The Framework of User-Centric Optimization in Web-Based Applications

Ye Xia* Hoi-Sheung Wilson So Richard Hyong-Jun La Venkat Anantharam
Steven McCanne David Tse Jean Walrand Pravin Varaiya

Department of Electrical Engineering and Computer Science
University of California, Berkeley

Abstract

This paper takes a user-centric optimization (UCO) viewpoint of the web-browsing process, and asks how to transfer a web page in the most satisfying way to the user, given the limited bandwidth on the path from the web server to the user. We realize that the time to complete the download of a page is not an accurate measure of the perceived performance. Most web browsers today request images embedded in a web page in the order they appear in the HTML file. This approach ignores the fact that different images serve different purposes and may have different utilities to the user. Following the proposal of Gupta [1], we suggest arranging the transfer order of the images by taking into account their sizes and their importance. The objective of the paper is to establish a framework for systematic examination of this idea and the exploration of the design space. We formulate the transfer of a web page as an object scheduling problem to maximize the total utility the user receives by the end of the transfer. The analysis shows that linear utility functions and their corresponding *weighted shortest processing time* (WSPT) schedule have nice properties. Next, we investigate the implementation of UCO within the HTTP framework. In particular, we introduce our proxy-based implementation that can serve both as an experimental platform and as a prototype for gradual deployment.

keywords optimization, object scheduling, utility functions, shortest processing time schedule, proxy, web, HTTP, HTML, meta-information

*This project is funded by NSF Award #9872764.

1 Introduction

Web browsing is supposed to be a highly interactive process. However, slow internet access lines, congested links, and busy servers cause the problem known as World Wide Wait. While expecting that improvement of the communication infrastructure will be necessary to solve the problem, we also believe that optimizing the use of the communication facility is a key ingredient of the solution for both today and the foreseeable future. This paper takes such optimization viewpoint of the web-browsing process. In a nutshell, we suppose the communication path from the web server to the user (or the web browser) has limited capacity. The objective is to transfer a web page on that path in the most satisfying way to the user.

Most web browsers today request images embedded in a web page in the order they appear in the HTML file. This approach ignores the fact that different images serve different purposes in a web page. Because each image carries a different meaning, they have different importance or utilities to the user. If a large but useless image appears early in the HTML file, it will take up most of the precious bandwidth while the smaller, more important images wait behind it. Following the proposal in [1], we suggest arranging the transfer order of the images by taking into account their sizes and their importance.

In this paper, we systematically examine these ideas of user-centric optimization (UCO). The main objective is to establish a discussion framework for systematic examination of the design space, including both the algorithmic and architectural issues. We start from defining utility functions that represent user's preferences over objects on web pages. Then, the transfer of a web page is formulated as an object scheduling problem that maximizes the total utility the user receives by the end of the transfer. The emphasis is on the nice properties of linear utility functions and their corresponding *weighted shortest processing time* (WSPT) schedule. Next, we investigate protocol and system issues related to implementing UCO within the HTTP framework. In particular, we introduce our proxy-based implementation that can serve both as an experimental platform and as a prototype for gradual deployment.

The most related work to our paper is [2], which experimentally evaluated the performance of the Shortest-Processing-Time first (SPT) schedule for *connection* scheduling at web servers. Our work differs from [2] in several aspects. First, our paper deals with single-user optimization. Scheduling takes place at the object level within a connection. Second, we take a more general utility-optimization approach. The resulting scheduling algorithms depend on the choice of utility functions and other constraints imposed by the problem. We discuss a few interesting utility functions, their associated schedules and various complications. In particular, the linear utility function is considered for three different situations: constant bandwidth, time-varying bandwidth and random bandwidth. Third, we give analytic expressions for the performance of the SPT schedule for exponential and Pareto distributions of the object sizes.

Our other contributions include the following. We have identified the requirements of web page transmission, and therefore, paved the way for adapting the general scheduling theory to the specific problem. In the architectural aspect, our prototype has the novel feature that it infers the utility values of images embedded in a web page based on the structure of the HTML file.

The rest of the paper is organized as follows. The remaining of section 1 discusses the service model and optimization model for the web-browsing process. Section 2 discusses three utility functions that are relevant to the problem of web object transfer. Section 3 focuses on the linear utility function and shows that the associated optimal schedules are very simple even for complex situations that might arise in practice. Section 4 presents quantitative results that demonstrate the degree of improvement from the SPT schedule. In section 5, we turn to protocol and architectural aspects of implementing object scheduling within the framework of the HTTP. We first discuss general strategies and design choices for such an implementation, then present our proxy-based prototype for user-centric web browsing. We conclude the paper in the final section.

1.1 Service Models

Communication between the user and the server in a web-based application is typically initiated when the user makes a request by clicking a link in a web page. Even though the communication is two-way, the amount of data flow is very often asymmetric with more traffic from the server to the user. Therefore, the performance of web-based applications perceived by the user is typically determined by how traffic flows from the server to the user. While acknowledging that the setup of the communication channel and the sending of the user's request also have performance implications, the focus of this paper is on data transfer from the server to the user using a given communication channel. We will refer to such a data transfer session as a *flow*.

When the data within a flow have well defined syntactic and semantic boundaries and each piece of data between two consecutive boundaries can be manipulated independently by the application, we call each piece an *object*. For instance, an object can be a JPEG or GIF image, a frame of video, a chunk of text, or an audio clip. If the encoding scheme allows it, a large object can be further divided into smaller objects. For example, one block of an image or one scan of a progressive JPEG image can also be an object. An object is intimately related to the concept of application level framing (ALF), as identified in [3]. An object can be organized as an application data unit (ADU), and an ADU can be considered as an object.

The ordering relationship among objects within a flow may or may not have significance. We call a flow with objects that are unordered or have limited orders *unordered object-based flows*. When the objects are totally ordered, the flow is called *ordered object-based flow*. When a flow appears to be a stream of bits with no obvious object boundaries within it, we call it a *stream-based flow*. In this case, the order of bits needs to be maintained.

Based on the types of services, we can categorize flows into short interactive transaction, long file transfer and stream-based media delivery. Interactive web browsing is the main example of short interactive transaction. The content to be transferred is typically authored with the requirement of interactivity in mind. However, due to the wide range of communication capacity, different users experience a wide range of delays. A transfer usually takes the form of an unordered object-based flow. This type of service typically requires eventual transmission of all data, which requires reliable communication, although in certain cases it can be argued otherwise. In the case of long file transfer, reliable communication is required but interactivity is not required. File transfer usually takes the form of stream-based flow. Unlike file transfer where the user still wishes to receive the complete file as quickly as possible, stream-based media delivery only requires enough data be to delivered at the scheduled time instances. We may distinguish between real-time and non-real-time media delivery. IP telephony or video conferencing are examples of the former, and movie delivery is an example of the latter. Most media delivery applications can tolerate some level of losses of their data packets. However, certain control information may need to be transferred reliably. The rate of traffic generation by real-time stream is limited by the codec. The traffic source for non-real-time stream is usually a large file, and can be transferred at the maximum rate of the network if uncontrolled. Both types of stream-based media delivery take the form of either ordered object-based flow or stream-based flow.

Interactive web browsing, which takes the form of unordered object-based flow, is the main focus of the paper.

1.2 Optimization Models

This paper studies how to improve user's satisfaction from a flow under a given communication channel with limited capacity. For that purpose, we will give a characterization of the user's satisfaction by assigning utility functions to a flow.

Characterization of user's perceived performance requires finding the correct abstraction or description for the downloading process. Traditionally, such abstraction includes bandwidth, delay, delay jitter and packet loss ratio, etc. User's satisfaction can be expressed as a function of these quantities. In a simple scenario, the user of the flow specifies desirable values for each (or some) of the above quantities. It is not always clear how some of the quantities are defined, whether they are the most convenient parameters to work with in practice, or what their relationships are with the user's satisfaction. For instance, the notion of bandwidth is intimately related to the time scale on which the bandwidth is measured. Delay and delay jitter are difficult to control. And we do not always know how much each unit of bandwidth or a particular delay bound corresponds to the perceived quality. In this paper, bandwidth is defined as the instantaneous rate at which data traffic is transferred.

It is also unclear how many parameters are needed to sufficiently capture the downloading process, and hence, the user's notion of quality. Alternatively, we can avoid the usual abstraction and take a more intrinsic approach. The distinction of the three types of flows is helpful in thinking about this issue. For a stream-based flow, a comprehensive characterization of the user's satisfaction is a utility function, $v(t,x)$, that depends on the entire data-reception process, i.e., when each bit of data is received. The function has the interpretation that when x bits are received by time t, the value of flow to the user is $v(t,x)$.

For an object-based flow that transfers n objects, numbered from 1 to n, we assume the utility function depends only on the reception (or completion) times of the objects and is denoted by $v(C_1,...,C_n)$, where C_i is the reception time of object i. The utility function is interpreted as the value received by the user if object i is received at time C_i, for all i. This assumption is a direct consequence of the definition of an object. Implicit in the definition, a partial object cannot be rendered by the application and brings no value to the user. From the interpretation of the utility function, it is also reasonable to assume that such a utility function is monotone in any of the reception times. More specifically, suppose $\{C'_1,...,C'_n\}$ is another set of reception times satisfying, $C_i \le C'_i$ for all i. The utility function v is called *regular* if $v(C_1,...,C_n) \ge v(C'_1,...,C'_n)$. We further assume the utility function can be written as

$$v(C_1, C_2, ..., C_n) = \sum_{i=1}^{n} v_i(C_i) \tag{1}$$

where $v_i(C_i)$ is the value of object i when it is received at time C_i.

As mentioned above, the user's satisfaction from a flow clearly depends on the network conditions. For instance, when the network bandwidth is large, the transfer is completed more quickly, and the user is more satisfied. This leads to the usual practice of using bandwidth as a measure of user's satisfaction. In fact, bandwidth is very often the only measure. Suppose the trajectory of bandwidth as a function of time is fixed for a flow. In the case of stream-based flow or ordered object-based flow, no improvement in the user's satisfaction can be made over a work-conserving schedule. For an unordered object-based flow, a key observation on which the paper is based is that, in addition to bandwidth, the user's satisfaction also depends on the downloading order of the objects on a web page. The utility function $v(C_1, ..., C_n)$ is capable of capturing the user's preferences about different ways of downloading the objects.

In this paper, performance optimization involves a single user and a single unordered object-based flow bandwidth-limited at some network location, most likely at the user's access line, at the ISP's access point to the backbone, or at the ISP's peering points. The central problem is to schedule the transmission of the objects in such a way that the utility function in (1) is maximized. With respect to the OSI layered networking model, the optimization is restricted to the application layer and the transport layer. All optimization occurs at the edge of the network, i.e., the client and the server. We do not consider optimization for the network routers. In our model, the network bandwidth is allocated to the user according to some predefined policies to which the user has no influence, for, to assume otherwise will necessarily involve the network layer. The allocated bandwidth can be deterministic or random, and can be time-varying in the deterministic case.

When the performance measure is regular, there exists an optimal schedule that is work-conserving and non-preemptive (See page 13 and 14 in [4]). A work-conserving scheduler is one that may not stop when there are unfinished jobs in the system. In a non-preemptive scheduler, once an object starts to be serviced, it will be completed before the scheduler services another object. Since the class of work-conserving and non-preemptive schedules corresponds to the set of all sequences of the n objects, the scheduling problem is equivalent to finding an optimal sequence of the n objects.

The solution to the above optimization problem can be extended to the case of a single user with multiple flows. Suppose the user is allocated a fixed bandwidth for all his flows. A reasonable schedule is to optimize the total utility. When all flows are object-based, the solution to the single-flow problem applies. It is in general difficult for the user to completely specify the utility functions for stream-based flows, since the utility function is defined on a function space of the download process. When there is a mixture of object-based flows and stream-based flows, we propose a sub-optimal approach to the problem with two levels of optimization. At the first level, the user allocates a portion of his total bandwidth to each flow in real time. He can observe the performance of each flow and change the allocation any time as he wishes. The utility function is implicitly expressed in the choices the user makes about the bandwidth at a discrete set of time instances. At the second level, single-flow optimization is applied to the unordered object-based flows under the assigned bandwidth.

2 Choices of Utility Functions and Their Optimal Schedules

In this section, our objective is to discuss possible forms of the utility functions and their corresponding optimal schedules. The optimization problem formulated above belongs to the general area of single machine scheduling. Let us assume the bandwidth μ is a constant for the moment. Let s_i be the size of object i. Let p_i be the transmission time for object i, i.e., $p_i = s_i/\mu$.

2.1 Practical Considerations for Choosing Utility Functions

By now we know the utility function for each object $v_i(C_i)$ should be a non-increasing function. We now discuss specific constraints in web object transfer that affect the choice of the utility functions.

2.1.1 Specifying Values of Utility Functions

In practice, a user faces two problems with specifying the utility functions. First, he needs to decide on a large number of function values for every C_i and for every object i. Second, since each value corresponds to an abstract notion of utility, he may find it impossible to determine the value. Paradoxically, the user might find it more natural to assign a ranking directly to each possible sequence of the objects, if he has the patience to enumerate the potentially large number of possible sequences. On the other hand, a particular permutation can be the optimal schedule for more than one utility functions. This suggests that we can approximate the true utility functions by simpler parametric functions and still derive optimal or near optimal permutation. We will later discuss three functions that can approximate a non-increasing utility function.

2.1.2 Computational Complexity

Optimal object-sequencing problems are often NP-hard for even simple utility functions. Finding polynomial-time approximation to the optimal solution can also be difficult. Due to the on-line nature of web object transmission, we prefer utility functions that do not lead to NP-hard scheduling problems or that have simple nearly-optimal solutions.

2.1.3 Feasibility of Real-time UCO

A user cannot specify his preference before the content of the web page is known, and it is not possible to do so manually in real time after the content is revealed. Hence, only known objects at known web sites can have pre-specified utility functions. The specification of utility functions is intimately related to the issue of document representation (for example, using XML [5]), which is an area of technology under active development led by the World Wide Web Consortium (W3C) [6]. Suppose the key attributes of objects and web pages can be specified using the representation technology, it is then possible for the user to define a set of policies, one for each class of objects or web pages, based on which utility functions can be automatically generated when a web page is retrieved. In this situation, the user's preferences can be communicated to the server in real time. For example, for one class of web pages, the user's policy is to see something as soon as possible; for another class of web pages, he may want to view some objects and not the others. In another example, the user can specify that all images with size between 10KB and 50KB are retrieved first. The integration of utility function specification and document representation requires further evaluation.

Instead of choosing a utility function for each object in real-time, one can take advantages of the *law of large numbers* and schedule objects according to some "expected", therefore, static utility functions. There are two large number phenomena in the current situation: the large number of users and the large number of web pages. A given a web site can learn the average utility functions based on observations on the behaviors of a large number of users. A user can choose a generic utility function for each class of web pages based on his overall experiences with a large number of web pages. In either case, the server or the user can define utility functions based on statistical knowledge learned from experiences.

2.1.4 Granularity of Multiplexing

A peculiar characteristic of the current web documents is the large imbalance of the object sizes, which can differ by orders of magnitude. Text objects can be as small as a few kilobytes and image files can be as large as hundreds of kilobytes. Large objects often dominate the transmission delay. When large objects are present, choices are limited to improve user's perceived performance. On the other hand, the optimal schedule or near optimal schedule can be found easily. For example, schedule tens of kilobytes of text ahead of a large image can increase the utility of the text with a slight decrease in the utility of the image file. Under most circumstances, this schedule is better than the schedule that transmits the image first.

If a large object can further be divided into smaller objects and each of the smaller objects can be scheduled separately, possibly subject to precedence constraint, we then have so-called fine-grained multiplexing. Through fine-grained multiplexing, the size imbalance of the web objects is reduced and fine-grained transmission control and fine-grained UCO are possible. For example, when multi-level encoding of images is possible, the user obtains certain utility for each coarse copy he receives. As additional copies are received, the image quality improves, which brings the user more value [7]. Fine-grained multiplexing depends crucially on the document encoding and presentation schemes. An interesting question is what level of granularity brings user the most value.

2.2 Constant Utility Function with Hard Deadline

In the traditional job scheduling problem, it is common for a job to have a deadline (or due date) so that costs can be associated with the violation of the deadline. We consider the following functions, v_i.

$$v_i(t) = \begin{cases} w_i & \text{if } 0 \le t \le d_i \\ 0 & \text{otherwise} \end{cases}$$

where w_i's are positive numbers, representing the values of the objects. In this model, a utility w_i is gained if the object is received before the deadline d_i. Otherwise, the object becomes useless. This utility function is more suitable for real-time media delivery than web browsing. Our problem is to maximize $\sum_{i=1}^{n} v_i(C_i)$. The problem (minimization version) is proved to be NP-hard in [8]. Lawler [9] shows a pseudo-polynomial solution based-on dynamic programming with time complexity $O(nT)$, where $T = \sum_{i=1}^{n} p_i$. Fully polynomial time approximation algorithms were found by [10] with time complexity $O(n^2/\epsilon)$,where ϵ is the desired relative accuracy of the solution from the true optimal value. In summary, one can find a good approximate solution for this optimization problem relatively easily. When all w_i's are equal, the problem is identical to the problem of minimizing the number of tardy jobs, and can be solved by Moore and Hodgeson algorithm [11] in polynomial time.

The concept of deadline for a web object can be problematic. A deadline for the transmission of a web object may not have real significance. For example, a non-real time object has no hard deadline; however, it is always preferable for it to arrive sooner. It is plausible that the time requirement for interactivity (normally around 200 ms) can be used as the deadline. However, the significance of this kind of deadline depends crucially on the transmission speed. If the link speed is so low that the transmission time of the object far exceeds the its deadline, specifying such a deadline appears to be pointless. Similarly, if we choose the values for the deadlines so large that no deadline is violated by any permutation, then any permutation is an optimal schedule.

2.3 Exponential Utility Function

An exponential utility function takes the form of

$$v_i(t) = \alpha_i e^{-\gamma t} + \beta_i$$

where $\alpha_i > 0$ and $\gamma > 0$, giving a decaying function of t. Notice that β_i's play no role in determining the optimal schedule. As long as all objects have to be processed, β_i's contribute the same constant term for any permutation of the n objects, and they do not affect the optimality of the schedule. Since we expect the value of an object eventually decays to zero, we can assume $\beta_i = 0$ for all i. With this choice of β_i, α_i can be interpreted as the initial value of the object, and $1/\gamma$ can be interpreted as a "soft" deadline. Unlike the constant function with a hard deadline, when the deadline is violated by all permutations or when the deadline is satisfied by any permutation, the exponential function still matters in the sense that most schedules are not optimal.

The optimal schedule is to sort $(\alpha_i e^{-\gamma p_i})/(1 - e^{-\gamma p_i})$ in decreasing order.

Proof: We use adjacent pair interchange argument. Suppose the optimal sequence is $\pi = (1, 2, ..., n)$. Take object i and $i+1$ in the sequence. In the optimal schedule, object i starts to be transmitted at time C_{i-1}, where we assume $C_0 = 0$. Starting with the optimal sequence, switching the position of object i and $i+1$ decreases the utility of object i by

$$(\alpha_i e^{-\gamma(C_{i-1}+p_i)} + \beta_i) - (\alpha_i e^{-\gamma(C_{i-1}+p_{i+1}+p_i)} + \beta_i) = \alpha_i e^{-\gamma C_{i-1}} e^{-\gamma p_i}(1 - e^{-\gamma p_{i+1}})$$

and increases the utility of object $i+1$ by

$$(\alpha_{i+1} e^{-\gamma(C_{i-1}+p_{i+1})} + \beta_{i+1}) - (\alpha_{i+1} e^{-\gamma(C_{i-1}+p_{i+1}+p_i)} + \beta_{i+1}) = \alpha_{i+1} e^{-\gamma C_{i-1}} e^{-\gamma p_{i+1}}(1 - e^{-\gamma p_i})$$

Since the sequence π is optimal, the total change of utility should be non-positive. Therefore,

$$-\alpha_i e^{-\gamma C_{i-1}} e^{-\gamma p_i}(1 - e^{-\gamma p_{i+1}}) + \alpha_{i+1} e^{-\gamma C_{i-1}} e^{-\gamma p_{i+1}}(1 - e^{-\gamma p_i}) \leq 0$$

Equivalently,

$$\frac{\alpha_i e^{-\gamma p_i}}{1 - e^{-\gamma p_i}} \geq \frac{\alpha_{i+1} e^{-\gamma p_{i+1}}}{1 - e^{-\gamma p_{i+1}}}$$

■

One potential drawback with the exponential function is that a single parameter γ is used for all objects. That is, all objects have the same deadline.

2.4 Linear Utility Function

Let the utility of each object i be,

$$v_i(t) = \beta_i - \alpha_i t$$

where $\alpha_i > 0$, and $\beta_i \geq 0$ for all i. In one interpretation, the value of object i starts at β_i at time 0 and decreases linearly with the reception time at a slope α_i. The optimal schedule is to transmit objects in the increasing order of p_i/α_i. It has a time complexity $O(n \log n)$. Since

$$\max \sum_{i=1}^{n} (\beta_i - \alpha_i C_i) = -\min \sum_{i=1}^{n} (\alpha_i C_i - \beta_i) \tag{2}$$

the optimization objective is in effect to minimize the sum of weighted completion time. In the traditional scheduling terminology, the optimal schedule is called *Weighted Shortest Processing Time* (WSPT) schedule, which is a generalization of the SPT schedule. In this traditional view, the α_i's can be interpreted as the weights assigned to the objects. Note that β_i's are irrelevant for finding the optimal schedule.

It is well known that the SPT schedule is optimal for a variety of performance measures [11] [4]. Suppose each object i has a deadline d_i. Define $L_i = C_i - d_i$ to be the *lateness* of object i, $T_i = \max\{C_i - d_i, 0\}$ to be the *tardiness*, and $N(t)$ to be the number of unfinished objects by time t. The SPT schedule minimizes the mean lateness, $\bar{L}$, and the mean number of unfinished objects, $\bar{N}$. Moreover, if all objects have the same deadline, the SPT schedule minimizes the mean tardiness, $\bar{T}$. If it is impossible for any object to be on time in any sequence, then $\bar{T}$ is minimized by the SPT schedule; if the SPT schedule yields no objects on time, then it minimizes $\bar{T}$.

In the special context of web object transfer, the SPT schedule has many more advantages.

- It is suitable in many situations when the notion of due dates and values of the objects are vague.
- Given a fix time, it completes more objects than any other scheduler. This is especially beneficial when a page consists of many small objects and one or a few large objects, because the large objects are pushed to the end of the transmission sequence and most objects will arrive during the early period of the transfer session.
- It is friendly to fine-grained encoding of document. For instance, in the case of multi-resolution encoding, the size of the lower resolution object is small, and can arrive early and be displayed quickly. It possible that, in many situations, most of the value is delivered to the user at this point.
- Users do not need to specify any utility functions.

We will have more to say about the performance of the SPT schedule in a later section. Many of the advantages of the SPT schedule extend to the WSPT schedule, making the linear utility function the most interesting one.

3 Linear Utility Function - A Detailed View

In this section, we look at the linear utility function in more complex but practical situations and show the simplicity of the optimal schedules.

3.1 Linear Utility Function and Precedence Constraint

Up to now, we have assumed no precedence constraint among objects. In other words, any permutation of the objects gives rise to a valid transmission sequence. It turns out that, for a large class of precedence constraint, the optimal schedule for linear utility functions has polynomial-time algorithms . Typically, precedence constraint is represented on a directed acyclic graph (DAG) $G = (N, A)$, in which each node $i \in N$ represents an object and each arc $(i, j) \in A$ represents the precedence constraint that object i must be processed before object j. It can be shown that scheduling under precedence constraint that takes the form of a series-parallel DAG can be solved by a polynomial-time algorithm [12]. The precise definition of a series-parallel DAG can be found in [12]. The class of series-parallel digraph is large enough to include chains, outtrees, where every node has no more than one predecessor, and intrees, where every node has no more than one successor. By definition, the series or parallel compositions of the above three types of DAGs are also series-parallel.

We argue that the class of series-parallel DAG is adequate to represent precedence constraint at different levels of granularity for the web objects. We start with the coarsest level where each web object is a complete image, audio or video file, or text, etc. In other words, each web object has well defined semantic boundary. At this level, the number of objects on a typical web page is small, and the precedence relations among objects are typically simple, maybe taking the form of chains, outtrees and intrees in parallel. At a finer granularity, each of the above objects can be further divided into smaller objects. Let us call a set of objects originated from the same larger object an object group. If the precedence constraint among these finer objects in each object group can be represented by series-parallel DAGs, then the precedence constraints among all objects can be represented by a series-parallel DAG.

3.2 Linear Utility Function and Random Transmission Times

In the previous sections, we have assumed the transmission time of each object is a constant. In this section, we consider the case where the transmission times are random variables. Given the n fixed-sized objects to be transmitted, the transmission times can be random due to a few factors. For instance, the bandwidth in the transmission pipe can be modeled as a random variable. The random delay in the network is also part of the transmission time. Another situation is when a packet is lost in the network, retransmission of the lost packet will take additional random amount of time.

Let us denote the random transmission or processing time of object i by X_i, $i = 1, 2, ..., n$, which are not necessarily independent of each other. Since the object completion times are random, the objective is to schedule the n objects in order to optimize the expected utility. Unlike the deterministic scheduling case, we need to consider two classes of scheduling policies in the stochastic case.

Definition 3.1 *[13] In a* ***non-preemptive static policy****, the complete schedule is determined at time 0, and will not change for the entire processing session of the n objects. Non-preemption has the usual meaning.*

Definition 3.2 *In a* ***non-preemptive dynamic policy****, every time the server is free, the next object to be transmitted is determined using all the currently available information. Non-preemption has the usual meaning.*

The reason to make the above distinction is that the stochastic process $\{X_1, X_2, ..., X_n\}$ yields more information as they are observed, and therefore, the *dynamic policy* corresponds to a larger classes of scheduling algorithms than the *static policy*. It is shown in [13] that the sequencing rule that follows the increasing order of EX_i/α_i maximizes the expected value of the linear utility function in the class of non-preemptive static policies and in the class of non-preemptive dynamic policies.

The optimality of the above sequencing rule goes further. It can be shown that it is optimal also in the class of *preemptive dynamic policies* when all distributions of the processing times have *increasing completion rate* [13], where the *completion rate* of object i, $c_i(t)$, is defined by

$$c_i(t) = \frac{f_i(t)}{1 - F_i(t)}$$

Here, we assume that the processing time of object i, X_i, is a continuous random variable with distribution F_i and density f_i.

In the deterministic model, the processing time of object i is simply $p_i = s_i/\mu$, where μ is the constant bandwidth. In the case of random transmission times, it is not a trivial task for the server to know the processing time distributions, which is essential for forming the optimal schedule. We hope to leverage on the only piece of information that is certain, the object sizes. Suppose the processing time distributions are such that the expected processing times are in agreement with the object sizes, i.e.,

$$\frac{EX_p}{EX_q} = \frac{s_p}{s_q} \tag{3}$$

for any pair of $p, q \in \{1, 2, ..., n\}$. Then, in the case of the linear utility function, the sequence that follow the increasing order of s_i/α_i is optimal for both the deterministic case and the stochastic case in the class of *non-preemptive static or dynamic policies*.

It is reasonable to believe that the condition in (3) can be satisfied in many realistic situations. We propose the following model to justify this. Suppose the basic transmission unit is a packet of a fixed size, and suppose the processing times for packets are independently and identically distributed. Consider two objects p and q. Let Z_p^k, Z_q^k be the processing times for the k^{th} packet for object p and q, respectively. Let s_p and s_q be the sizes of the objects in number of packets for object p and q, respectively. Then, the total processing times are $X_p = \sum_{k=1}^{s_p} Z_p^k$ and $X_q = \sum_{k=1}^{s_q} Z_q^k$, and condition in (3) is satisfied.

3.3 Linear Utility Function and Time-Varying Bandwidth

Suppose the bandwidth is deterministic but time varying and piece-wise continuous, denoted by $\mu(t)$, and suppose the size of the n objects are fixed. It is still true that, for a regular utility function, there exists an optimal schedule that is (i) work-conserving and (ii) non-preemptive. (iii) The optimal sequence in general depends on the bandwidth trajectory. For many utility functions, the scheduling problem becomes very hard. (iv) For linear utility functions with identical slope, an optimal schedule is to order the objects by their sizes in increasing order, which is independent of the bandwidth trajectory. We will give justifications for these claims.

Suppose that in an optimal schedule, the server has a period of inactivity from time t_1 to t_2. Then, eliminating the inactive gap by moving all objects scheduled after time t_2 ahead does not decrease the utility, since the utility function is non-increasing in the reception times. This proves (i).

Now, suppose π is a work-conserving optimal schedule, in which object i is preempted by other objects before its completion. Let t_f be the completion time of object i in π. Let s_i be the size of object i and define

$$t^* = \sup\{t : \int_t^{t_f} \mu(\tau) d\tau = s_i\}$$

Let π^* be the new schedule in which object i starts service at time t^* and ends at t^f. Complete or partial objects that preempt i in π are moved ahead without changing their relative orders, filling the void left by object i. The new schedule π^* is at least as good as π, and hence is also optimal. This procedure can be repeated for each object to get an optimal non-preemptive schedule, and hence, (ii).

To show (iii), consider a case with two objects. Let the size $s_1 = 5$ and $s_2 = 10$. The utility function for each object is defined as follows.

$$v_1(t) = \begin{cases} 10 & \text{if } 0 \leq t \leq 3 \\ 0 & \text{otherwise} \end{cases}$$

$$v_2(t) = 20 - 2t$$

Let the schedule $\pi^1 = \{1, 2\}$ and $\pi^2 = \{2, 1\}$. Consider a bandwidth trajectory $\mu^1(t) = 5$. The total utility gained from each schedule is: $g(\pi^1) = 24$ and $g(\pi^2) = 26$. Hence, π^2 is the optimal sequence. Consider another bandwidth trajectory $\mu^2(t) = 2.5$. In this case, $g(\pi^1) = 18$ and $g(\pi^2) = 12$. Hence, π^1 is the optimal sequence.

Now suppose all objects have a linear utility function of the form $v_i(t) = \beta_i - \alpha_i t$, where $\alpha_i > 0$ and $\beta_i \geq 0$. Given a fixed bandwidth trajectory $\mu(t)$ and an optimal sequence π, let us suppose object i and j are adjacent in π and i proceeds j. Let i starts service at time t_1, ends service at t_2 and let j ends service at t_3. Let π' be a sequence with object j and i interchanged. In π', object j starts service at t_1 and ends service at t'_2, and object i ends service at t_3. Since π is optimal, we must have,

$$\alpha_i t_2 + \alpha_j t_3 \leq \alpha_j t'_2 + \alpha_i t_3$$

which yields,

$$\frac{t_3 - t'_2}{\alpha_i} \leq \frac{t_3 - t_2}{\alpha_j} \tag{4}$$

Let us denote $p_i(t)$ as the service time for object i when it ends service at time t. We then have $p_i(t_3) = t_3 - t'_2$ and $p_j(t_3) = t_3 - t_2$. Suppose all α_i's are identical. Then, (4) becomes

$$p_i(t_3) \leq p_j(t_3) \tag{5}$$

Since at any time t, $s_i \leq s_j$ if and only if $p_i(t) \leq p_j(t)$, it suffices to sequence the n objects according to the increasing order of their sizes. This demonstrates (iv).

When α_i's are not identical, optimal sequencing seems to be a very difficult problem. The condition in (4) is only necessary but not sufficient for optimality. The following algorithm can find us a "locally" optimal sequence in the sense that the resulting sequence cannot be improved by exchanging positions of two neighboring objects.

Let t_n be the completion time for the entire transfer session. The last object to be transmitted, denoted by $\pi(n)$, should have the largest value of $p_i(t_n)/\alpha_i$ of all objects. Suppose we have determined the last k objects to be transmitted, $\pi(n-k+1), \pi(n-k+2), ..., \pi(n-1), \pi(n)$. Let t_{n-k+1} be the completion time of object $\pi(n-k+1)$. Then, $t_{n-k} = t_{n-k+1} - p_{\pi(n-k+1)}(t_{n-k+1})$ is the starting time of object $\pi(n-k+1)$. Then, the $(n-k)^{th}$ object, $\pi(n-k)$, should have the largest value $p_i(t_{n-k})/\alpha_i$ among the remaining objects yet to be scheduled. The total running time is $O(n^2)$ for this algorithm, where n is the number objects.

In reality, the complete bandwidth trajectory may not be known ahead of time. In that case, one can modify the above algorithm as follows. After finishing transmission of an object at time t, the object with the smallest value of $\frac{s_i/\mu(t)}{\alpha_i}$ among all remaining objects is chosen for the next transmission, where $\mu(t)$ is the bandwidth at time t. The modified algorithm also applies when the bandwidth is random but with unknown statistics.

4 Quantitative Evaluation of SPT Schedule

The previous sections mainly establish the optimality of the WSPT schedule in a variety of situations. In this section, let us assume all objects have equal weight and evaluate quantitatively the improvement of the SPT schedule over a random schedule. To model the case where a user retrieves a random page with n objects, let us suppose the transmission times of the objects are independent and identical random variables drawn from a distribution F. We will consider two distributions: the exponential distribution and the Pareto distribution. For each realization of the transmission times, we apply the SPT schedule and the random schedule, which simply transmits the n objects in the order they are given. The performance criteria is the expected mean completion time $E\bar{C_n} = \frac{1}{n}\sum_{i=1}^{n} EC_i$. We use $\bar{C_n}$ as the mean completion time to emphasize its dependence on n, the number of objects. We already know that SPT schedule minimizes $\bar{C_n}$, and therefore, $E\bar{C_n}$. Let $X_1, X_2, ..., X_n$ be the transmission times for the objects. Let $X_{(k)}$ be the k^{th} order statistics of $\{X_1, X_2, ..., X_n\}$, i.e., $X_{(1)} \le X_{(2)} \le ... \le X_{(n)}$. In the case of SPT schedule, $\bar{C}_n^{SPT} = \sum_{i=1}^{n}(n-i+1)X_{(i)}/n$. Let the transmission bandwidth μ be 1 so that the file size and the transmission time coincide.

4.1 Exponential Transmission Times

Suppose each X_i is distributed exponentially with mean $1/\nu$, for $i = 1, 2, ..., n$. The first moment of any order statistics has a closed-form expression (see page 49 in [14]).

$$EX_{(k)} = \frac{1}{\nu}\sum_{i=1}^{k}\frac{1}{n-i+1} \tag{6}$$

for $k = 1, 2, ..., n$. It can be shown that, for SPT schedule,

$$E\bar{C}_n^{SPT} = \frac{n+3}{4}\frac{1}{\nu} \tag{7}$$

For the random schedule

$$E\bar{C}_n^{RAND} = \frac{n+1}{2}\frac{1}{\nu} \tag{8}$$

The expected mean completion time in the SPT schedule is about half of that in the random schedule. Since the reception time for the last object is the same in both schedules, it must be true that some other objects arrive earlier in the SPT schedule. In fact, for each realization of transmission times, SPT schedule finishes more objects than any other schedule at any time. The user may feel objects arrive faster in the SPT schedule.

It is not surprising that, in both schedules, $E\bar{C}_n$ is linear in the number of objects, as the expected duration of the entire transfer session also increases linearly with the number of objects. In the case of fine-grained multiplexing, suppose each object is further divided into m smaller and identical objects. The mean size of each smaller object becomes $1/(m\nu)$. Hence, $E\bar{C}_n$ is not affected by fine-grained multiplexing for large n.

4.2 Pareto Transmission Times

Previous statistical studies on the web file sizes and on the sizes of actual http transfers give strong indication that the object sizes have heavy-tail distribution [15]. In this subsection, we will assume the object sizes have Pareto distribution with c.d.f. $F(x) = 1 - K^{\alpha} x^{-\alpha}$, where $x \geq K$ and $\alpha > 0$. For our purpose, we further require $\alpha > 1$, in which case the mean of the distribution exists and is equal to $\frac{\alpha K}{\alpha - 1}$. The first moments of the order statistics for the transmission times are, (see page 50 in [14])

$$EX_{(k)} = \frac{n!}{(n-k)!} \frac{\Gamma(n-k+1-1/\alpha)K}{\Gamma(n+1-1/\alpha)} \tag{9}$$

for $k = 1, 2, ..., n$. In the above, for $s > 0$, the gamma function is defined by,

$$\Gamma(s) = \int_0^{\infty} e^{-x} x^{s-1} dx$$

We will evaluate $E\bar{C}_n$ numerically for a few realistic cases. In case 1, we pick $\alpha = 1.1$ and $K = 100$ (bytes) for the Pareto distribution. In this case, the mean object size is 1100 bytes. In case 2, we pick $\alpha = 1.5$ and $K = 100$, and the resulting mean object size is 300 bytes. The expected mean completion times $E\bar{C}_n$ for the SPT schedule and the random schedule are shown in figure 1 and figure 2 for the two cases. The values for $E\bar{C}_n$ are approximately linear in n. In the first case, SPT schedule has a much smaller value for $E\bar{C}_n$ than the random schedule, with $E\bar{C}_n^{SPT} \approx \frac{1}{6} E\bar{C}_n^{RAND}$. In the second case, $E\bar{C}_n^{SPT} \approx \frac{1}{2} E\bar{C}_n^{RAND}$. The improvement of SPT schedule depends on the actual statistics of the object sizes.

To understand how the performance improvement of the SPT schedule depends on the parameters for the Pareto distribution, we resort to an asymptotic theorem regarding linear combinations of order statistics, found in [16]. Let us define $S_n = \bar{C}_n/(n+1)$, and a function $J(u) = 1 - u$, for $0 \leq u \leq 1$. Then, $S_n = \frac{1}{n} \sum_{i=1}^{n} J(\frac{i}{n+1}) X_{(i)}$ in the SPT schedule. Theorem 3 in [16] says, if $E \mid X_i \mid < \infty$ and $J(u)$ is bounded and continuous, then as $n \to \infty$, $ES_n \to \theta(J, F)$, where

$$\theta(J, F) = \int_0^1 J(u) F^{-1}(u) du = \int_{-\infty}^{\infty} x J(F(x)) dF(x) \tag{10}$$

When $F(x)$ is Pareto, it is easy to show

$$\theta(J, F) = \frac{\alpha K}{2\alpha - 1}$$

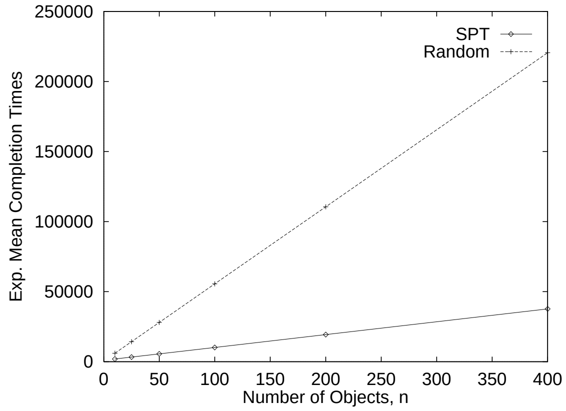

Figure 1: $K = 100$, $\alpha = 1.1$

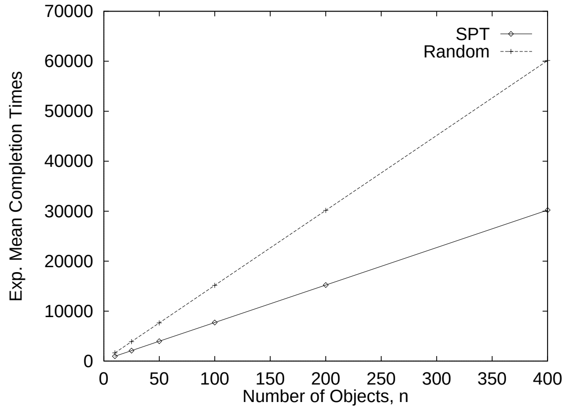

Figure 2: $K = 100$, $\alpha = 1.5$

Hence, for large n, $E\bar{C}_n^{SPT}$ grows approximately linearly with $n + 1$ at a slope $\theta(J, F)$. As $\alpha \downarrow 1$, i.e., as the distribution becomes more heavy-tailed, the slope approaches K; as $\alpha \to \infty$, the slope approaches $K/2$. In the random schedule, $E\bar{C}_n^{RAND} = \frac{\alpha K}{2(\alpha-1)}(n+1)$. As $\alpha \downarrow 1$, its slope approaches ∞; as $\alpha \to \infty$, its slope approaches $K/2$.

A useful figure of merit can be defined by $\gamma = E\bar{C}_n^{SPT}/E\bar{C}_n^{RAND}$, which is approximately $\frac{2(\alpha-1)}{2\alpha-1}$ for large n. As the file size becomes more and more heavy-tailed, the expected mean completion time for the SPT schedule becomes an increasingly smaller fraction of that for the random schedule. This is in sharp contrast with the case of exponential file sizes where the same ratio, approximately $1/2$, does not depend on the parameters of the distribution. As examples for the Pareto case, for $\alpha = 1.5$ or 1.1, $\gamma = 1/2$ or $1/6$, respectively. These numbers are in agreement with figure 1 and figure 2. The figures also indicates that the approximation by the asymptotic result becomes fairly accurate for small n.

Since $0 \leq J() \leq 1$, from (10), we know

$$\theta(J, F) \leq \int_{-\infty}^{\infty} x dF(x) = EX_i$$

Hence, if we keep the mean file size the same, the worst case distribution in terms of the expected mean completion time is the deterministic distribution.

Let us try to understand some peculiar features of the Pareto transmission times. Among n independently and identically distributed Pareto random variables, the maximum is a special one. This can be seen by looking at the quantity $EX_{(k+1)}/EX_{(k)}$, for $1 \leq k < n$. By using (9) and the familiar

$$\Gamma(s+1) = s\Gamma(s) \qquad \text{for } s > 0$$

we get,

$$\frac{EX_{(k+1)}}{EX_{(k)}} = \frac{n-k}{n-k-1/\alpha}$$

As $\alpha \downarrow 1$, $EX_{(n)}/EX_{(n-1)}$ increases to infinity. On the other hand, $EX_{(k+1)}/EX_{(k)} < 2$ for $1 \leq k < n-1$, and for most values of k, the ratio is in fact close to 1. In other words, the value of $EX_{(k)}$ increases very slowly except for the last few k. It has a sudden upward jump at $k = n$.

Depending on α, the expectation for the maximal value can be significantly greater than that for any other order statistics. For the values of α and K that are relevant to our problem and when n is not so large, the maximum is often larger than or comparable to the sum of the rest random variables. For instance, when $K = 100$, $\alpha = 1.1$ and $n = 50$, $EX_{(n)} = 36839$ and $E\sum_{i=1}^{n-1} X_{(i)} = 18160$. We are led to consider an even simpler schedule: move the largest object to the last position and leave the rest of objects where they were. We call this schedule Largest-To-Last (LTL). It is expected to perform well when the object sizes are very heavy-tailed and when the number of objects is not so large. To quantify its performance, note that

$$E[X_i \mid X_i < X_{(n)}] = \frac{nEX_i - EX_{(n)}}{n-1}$$

$$\begin{aligned} E\bar{C}_n^{LTL} &= \frac{1}{n}\Big(\sum_{k=2}^{n} kE[X_k \mid X_k < X_{(n)}] + EX_{(n)}\Big) \\ &= \frac{n+1}{2}EX_i - \frac{1}{2}EX_{(n)} + \frac{1}{2n}EX_{(n)} \end{aligned} \qquad (11)$$

(11) is valid for any distributions for which the expectations are defined. It indicates that the LTL schedule should greatly improve over the random schedule when the maximum is comparable to the sum, which is often the case for the web documents. Table 1 shows the performance comparison for the three schedulers. When the number of objects is less than 50, the LTL scheduler is nearly optimal.

Table 1: Performance comparison for SPT, LTL and random schedulers

α	n	$E\bar{C}_n^{SPT}$	$E\bar{C}_n^{LTL}$	$E\bar{C}_n^{RAND}$
1.1	10	1925	2199	6050
	50	5592	9999	28050
	100	10175	21321	55550
1.5	10	975	1084	1650
	50	3975	5864	7650

4.3 Experimentation

In the performance analysis for the object transmission schedules, we have assumed that the mean completion time is a reasonable performance measure for typical users. With this performance measure, the advantage of the SPT schedule over the random schedule is fairly compelling. One may wonder if the improvement of this performance measure corresponds to perceived performance improvement during the actual browsing session. We have conducted some experiments to verify this. In the experiments, we randomly draw a number of objects from a repository of objects and display them. The objects are mainly GIF or JPEG images collected from our Web browser's disk cache. It is reasonable to believe that their size variation is typical, i.e., following the true distribution of object sizes in web pages. To emulate the transmission delay due to the finite bandwidth, we artificially delay the display of each object according to the bandwidth setting and the object size. In our experiment, the bandwidth can take values from the 28.8 Kbps modem speed to 10 Mbps. In each experiment run, we keep the bandwidth constant. The objects selected are displayed according to the SPT schedule, and then, according to the random schedule for comparison. We have perceived noticeable performance improvement with the SPT schedule, particularly at the small bandwidth values. Even at 1.5 Mbps or higher, data transfer according to the SPT schedule typically appears much more responsive than the random schedule, giving the user a more pleasant feeling. Interested readers can find the experiment at [17].

5 Proxy-based Implementation of UCO within HTTP

5.1 Overview

At the present, TCP and UDP are the standard transport protocol, and HTTP [18] [19] is the standard application protocol that web browsers use to communicate with web servers. In order to verify that UCO can be integrated with HTTP and to gain some insights into the implementation and deployment of UCO, we built a prototype for user-centric web browsing. This prototype serves both as a feasibility proof for UCO within the HTTP framework and a visual proof that UCO increases users' satisfaction in web browsing.

HTTP is a request-response protocol. In a typical scenario, a client wishing to "open" an URL initiates a new TCP connection to the server and issues an HTTP GET request for the specified URL. Upon receiving the request, the server replies with an HTTP header informing the client of the request status, followed by the content of the requested resource. In HTTP/1.0, the connection is usually closed immediately unless the `Keep-Alive` option is used. HTTP/1.1 allows the same TCP connection to be reused for multiple requests. Most web sites are organized as pages hyperlinked to each other. Each page is represented by a base HTML with anchors pointing to zero or more embedded objects. Embedded objects range from images to audio and video clips. In the rest of the paper, we will focus on optimizing the download of images only.

Implementing UCO in HTTP imposes restrictions on the kinds of optimizations that are possible. Because HTTP runs on top of TCP, which holds out-of-order data to guarantee in-order delivery, unnecessary delay is introduced. Moreover, HTTP requests usually have the granularity of a file; re-ordering can therefore happen only at the coarse granularity of a file. Although HTTP/1.1 allows requests for a particular range of bytes of an URL, the lack of a flexible, meaningful naming scheme makes it difficult for clients to make requests at a granularity finer than an URL.

5.1.1 Location of the Object Scheduler

The determination of the schedule can be at either the server or the client. In the client-based approach, the client requests different objects in the order that maximizes user satisfaction, and the server serves requests in a first-come-first-serve(FCFS) fashion. If the server does not preserve a FCFS order, the client can still enforce a particular order by allowing no more than one HTTP request outstanding at anytime. This, however, reduces efficiency because requests cannot be pipelined to hide the network latency.

In the server-based approach, the client requests the embedded objects as usual, but the server is allowed to return objects in an order different from the request order. This model is incompatible with HTTP/1.1 because, by default, the client issues all of its requests over the same TCP connection to the HTTP server, and the server must send its responses to those requests in the same order that the requests were received. In HTTP/1.0, a client usually opens one connection per object. The server then has the freedom to choose the order in which the requests are served. In this approach, the client needs to describe and inform the server about its preferences over the objects.

In the spirit of UCO, it seems that the actual scheduling should be at the client because it knows about the user's preferences. On the other hand, the server knows about the content of the web pages. The client-based approach scales better than the server-based approach because the computation of the optimal schedule is done by the clients. In the rest of the paper, we focus on the client-based approach.

5.1.2 Meta-information

Regardless of the mechanism used to enforce a particular schedule, the client normally needs to obtain enough information about each of the objects to be downloaded before an optimal schedule can be determined. This piece of required information affects neither the content nor the presentation of the object; it is only used to carry out the cost-benefit analysis for determining the optimal download order. We call this piece of information the *meta-information* about an object. The exact data included in the meta-information depends on the specific utility function chosen. However, the size of the object, the media type, and the semantic importance of the object should be included. It is desirable for the client to receive the meta-information for each object as soon as possible. At the earliest, the client can request it at the same time the HTML file is requested.

Without modifying the HTTP, there are at least two options for setting up the meta-information. One is to embed meta-information directly in the HTML file and encode it as attribute name and value pairs in the HTML tags. This scheme requires extending current HTML specification. We can invent new attributes for an `<IMG>` tag to store the object size, media type, and semantic importance. An image tag might look like the following:

```
<IMG src="http://www.xyz.com/abc.jpg" ... object_size="1134Bytes"
 media_type="gif" semantic_importance="high">
```

Another option is to store meta-information associated with all embedded objects of an HTML file in a dedicated file whose name can be easily derived from the name of the HTML file. For example, we can name the meta-information file by appending ".meta-info" to the name of the HTML file. Placing meta-information in separate files requires no changes to the HTML specification. Under this scheme, only those clients who knows about the existence of meta-information request it.

One major complexity of user-centric web browsing is to develop a rich, yet compact, summary of the semantic information of objects. There have been major efforts to develop a new language called XML for authoring web pages in a way that separates the semantics of a page from the representation of a page [5]. XML has language structures that allow machines to more easily extract semantic information from web pages. Therefore, XML can potentially be a solution to the problem.

While setting up the meta-information normally requires human intervention, certain meta-information such as object (file) sizes can be generated automatically by the computer. In this case, the meta-information can either be derived when a client requests the web page or when the HTML file is modified. The former has the added advantage that it works even if the page is dynamically generated. Whenever the URL corresponding to the meta-information file is requested, a special CGI program is started to extract the meta-information from the relevant objects and to return them to the client.

5.2 Proxy-based Implementation Strategy

In order to implement the client-based approach to UCO, the web browser should be able to

1. retrieve meta-information automatically at the time when the HTML file is requested;
2. collect user-preferences;
3. compute the optimal object download schedule;
4. enforce a download order that matches the optimal schedule.

Server software does not need to be changed, but should provide meta-information about the objects by either adding extra attributes in the HTML files, or by serving special meta-information files.

At the client side, the straightforward approach to deploying user-centric web browsing is to implement a brand new browser or modify existing browsers. However, at this proof-of-concept stage, the initial overhead is still substantial. For the purpose of rapid development and quick testing, we have chosen a proxy-based implementation at the client side. It turns out that the idea is fairly general, because we can use the proxy for both object scheduling and for inferring the meta-information. This approach allows us to carry out UCO even without the support of the content provider or the server. Hence, the proxy-based implementation also serves as one of the possible migration paths for deploying UCO. We have developed some heuristic for inferring meta-information for images from the structure of the base HTML file. More details are given in section 5.3.1

5.2.1 Client-Support by Proxy

Item 1 to 4 can be achieved by using a client-side HTTP proxy that runs on the same machine as the browser. When the browser requests for a base HTML file of a web page, the proxy automatically requests the associated meta-information file from the server assuming that the meta-information is stored in separate files (or equivalently, referenced by a different URL.) If the meta-information is stored directly within the HTML file, the proxy needs only to extract such information. Thus, item 1 is achieved. Item 2 can be achieved by defining a set of special URLs that are intercepted and serviced by the proxy directly. For example, we can stipulate that all requests of the form:

`http://localhost/proxy-user-interface-service/optionname`

are serviced by the proxy. The proxy can return HTML forms for the user to view and configure his preferences within a browser window. These special URLs can be organized as hierarchical bookmarks in much the same way nested menus are organized in browsers. Given the meta-information and user preferences, the client-side proxy can easily accomplish item 3. Item 4 requires the client to cooperate with the client-side proxy. The proxy must not allow the client to use a persistent connection or the `Keep-Alive` option in HTTP. Otherwise, the client will only open one connection to the proxy. As a result, the proxy cannot affect the order of object download because it must return objects in the same order as the requests. The proxy can avoid this situation by pretending to be either HTTP/1.0 compliant only or by not supporting persistent connections. Note that persistent connection should be implemented by all HTTP/1.1 compliant proxies, but is not required.

Since proxies do not have any control over the browsers, the proxy-based approach has some limitations in the types of possible optimization. First, the proxy cannot capture user events that may be helpful in dynamically estimating the utilities of objects. For example, the proxy cannot find out which region of a web page is currently visible, and is unable to load the on-screen objects first. Second, the proxy cannot request and return an image that, it believes, has a high utility until the client explicitly requests it. Third, to allow proxy enough freedom for object re-ordering, the client should open as many connections and issue as many simultaneous requests as possible to the proxy. Unfortunately, most popular web browsers artificially limit the number of simultaneous connections to a hard-coded small number, usually four. Nonetheless, writing special proxies is still the easiest solution to implementing UCO because it allows us to focus on download ordering by insulating us from the immense complexity of parsing HTML, executing inline scripts, laying out page properly, and interacting with user events.

5.3 The Design of Proxy

The operations of the proxy are shown in Figure 3. Because web pages currently do not provide any explicit meta-information, our prototype tries to infer the meta-information from the base HTML file. In most cases, the information stored in HTML files gives the client enough hints to infer the role of images. Once the role of an image (e.g. menu, thumnail, decoration, or advertisement) is determined, a utility value is assigned to the image according to its role. In principle, the size of the file can also be inferred from the on-screen dimension of the image. At the present, our prototype does not have this capability. Our proxy enforces an image-download schedule in descending order of images' utility values. All of the above steps are carried out transparently to the browser. The client-side proxy resides on the same machine as the client.

5.3.1 Heuristic for Image Classification

In this section, we will examine the image-classification algorithm in details. In particular, we try to understand why it is possible to infer the roles of images in the first place. The goal of the classification algorithm is to determine which of the following categories best describes each image.

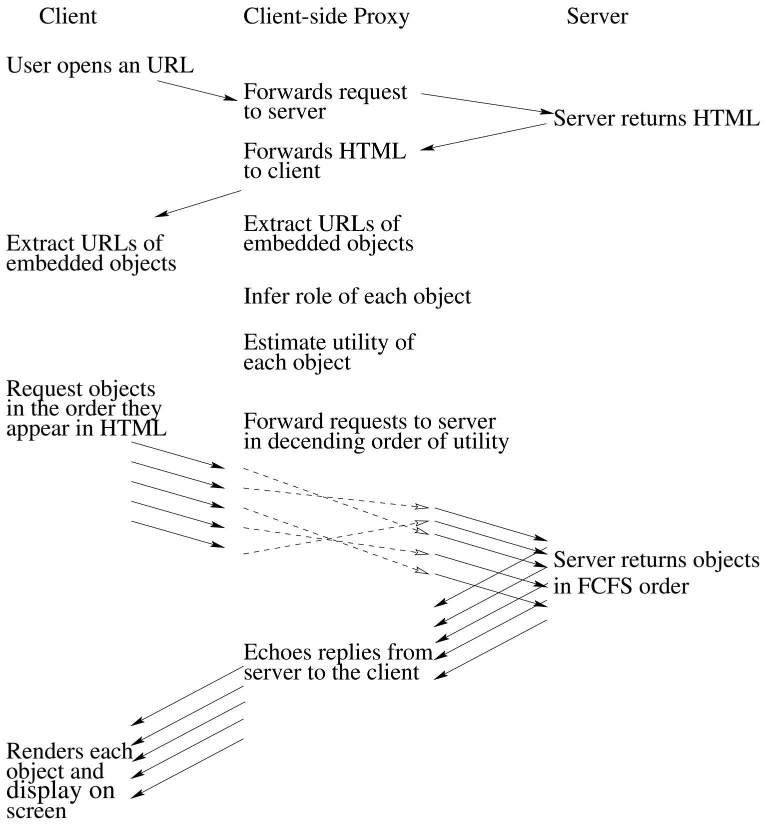

Figure 3: Operational steps of UCO prototype

Table 2: Image categories

Category	Description
1. Menu	an image that is a part of or an entire navigation menu for a web site
2. Thumbnail	a small image that is a link to a high-resolution version of the same image
3. Decoration	an image for decoration purposes only and has little or no semantic importance
4. Advertisements	an image used for advertising purposes
5. Button	an image used as a button for submitting data to the server
6. Photos	an image that conveys significant semantic information
7. General Links	a graphical representation of a link
0. Other	an image of unknown nature

In order to infer the role of an image in a web page, we must look beyond the URL of the embedded image. The key to successful classification relies on the ability to extract relevant information from the context in which the image is embedded. For instance, from the context of the image reference within the HTML, we can find out if an image is a hyperlink. The fact that an image is a hyperlink tells us it is unlikely to be a decoration image.

The heuristic one chooses can be arbitrarily complex. However, in practice, even some simple rules are already very useful. Here are some information we can extract from an HTML file that can help us infer the purpose of an image in a page.

File Extension GIF files usually have `.gif` as an extension and are mostly used for computer-generated graphics. JPEG files have the `.jpg` or `.jpeg` extensions and are good for continuous-tone pictures. The JPEG files are likely to be photos that carry important semantic information.

Link URL Some images serve as hyperlinks. The structure of the link URL gives us valuable information about the image itself. This URL

```
http://cnn.com/event.ng/Type=click&ProfileID=1021&RunID=19077&AdID=14352&
GroupID=369&FamilyID=3095&TagValues=434.435.487.1009.1696&Redirect=http:%
2F%2Fadcenter.in2.com%2Fcgi-bin%2Fclick.cgi%3Ftid=15427%26cid=about-home2
-468x60%26hid=cnn%26time=1999.11.18.6.25.42.0
```

is the link URL of an image on a CNN web page. We can recognize from the URL that it is a computer generated URL pointing to a CGI program that redirects the users to a different site when they click on the image. Typically, the web site displaying an advertisement image embeds its identity and the destination URL directly in the image URL, so that the destination server can properly credit the referring web site for redirecting users. Although different companies have different naming scheme for tracking referrer web sites, the characteristics of the link URL of an advertisement can hardly be mistaken. More generally, if the link URL points to a site from a different domain than the domain that hosts the HTML file, the image is likely to be an advertisement.

Dimension Though not required, most web sites nowadays put the dimension of an image file in the HTML. The original purpose for this is to help the browser to quickly format the page without retrieving all the images first. The following is an example of image tag:

```
<IMG SRC="http://www.xyz.com/foo.gif" WIDTH=600 HEIGHT=3>
```

One can easily infer from the dimension that this image must be a graphical page separator. Indeed, many buttons and ad banners tend to fall within a certain range of aspect ratio and dimensions.

USEMAP Attribute An `<IMG>` tag has an optional attribute that allows an image to be used as an image map. Depending on the location of a click within the image boundary, the user is directed to different URLs. An image map is usually used to implement a navigational menu, e.g.,

```
<IMG SRC="image/nav_top.gif" WIDTH=466 HEIGHT=24 BORDER="0" USEMAP="#nav_top">
```

Form Submission Buttons Another common tag besides IMG that contains an image is the INPUT tag. It can specify an image in place of the familiar grey rectangular pushdown button. An example of it would be:

```
<INPUT TYPE=image SRC="/ureg/generic/en/signin_button.gif" WIDTH=55 HEIGHT=23
 BORDER=0>
```

Therefore, those buttons used for submitting data for query or other purposes can be easily recognized.

Other Hints The alternative description text (i.e. the ALT attribute) can provide additional information about the image. In most cases, web page designers do not include ALT attribute if the image has little semantic importance. The occurrence pattern of an image is another useful clue that one can exploit. Unlike important phrases that tend to appear over and over again in an article, important images rarely repeat within the same page. Decorative images such as arrows and bullets, however, tend to appear at various places within the same page.

5.3.2 Implementation Details

The proxy is implemented in Perl5. We choose Perl because it has excellent support for pattern matching which is essential for parsing HTML pages. The main body of the proxy is an infinite loop, as shown in figure 4.

```
begin Open the proxy server socket to listen for incoming connections from clients.
  while (TRUE)
        Wait for ΔT seconds for incoming connection(s)
        if at least one incoming request is waiting
          then
                Accept the connection for the request
                Read the request URL
                Keep the connection open
          else
                if there is at least one accepted request waiting to be serviced
                  then
                        Service the request with the highest utility
                        if URL type is HTML
                          then Parse the HTML file and store URLs for all embedded images
                               Classify each image as one of the eight categories
                               Estimate the utility of the image and store it in table
                        fi
                  else Sleep until new requests come in.
                fi
        fi
  end
end
```

Figure 4: Pseudo-code for client-side proxy

The most critical step in computing the optimal download schedule is to estimate the utility of each image. This process involves parsing the HTML file, inferring the role of each image, and finally assigning a utility value to each image. In our prototype proxy, we used a 5-step process:

Step 1 Parse the HTML file using a publicly available HTML parser, called the HTML::Parser module, written in Perl [20].

Step 2 Filter out all tags that contains embedded image (i.e. `<IMG>` and `<INPUT>` tags).

Step 3 For each instance of <IMG> and <INPUT> we gather the following meta-information that describes an image:

Tag	Name of the tag, which is either IMG or INPUT
HTML URL	URL of the web page that contains this image
Image URL	URL of the image file
Link URL	URL of the link [1] (if the image is a link)
Width	Width of the image on the screen
Height	Height of the image on the screen
ALT text	Text description of an image embedded in the HTML

Step 4 Use heuristic to classify the image as one of the eight categories listed in Table 2. Each heuristic rule is a function that takes in the meta-information of an image and returns a category name. Each rule is invoked in a predefined order until one returns an answer or until all rules have been tried. If none of the rules matches, "other" is returned.

1. if TAG equals "INPUT", return "Button".
2. if USEMAP attribute is present, return "Menu".
3. if file extension of the link URL is "JPEG" or "GIF", return "Thumbnail".
4. if domain name of HTML URL $\neq$ domain name of link URL, return "Ad".
5. if link URL is longer than 160 characters, return "Ad".
6. if width ≤ 10 [2] or height ≤ 10 or aspect ratio [3] ≤ 0.1 or aspect ratio ≥ 10, return "Decoration".
7. if width ≤ 15 and height ≤ 15, return "Decoration".
8. if width ≤ 100 and height ≤ 40 and aspect ratio ≥ 4, return "Menu".
9. if file extension of the image URL is "JPEG" return "Photo".
10. if image area ≥ 30000 pixels and $0.5 \leq$ aspect ratio ≤ 4.0, then return "Photo".
11. if image has a link, return "General Link".
12. return "Other".

Step 5 Once an image is classified as belonging to one of the categories, it can be assigned a utility value. Currently, the utility of an image is a fixed value assigned to each different category. In our prototype, we ignore the size differences of images, and hence, the images are assigned download priority based on their utility values. This mapping from category to utility should take into consideration the preferences of the individual user. The mapping that we used for our experiments is as follows.

[1] The link URL is the one URL that the browser goes to when the user clicks on the image.
[2] All dimensions are measured in pixels.
[3] Aspect ratio equals width divided by height.

Category of Image	Utility
Menu	8
Thumbnail	7
Photo	6
Button	5
General Link	4
Other	3
Decoration	2
Ad	1

5.4 Experiments with Proxy

We ran a set of preliminary experiments to evaluate our prototype. The first experiment was to test if we can infer the role of an image in an HTML file. We ran our classification algorithm over pages from several popular web sites. The objective was less of trying to find a set of sophisticated heuristic that would classify images well. Instead, we wanted to see whether an HTML file contains enough hints that allow us to infer the role of an image before it is downloaded.

We first save a copy of the CNN, ABC News and Cornell University home pages as HTMLs. Second, we run a Perl script that implements the classification algorithm described in Section 5.3.2 to infer the role of each image. Next, we look at the images and decide which of the eight categories each of the images fits best. Admittedly, this step is highly subjective. Finally, we compare the classification given by the Perl script vs. what we give.

Table 3: Experimental results for the classification algorithm

Image Category	ABC News Cor / Tot	CNN Cor / Tot	Cornell Cor / Tot
Thumbnail	0/0	0/0	0/0
Menu	2/2	2/2	0/0
Decoration	11/13	6/6	1/1
Advertisements	2/2	7/9	2/2
Button	0/0	2/2	0/0
Photos	2/2	2/5	2/2
General Links	9/10	5/10	6/6
Other	2/2	0/0	4/7
Total (Percentage)	28/31 90.3%	24/34 70.6%	15/18 83.3%

Table 3 shows the number of correctly classified images, denoted by "Cor", out of the total number of images, denoted by "Tot". As seen from the results, the classification algorithm works quite well even with simple heuristic. Currently, we do not consider the location of an image in a page. We can certainly improve the heuristic by taking into account the locations of images. We feel that the HTML file contains enough information for us to distinguish images of different purposes with high accuracy. If the users prefers more fine-grained separation of image roles, new categories can be defined.

In the second experiment, we incorporated the classification algorithm into the client-side proxy. The goal is to see the effects of image re-ordering. To emulate a slow network link, we artificially limited the rate at which the proxy sends data to the browser to 28.8kbps. As expected, images were served in the descending order of their utility values. Advertisements were often correctly detected and transferred last. The limit of four simultaneous TCP connections did not pose a severe problem for our prototype because our prototype does not pipeline requests to the server. However, for better performance, we should allow at least two connections in flight to the server. In that case, four simultaneous connections will limit the freedom to re-order images, thereby hampering the degree of optimization. This problem can be easily solved by making minor changes to web browsers to allow more than four requests in flight.

6 Conclusion

The discussions in the paper lead to the conclusion that the linear utility function is among good choices of utility functions. Without precedence constraint, the optimal schedule is to sequence the n objects in the increasing order of p_i/α_i. Since p_i is not easily calculated due to the uncertainty in the transmission speed, we propose to sequences the objects in increasing order of s_i/α_i, where s_i is the size of object i. Assuming precedence constraint can be represented by series-parallel digraphs, we can use Lawler's algorithm [12] to find the optimal schedule, with the sizes s_i replacing the processing times p_i. Computational complexity for finding the optimal schedule is $O(n \log n)$. The scheduling algorithm leads to an optimal or near-optimal schedule that maximizes the expected sum of utilities for a wide class of random bandwidth variations, and it is very likely that in reality the bandwidth distributions belong to this class. The algorithm requires only one parameter α_i to be specified for each object.

We have demonstrated that we can implement UCO by inserting a client-side proxy between the client and the server. Best of all, we can affect the download order of objects without modifying the clients or the servers. Using the structure of the HTML file to infer meta-information allows the proxy to operate without the server's cooperation and yields satisfactory results. We believe that this will encourage users to try out UCO because one can reap the benefits of UCO without any assistance from the client or the server.

Experiences with the client-side proxy tell us that an architecture using a proxy can be very flexible. In fact, the current proposed UCO framework can be extended to including both a client-side proxy and a server-side proxy. A server-side proxy will be located on the same host as the server, but will be listening at a different, but well known port. It can serve several purposes. First, it can speak a transport protocol other than TCP or an application protocol different from HTTP to solve some well known performance problems of HTTP-over-TCP [21]. Second, the server-side proxy can transfer different versions of the same image based on the network condition, in a way similar to HTTP's content negotiation [19]. In fact, the server proxy can even re-encode an image using a layered codec or encode the image at a lower resolution to allow faster data transfer over slow wireless or modem links. This approach is similar to Gilbert's [7] except that he uses only one proxy, which corresponds to our server-side proxy. Gilbert's proxy is connected to the server over a high-speed, uncongested network, rather than living on the same machine, and it actually changes the HTML file seen by the client. Our client-side proxy is passive in the sense that the client still sees the same page.

Acknowledgement

The authors would like to thank David Starobinski for many helpful suggestions, and thank Venkat Anantharam, Steven McCanne, David Tse, Jean Walrand and Pravin Varaiya for their guidance.

References

[1] Rajarshi Gupta, "WebTP: A User-Centric Receiver-Driven Web Transport Protocol," M.S. thesis, University of California, Berkeley, 1998.

[2] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers," in *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, Colorado, October 1999.

[3] David D. Clark and David L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *Proceedings ACM SIGCOMM '90*, Philadelphia, September 1990.

[4] Kenneth R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons, 1974.

[5] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0*, February 1998, http://www.w3.org/TR/1998/REC-xml-19980210.

[6] World Wide Web Consortium, "http://www.w3c.org," .

[7] Jeffrey Gilbert and Robert W. Broderson, "Globally Progressive Interactive Web Delivery," in *Proceedings 1999 IEEE Infocom*, New York, March 1999, pp. 1291–1299.

[8] Richard M. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computation*, pp. 85–103. Plenum Press, 1972.

[9] E.L. Lawler and J. M. Moore, "A Functional Equation and Its Application to Resource allocation and Sequencing Problems," *Management Science*, pp. 77–85, 1969.

[10] Sartaj K. Sahni, "Algorithms for Scheduling Independent Tasks," *Journal of the Association for Computing Machinery, Vol. 23, No. 1*, pp. 116–127, 1976.

[11] Simon French, *Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop*, John Wiley & Sons, 1982.

[12] E.L. Lawler, "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints," *Annals of Discrete Mathematics, 2*, pp. 75–90, 1978.

[13] Michael Pinedo, *Scheduling - Theory, Algorithms and Systems*, Prentice-Hall, Inc., 1995.

[14] H. R. David, *Order Statistics*, John Wiley & Sons, second edition, 1981.

[15] Mark E. Crovella and Azer Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, vol. 5(6), pp. 835–846, December 1997.

[16] Stephen M. Stigler, "Linear Functions of Order Statistics with Smooth Weight Functions," *The Annals of Statistics*, vol. 2, no. 4, pp. 676–693, 1974.

[17] Richard Hyong-Jun La, "Webtp demo," http://www-path.eecs.berkeley.edu/~hyongla/webtp.html.

[18] T. Berners-Lee, R. Fielding, and H. Frystyk, *Hypertext Transfer Protocol – HTTP/1.0, RFC1945*, IETF, May 1996, ftp://ftp.isi.edu/in-notes/rfc1945.txt.

[19] R. Fielding et al., *Hypertext Transfer Protocol - HTTP/1.1, RFC2616*, IETF, June 1999, ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[20] Gisle Aas, *HTML::Parser–SGML Parser Class*, http://www.appwatch.com/Linux/App/1225/data.html.

[21] Venkata N. Padmanabhan, *Addressing the Challenges of Web Data Transport*, Ph.D. thesis, University of California, Berkeley, 1998.