x$. On the other hand, $\mathcal{N}_A^{OPT} = x/2$. The corresponding optimal job service sequence is $S^{OPT}(1) = S^{OPT}(2) = x/2$, $S^{OPT}(2k+1) = A(2k) + A(2k+1)$ and $S^{OPT}(2k+2) = 0$ for $k = 1,2,...,T$. Then the competitive ratio is at least $\frac{(1+\alpha)x}{x/2} = 2(1+\alpha)$ and cannot be bounded. ■

VI. The Min-Max Online problem

In Section V we showed that the original online problem has no constant competitive algorithm. From the proof we can see that the difficulty lies in the correct identification and usage

of "free" time slots. This motivates our next step on a similar problem where *no* "free" time slots are allowed. Specifically, we consider the problem of minimizing the *maximum* number of jobs served in a slot, instead of the 95th percentile. Formally, let $A(i)$ and $\boldsymbol{S(i)}$ be the number of job arrivals (with unit work) and the vector of jobs served in time slot i, respectively.

Again, jobs can be delayed for at most D time slots, and our goal is to minimize the maximum of the sum of $\boldsymbol{S(i)}$, i.e., total number of jobs served in a time slot, over a potentially infinite time horizon ($N \to \infty$).

We refer to this problem as the min-max problem. We note that the offline version of this problem is a special case of the problem solved by Yao et al. [13]. Here we consider the online version of the problem.

A. An efficient online algorithm for the min-max problem

We term our approach ES as short for "Equal Split".

Without loss of generality, we define $A(i) = 0$ when $i \leq 0$. With this, the algorithm is quite simple.

Algorithm 1 ES: Calculate $\boldsymbol{S^{ES}(i)}, i = 1, 2, ...$

1: **for** time slot $i = 1, 2, ...$ **do**
2: Obtain $A(i)$
3: $\boldsymbol{S(i)} \leftarrow \frac{1}{D+1}(A(i-D), A(i-D+1), ..., A(i))$
4: **end for**

Consider Algorithm 1, in each time slot, the arriving jobs are split into $D+1$ sets of equal size. In this time slot, as well as each of the following D time slots, one set will be served.

The competitive ratio of this simple algorithm is given in Theorem 4.

Theorem 4. *The competitive ratio of* ES *is* $\frac{2D+1}{D+1}$.

Proof: Let $F_A(i) \triangleq \sum_{j=i-D}^{i} A(j)$ be the number of job arrivals during consecutive $D+1$ time slots ending with time slot i. Additionally, let $\mathcal{F}_A \triangleq \max_i F_A(i)$. Let M_A^{AL} refer to the maximum number of jobs served in one time slot for an arrival sequence A, by an algorithm AL. Then we have:

$$\mathcal{M}_A^{\text{ES}} = \frac{\mathcal{F}_A}{D+1}. \tag{1}$$

This is because in scheme ES, the number of jobs served in time slot i is exactly $\frac{\sum_{j=i-D}^{i} A(j)}{D+1}$.

However, in an optimal solution we have:

$$\mathcal{M}_A^{OPT} \geq \frac{\mathcal{F}_A}{2D+1}. \tag{2}$$

This is the case since all jobs arriving between time slots $i-D$ and i have to be served between time slot $i-D$ and time slot $i+D$, a consecutive of $2D+1$ time slots.

Comparing (1) and (2) we can see that the competitive ratio of ES is at most $\frac{2D+1}{D+1}$. On the other hand, consider the following arrival series: $A(1) = (D+1)^2$, $A(2) = A(3) = ... = A(D+1) = D+1$ and $A(i) = 0\ \forall i > D+1$. It can be verified that $\mathcal{M}_A^{OPT} = D+1$ and $\mathcal{M}_A^{ES} = 2D+1$. Therefore, the competitive ratio is tight.

■

B. Lower bound for online algorithms

Above we showed that ES is $\frac{2D+1}{D+1}$ competitive. In fact, this ratio is not far from what any online algorithm can possibly achieve. We now give a lower bound on the performance of any online algorithm.

Theorem 5. *For the min-max problem with maximum allowed delay D, the competitive ratio of any deterministic online algorithm denoted by OL is at least $\frac{2D+1}{D+F(D)}$, where $F(D) = \sum_{i=1}^{D+1} \frac{i}{D+i}$. In other words,*

$$\frac{\mathcal{M}_A^{OL}}{\mathcal{M}_A^{OPT}} \geq \frac{2D+1}{D+F(D)}. \tag{3}$$

The proof approach is to show that an adversary can force any online algorithm to achieve a competitive ratio of at least $\frac{2D+1}{D+F(D)}$.

Proof: Let $\alpha = \frac{2D+1}{(D+F(D))(D+1)}$. We need to show that the competitive ratio of any online algorithm is at least $(D+1)\alpha$. We use $\mathcal{M}_A^{OPT}(i)$ to denote the optimal solution to the min-max problem when $A(i+1)$ and all follow up arrivals are set to 0.

Let the adversary first set $A(1) = 1$. It is obvious that $\mathcal{M}_A^{OPT}(1) = \frac{1}{D+1}$. If the online algorithm serves $S^{OL}(1) \geq \alpha$, then the adversary stops, and the competitive ratio is at least $(D+1)\alpha$. Otherwise, the carry-over $U_A^{OL}(1) \geq 1-\alpha$. Now the adversary sets $A(2)$ to be 1 also. $\mathcal{M}_A^{OPT}(2) = \frac{2}{D+2}$. If the online algorithm results in $S^{OL}(2) \geq \frac{2(D+1)}{D+2}\alpha$, then the adversary stops, and the competitive ratio is at least $(D+1)\alpha$. Otherwise, the adversary sets $A(3) = 1$. Then $\mathcal{M}_A^{OPT}(3) = \frac{3}{D+3}$. If the online algorithm results in $S^{OL}(3) \geq \frac{3(D+1)}{D+3}\alpha$, then the adversary stops, and the competitive ratio is at least $(D+1)\alpha$. Continuing in this manner, the adversary sets $A(1)$ through $A(D+1)$ to be 1, then $A(D+2)$ through $A(2D+2)$ to be $D+1$, and so on. At any time step i, if the online algorithm does no less than $(D+1)\alpha$ of the optimum offline algorithm in the time slot i, or $(D+1)\alpha\mathcal{M}_A^{OPT}(i)$, the sequence stops, and the adversary has forced a competitive ratio of $(D+1)\alpha$. Otherwise, the sequence of jobs faced by the online algorithm from time slot 1 to $k(D+1)$ is $1, D+1, (D+1)^2, ..., (D+1)^{k-1}$, each repeated $D+1$ times. From the work of Yao et al. [13], we know that

$$\mathcal{M}_A^{OPT}(p(D+1)+r) = \frac{r(D+1)^p}{D+r},$$
$$p = 0, 1, ..., k-2, r = 1, 2, ..., D+1$$

Thus, the carry-over to time slot $k(D+1)+1$ satisfies

$$U_A^{OL}(k(D+1)) \geq \sum_{i=1}^{k(D+1)} [A(i) - S_A^{OL}(i)]$$
$$\geq (D+1)\sum_{i=0}^{k-1}(D+1)^i$$

$$- (D+1)\alpha(\sum_{i=1}^{D+1}\frac{i}{D+i})(\sum_{j=0}^{k-1}(D+1)^j)$$
$$= (D+1)\frac{(D+1)^k-1}{D}(1-F(D)\alpha)$$

Now let $A(k(D+1)+1) = (D+1)^k$. In this case we have $\mathcal{M}_A^{OPT}(k(D+1)+1) = (D+1)^{k-1}$. If the online algorithm results in $S^{OL}(k(D+1)+1) \geq \alpha(D+1)^k$, then the adversary stops, and the competitive ratio is at least $(D+1)\alpha$. Otherwise,

$$\begin{aligned} &U_A^{OL}(k(D+1)+1) \geq U_A^{OL}(k(D+1)+1) \\ &+ (1-\alpha)(D+1)^k \\ &= (D+1)\frac{(D+1)^k-1}{D}(1-F(D)\alpha) + (1-\alpha)(D+1)^k \\ &= [\frac{2D+1}{D} - (\frac{D+1}{D}F(D)+1)\alpha](D+1)^k \\ &- o((D+1)^{k-1}) \end{aligned} \tag{4}$$

Note that the last term on the right side of Equation (4) is $\frac{(D+1)(1-F(D)\alpha)}{D}$. It is much less than $(D+1)^{k-1}$ when k is large, so we use $o((D+1)^{k-1})$ to represent it.

Next the adversary lets all subsequent number of arrivals be 0. We know that $\mathcal{M}_A^{OPT} = (D+1)^{k-1}$. Moreover, $U_A^{OL}(k(D+1)+1)$ has to be served in the following D time slots. Thus

$$\mathcal{M}_A^{OL} \geq \frac{U_A^{OL}(k(D+1)+1)}{D}$$

Thus the competitive ratio

$$\begin{aligned} &\frac{\mathcal{M}_A^{OL}}{\mathcal{M}_A^{OPT}} \geq \frac{U_A^{OL}(k(D+1)+1)}{D(D+1)^{k-1}} \\ &= \frac{(D+1)^k[\frac{2D+1}{D} - (\frac{D+1}{D}F(D)+1)\alpha] - o((D+1)^{k-1})}{D(D+1)^{k-1}} \end{aligned} \tag{5}$$

Let $k \to \infty$ in Equation (5) and plug in $\alpha = \frac{2D+1}{(D+F(D))(D+1)}$, we have

$$\begin{aligned} &\frac{\mathcal{M}_A^{OL}}{\mathcal{M}_A^{OPT}} \geq \frac{(D+1)(2D+1)}{D^2} - [\frac{(D+1)^2}{D^2}F(D) + \frac{D+1}{D}]\alpha \\ &= \frac{(D+1)(2D+1)}{D^2} - \frac{(2D+1)((D+1)F(D)+D)}{D^2(D+F(D))} \\ &= \frac{2D+1}{D+F(D)} = (D+1)\alpha \end{aligned}$$

This completes the proof. ■

We note that when $D = 1$, the lower bound is $\frac{2D+1}{D+F(D)} = \frac{18}{13}$. At the same time, ES achieves a ratio of $\frac{2D+1}{D+1} = \frac{3}{2}$, which is quite close to the lower bound. When $D \to \infty$, we have that $F(D) \to (1-\ln 2)D+1$. Thus the lower bound $\frac{2D+1}{D+F(D)} \to \frac{2}{2-\ln 2} \approx 1.53$. At the same time, ES's competitive ratio approaches 2.

VII. Applying OPT

In Section III, we have shown OPT gives an offline optimal solution to the 95th percentile problem. However, it relies on knowledge of future job arrivals. This limits the practicability of the algorithm. In this section, we provide a heuristic approach based on OPT that is more practical. We term this algorithm HEuristic for onLine PErcentile Reduction, HELPER in short.

A. The HELPER approach

In many cases, network traffic in real systems exhibits a pattern that repeats over the accounting cycle, which is typically 24 hours. For example, traffic volume at the same times on two consecutive weekdays days often shows strong correlations. Such correlation can be leveraged to obtain good performance in practice without having a priori knowledge of arrivals. Thus we can use traffic information obtained today as a prediction for tomorrow. Below we propose a simple heuristic approach based on this observation and OPT.

Let the job arrivals of day x be denoted by $A_x(1)$ through $A_x(N)$. We record job arrivals of day x. At the end of the day, we apply OPT and obtain $S_x(1), ..., S_x(N)$ and $U_x(1), ..., U_x(N)$. Now let $r_x(i) = \frac{S_x(i)-U_x(i-1)}{A_x(i)}, i = 1, 2, ..., N$, be the fraction of $A_x(i)$ that are not being delayed. With $r_x(i), i = 1, 2, ..., N$, the arrivals on day $x+1$ can be served according to the following algorithm.

Algorithm 2 HELPER: Calculate $S_{x+1}(i), i = 1, 2, ..., N$

Require: $r_x(i), i = 1, 2, ..., N, T = 0.05N$
1: $U_{x+1}(0) \leftarrow 0$
2: **for** time slot $i = 1, 2, ..., N$ **do**
3: Obtain $A_{x+1}(i)$
4: $s \leftarrow r_x(i)A_{x+1}(i) + U_{x+1}(i-1)$
5: $m \leftarrow (T+1)$th largest in $S_{x+1}(j), j = 1, 2, ..., i-1$
6: **if** $s > m$ **then**
7: $S_{x+1}(i) \leftarrow s$
8: **else**
9: $S_{x+1}(i) \leftarrow \min\{m, A_{x+1}(i) + U_{x+1}(i-1)\}$
10: **end if**
11: $U_{x+1}(i) \leftarrow A_{x+1}(i) + U_{x+1}(i-1) - S_{x+1}(i)$
12: Record $A_{x+1}(i)$
13: **end for**

Let us consider $A_{x+1}(i), i = 1, 2, ..., N$. Intuitively, when $A_{x+1}(i) = K \times A_x(i), i = 1, 2, ..., N$, where $K > 0$ is a constant, following the service schedule of day x results in optimal 95th percentile for day $x+1$. Such a schedule is characterized by the sequence $r_x(1)$ through $r_x(N)$. Thus, in day $x+1$, when $A_{x+1}(i)$ comes in, we should serve $s = r_x(i)A_{x+1}(i) + (1-r_x(i-1))A_{x+1}(i-1)$ jobs. This is stated in line 4 of Algorithm 2. On the other hand, the 95th percentile will not be increased unless more than m jobs are served, where m is the $(T+1)$st largest value from $S(1)$ through $S(i-1)$, which is already computed in previous time slots. (line 5). Thus we can serve up to the maximum of s and m, as expressed in lines 5 to 10. The idea here is that if $s < m$, we can still serve up to m jobs without increasing the 95th percentile. At the same time $A_{x+1}(i)$ is recorded and then used to learn $r_{x+1}(i)$. Then the sequence of $r_{x+1}(i), i = 1, 2, ..., N$ is used to schedule jobs arrive in day $x+2$, and so forth.

B. Discussions

In HELPER, we use $A_x(i)$ as a simple prediction of $A_{x+1}(i)$. We note that more sophisticated algorithms such as ARIMA[14] can be applied here to predict $A_{x+1}(i)$. These prediction algorithms might result in better performance of HELPER, but also bring in more computation overhead, while as shown in Section IX, our simple model already brings in significant results.

Also, it is easy to see that this heuristic approach does not come with any performance guarantee. It is not surprising that it can perform arbitrarily bad if $A_{x+1}(i)$ is not correlated with $A_x(i)$.

VIII. Extensions

One extension of the original problem is the case where the incoming jobs are not of the same size. Suppose now that each job consists of a number of tasks, and all tasks are of the same size. An additional constraint is that there are dependencies among these tasks, such that all the tasks of a job cannot be split and served in different time slots. For example, suppose a job that contains 3 tasks arrives in time slot 1. Due to dependency, it can be served as a whole in time slot 1 or 2. But it is not allowed to serve 2 tasks of this job in time slot 1, then 1 task in time slot 2. In this case, the min-max problem becomes NP-complete.

Theorem 6. *When jobs are not of unit size, the problem of minimizing the maximum number of job served is NP-complete*

Proof: The proof follows trivially by a reduction from PARTITION. ■

IX. Experiments

We now evaluate the performance of our approach through simulations. First, we describe our simulation setting.

A. Experimental settings

1) Workload data: We use hit records of Wikipedia as the basis, on top of which we generate our workload datasets. Wikipedia records its hourly hit rate of the previous month [1]. We use these records between August 11, 2010 and October 25th, 2010. Since the length of one time slot is usually 5 minutes, we interpolate this data so that the sample frequency is once every 5 minutes. Based on this record, we generate 6 time series with different characteristics. The statistics of these datasets are listed in Table I. Note that the "autocorr" column shows the auto-correlation with 24 hour lag. From the table we can see that the datasets show medium to high 24 hour lag auto-correlation and small (datasets 1 and 2) to high (datasets 5 and 6) coefficient of variation.

As an example, a portion of dataset 1 is depicted in Figure 1. It can be seen from this figure that these workload time series include diurnal patterns, as well as some "anomalies", such as change of traffic volume.

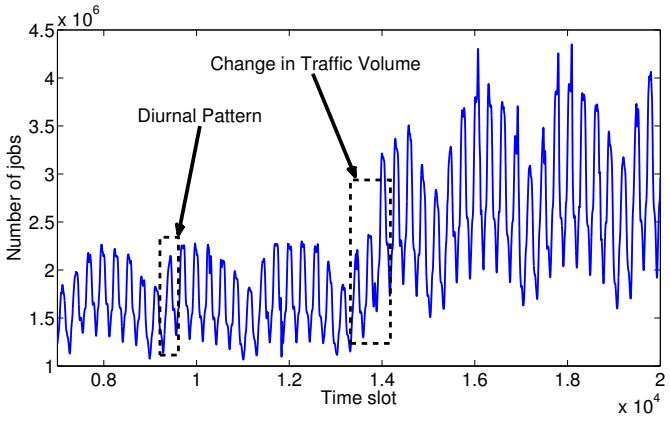

Fig. 1. An example of arrival dataset.

dataset	max$\times 10^6$	min$\times 10^6$	mean$\times 10^6$	autocorr	coef of var
1	4.35	1.04	2.01	0.93	0.34
2	2.32	1.00	1.58	0.82	0.17
3	3.30	0.23	1.66	0.74	0.54
4	3.34	0.25	1.71	0.78	0.54
5	2.35	0.00	0.72	0.71	0.89
6	2.34	0.04	0.86	0.72	0.70

TABLE I
Statistics of datasets.

2) Experimental methodology: We implement OPT, HELPER and ES. In the first set of simulations, we study the effect of different delays and percentile.

We run OPT with different D values against all datasets. We also modify OPT so that the minimization objective varies: 95th, 98th, and 99th percentile. (Recall that OPT can be adapted to find the optimal service schedule of any percentile; all that is needed is a change in the value of T.) We then drive all algorithms using all of the data sets and calculate the relative change in the 95th percentile and the maximum number of jobs served for each of the 74 days, as compared to the original scenario, where workload was served without any delay. Note that OPT and HELPER are originally designed to reduce the percentile value. We modify them so that OPT also computes the optimal solution for the min-max problem. Adopting this version of OPT, HELPER can also be used to reduce the maximum number of jobs served.

For each dataset, we obtain 3 sets of results. For OPT and ES the sets have 74 elements, whereas for HELPER there are 73 elements because no predictions are available for the first 24 hours.

B. Experimental results

In Figure 2 we plot the average performance of OPT in reducing the 95th, 98th and 99th percentile with different delay values, as compared with the original scheme. Each point in the figure is the average reduction in corresponding percentile over all 6 datasets. As D increases, OPT offer greater reduction. This is intuitive because increasing D allows greater flexibility in service. With this OPT is able to "accumulate"

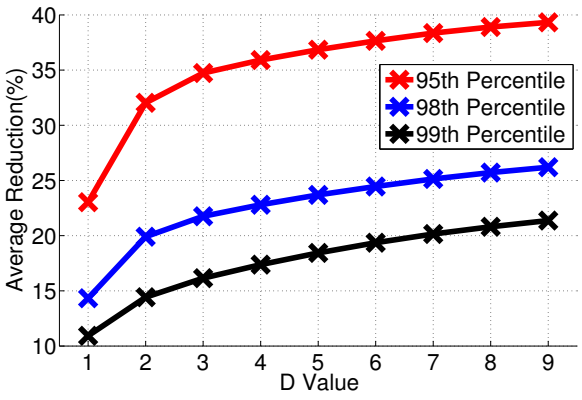

Fig. 2. Average reduction with different D values

more jobs in time slots that are not being accounted. Also, when the objective percentile decreases, OPT performs better since more "free" time slots are available.

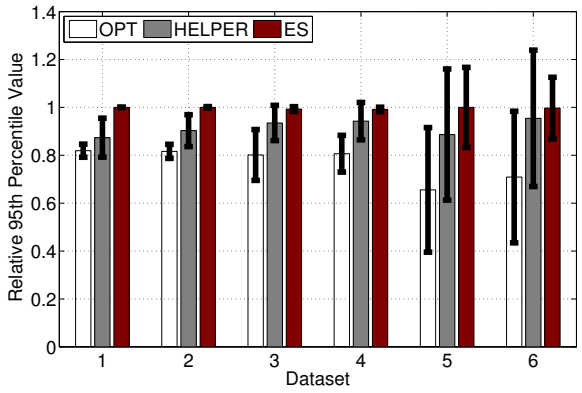

Fig. 3. 95th percentile results for OPT, HELPER and ES

In Figure 3 we show the performance of all algorithms in reducing the 95th percentile for all datasets. In this set of simulations D is set to be 1. As mentioned earlier, for each dataset, each algorithm generates more than 70 results. We use bars with different colors to show the mean value of these results generated by different algorithms. We also use the dark bars on top of each color bar to show the 95% (two sigma) confidence interval.

From Figure 3 we can see that OPT achieves significant reduction of the 95th percentile (18% to 35%) for all datasets. In the worst case, OPT provides more than 15% reduction as compared to the no delay scheme. HELPER reduces the 95th percentile by about 5% to 13% on average for different datasets. Moreover, the results of HELPER show a larger variation as compared to those of OPT. This is understandable because the predictions are not accurate, and this may cause HELPER to perform worse than the no delay scheme on some days, although on average HELPER results in better performance. Additionally, we can observe from Figure 3 that the variation in performance of HELPER increases significantly in datasets 5 and 6, where (1) the coefficients of variation are larger than those of datasets 1 through 4, and (2) the 24 hour lag autocorrelation is smaller than those of datasets 1 through 4. This makes sense because a high coefficient of variation and low autocorrelation indicate more "randomness" in the dataset. This makes accurate prediction of future traffic more difficult. Finally, ES does not show much improvement than the no delay scheme on average. It also shows increased variation as the coefficient of variation of the dataset increases. This is because ES is designed to reduce the maximum instead of the 95th percentile. It tends to "smooth" the number of jobs served instead of "accumulate" jobs arrived in several time slots into one "free" time slot, as OPT and HELPER do. Thus the "free" time slots may not be fully utilized in ES to serve more jobs.

To conclude, we see that for all datasets there is significant room for reduction, as demonstrated by results of OPT. Allowing larger deadline values brings in more potential reductions in percentiles. HELPER, while giving no performance guarantees, is effective in reducing the 95th percentile in all datasets. However, the performance of HELPER shows larger variance when the datasets include more "randomness". ES is not effective in reducing the 95th percentile.

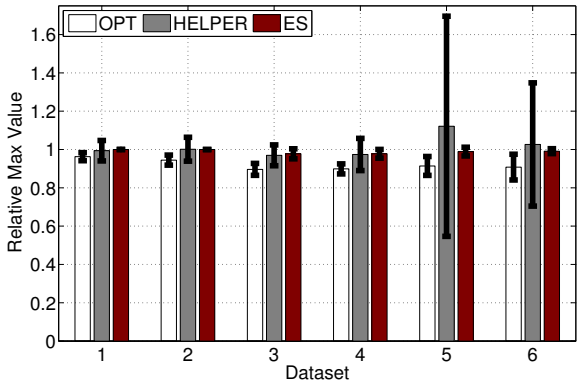

Fig. 4. Min-max results for OPT, HELPER and ES

Figure 4 depicts the reduction achieved (by the various algorithms) in the maximum number of jobs served. We observe that OPT reduces the maximum value by $\approx 4\% - 10\%$ on average. HELPER does not result in very good performance. For datasets 2, 5, and 6 it results in a larger maximum value than the no delay scheme. For other datasets, it gives at most a 3% reduction. Moreover, the variance in HELPER's performance is larger than that of both OPT and ES on all datasets. In summary, HELPER is not effective in reducing the maximum number of jobs served. Finally, ES shows a small reduction ($< 2\%$ on average, about 5% at most) in the maximum number of jobs served with very small variance in performance. The intuition here is that ES is designed for bounded *worst case* performance. There is no guarantee it would perform better than the no delay scheme in practice.

X. Related Work

In this paper, our main focus was on scheduling algorithms with uniform delay bound. As mentioned in Section I, the problem we consider is somewhat similar to bounded delay buffer management problem widely studied in literature. Although there are quite a few works in this area, none of them focus on the 95th percentile. For example, [7], [11], [8] studied online algorithms that try to maximize total *weight* of served jobs. They provide deterministic as well as randomized algorithms that achieve constant competitive ratio on throughput, which is the total weight of jobs served. The jobs that are not served before their deadlines are dropped. In our work, we study the problem of minimizing the 95th percentile or the maximum of the total number of jobs served in a time slot. No jobs are allowed to be dropped. Although these two problems appear to be related, it is not obvious that these techniques can be easily adapted to solve the problems we consider in this work.

In the CPU scheduling literature, specifically, in the domain of minimizing the energy and power consumed by a server through speed scaling, the works of Yao et al. [13] and Bansal et al. [2] are relevant. These works minimize the maximum number of jobs served with general deadline constraints.

Yao, Demers and Shenker [13] provide an optimal offline algorithm for minimizing the maximum energy consumed by a processor when the power consumed is a convex function of the speed. Their algorithm also minimizes the maximum speed of the processor at any point in time, subject to the constraint that all jobs are completed by their deadline.

Further, Bansal, Kimbrel and Pruhs [2] gave an optimal e-competitive algorithm for the online version of the energy minimization problem. This is also e-competitive for the online problem of minimizing the maximum speed at any instant such that all jobs are completed by their deadline. In general, we have a different goal of minimizing the 95th percentile. However in Section VI we study a similar problem to theirs and show that under the assumption of uniform delay our approach achieves a better competitive ratio.

A few papers do study the impact of the 95th percentile. For example, Dimitropoulos et al. [4] study the impact of time slot size through modeling and measurement study. Further, this accounting property is leveraged to reduce the cost of bulk data transfer by Laoutaris et al. [10]. Goldenberg et al. [5] design smart routing algorithms for data streams on multiple ISPs with the consideration of the 95th percentile model. In contrast, the main problems we consider are the following: minimize the 95th percentile through job level scheduling, and exploit the 95th percentile model.

In the broadcast scheduling literature, a related problem was studied by Charikar and Khuller [3]. They studied the problem of minimizing the response time for a fixed fraction of the total requests, ignoring the response time of a few requests, which is allowed to be large.

XI. Conclusions and Future Work

In this paper, we studied scheduling algorithms aimed at reducing the 95th percentile of jobs served, which is the basis of bandwidth cost accounting. We provided an offline algorithm that minimizes the 95th percentile for uniform delay constraints. We also showed that the online problem is hard. But for a slightly different problem, where the maximum is minimized, we provided a simple algorithm and prove that it achieves close to optimal performance.

As part of future work, we would like to close the performance gap between Equal Split ($\frac{2D+1}{D+1}$) and the lower bound ($\frac{2D+1}{D+F(D)}$). Here we presented a simple heuristic approach based on our offline algorithm. As part of future efforts, we are also interested in designing heuristics that are more effective in reducing the 95th percentile. Additionally, we would like to extend our work to the case of non-uniform delay, where different jobs can be delayed for different number of time slots.

XII. Acknowledgments

This work has been partially supported by NSF Grants CCF-1217890 and CCF-0937865.

References

[1] http://meta.wikimedia.org/wiki/Wikistats.

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proc. of the 45th IEEE Symposium on Foundations of Computer Science*, pages 520–529, 2004.

[3] M. Charikar and S. Khuller. A robust maximum completion time measure for scheduling. In *Proc. of the 17th ACM-SIAM symposium on Discrete algorithm*, pages 324–333, 2006.

[4] X. Dimitropoulos, P. Hurley, A. Kind, and M. Stoecklin. On the 95-percentile billing method. *Passive and Active Network Measurement*, pages 207–216, 2009.

[5] D. Goldenberg, L. Qiuy, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *Proc. of the ACM SIGCOMM*, pages 79–92, 2004.

[6] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Segev. Scheduling with outliers. *APPROX/RANDOM*, pages 149–162, 2009.

[7] L. Jez. A 4/3-competitive randomised algorithm for online packet scheduling with agreeable deadlines. In *27th International Symposium on Theoretical Aspects of Computer Science-STACS 2010*, pages 489–500, 2010.

[8] L. Jez. One to rule them all: A general randomized algorithm for buffer management with bounded delay. In *Proc. of the 19th European Symposium on Algorithms*, pages 239–250, 2011.

[9] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in qos switches. In *Proc. of the 33rd ACM symposium on Theory of computing*, pages 520–529, 2001.

[10] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram. Delay tolerant bulk data transfers on the internet. In *Proc. of the 11th international joint conference on Measurement and modeling of computer systems*, pages 229–238, 2009.

[11] F. Li, J. Sethuraman, and C. Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proc. of the 16th annual ACM-SIAM symposium on Discrete algorithms*, pages 801–802, 2005.

[12] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, second edition, 2006.

[13] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. of the 36th Symposium on Foundations of Computer Science*, pages 374–382, 1995.

[14] B. Zhou, D. He, and Z. Sun. Network Traffic Modeling and Prediction with ARIMA/GARCH. In *Proc. of HET-NETs Conference*, pages 1–10, 2005.