Evaluation of *ESI* and *Class-Based Delta Encoding*

Mor Naaman, Hector Garcia-Molina, Andreas Paepcke
Stanford University
{mor, hector, paepcke}@cs.stanford.edu

ABSTRACT

The portion of web traffic attributed to dynamic web content is substantial and continues to grow as users expect more personalization and tailored information. Unfortunately, dynamic content is costly to generate. Moreover, traditional web caching schemes are not very effective for dynamically-created pages. In this paper we study two acceleration techniques for dynamic content. The first technique is *Edge-Side Includes* (ESI), and the second is *Class-Based Delta Encoding*. To evaluate these schemes, we present a model for the construction of dynamic web pages. We use simulation to explore how system, page and algorithm parameters affect the performance of dynamic-content delivery techniques, and we present a detailed comparison of ESI and delta encoding in two representative scenarios.

1. INTRODUCTION

A large number of web pages served today are dynamically generated based on the "profile" of the particular requestor, or on the characteristics of a particular request. For example, a user's page at Yahoo can contain stock prices from the user's portfolio, weather summaries for cities of interest to the user, and scores from selected sports events. Users love to get "personalized" content, and dynamic pages will clearly be a growing fraction of web traffic. However, dynamic content is expensive to generate and deliver, as page construction is resource intensive, and the pages are too dynamic or too personalized to be cached.

Thus, with a "naive" dynamic web delivery scheme, most requests for pages propagate to the server. In addition to all the assembly work and resulting higher latencies, network bandwidth consumption is high, causing this strategy not to scale well. A number of techniques have been proposed to accelerate the delivery of dynamic web pages. Some of them (e.g., [2,15]) are only concerned with reducing (or handling) the computational load on the server, without any influence on the network traffic load. However, our interest is in techniques that incorporate network savings as well, enabling caching of significant parts of the dynamic content. There are two classes of (sometimes orthogonal) techniques suggested in the literature. Work in [3,4,5,14] focused on deferred assembly of the dynamic page where the final page is assembled from cacheable page fragments. On the other hand, delta encoding focused on serving deltas between the page and other (cacheable) pages.

In this paper we focus on two concrete proposed systems, as representatives for each of the techniques. One system is based on ESI (Edge Side Includes), a scheme proposed by Akamai and Oracle for describing page assembly. ESI provides a way to fragment the page such that it can be assembled on *edge servers*, which in addition are able to cache the page fragments.

The second system we evaluate in this paper is class-based delta encoding. Delta encoding (DE) was investigated in a web context in [7,9,10] and has recently been described and implemented in [8,1]. In class-based DE [1], the server may encode the dynamic page as a "delta" from some "base file". The delta is sent to the client together with a reference to the base file. If the client does not have the appropriate base file, it asks for the base file from the server. Fortunately, the base files are similar to a static web page, and can be cached on the client side or on any network cache.

For a description of additional dynamic-page delivery systems and techniques, see our extended technical report [13].

In particular, the contributions of this paper are:

- A page-content model that captures the essential features of dynamic web page creation and assembly.
- First-published performance evaluation of ESI and DE delivery in some representative scenarios, highlighting the impact of key data and system parameters.
- A proposed variation for class-based DE. The variation, *same-base reply*, can significantly reduce client traffic as compared to traditional class-based DE, with only a small increase in server-side traffic.
- A detailed comparison of ESI and class-based DE. It should be noted that ESI and class-based DE are not functionally equivalent. For instance, our modeled ESI does not reduce client traffic, while DE often does. Yet, our comparison helps us understand the tradeoffs between solutions, and the impact of load, number of caches, link per-byte cost, and other factors on the system's performance.

To this end, we start in Section 2 by describing the ESI system. In Section 3 we describe the page-content model, and the physical system model is described in Section 4. Section 5 briefly describes the setup for the simulation, while Section 6 presents the simulation results and an analytical discussion of the results for ESI. In Section 7 we describe class-based DE. Simulation results for DE follow in Section 8, together with a description of our same-base reply enhancement. Section 9 compares ESI and DE systems as informed by our simulation results, and we conclude in Section 10.

2. ESI

The first acceleration scheme we consider is Edge Side Includes as described in [4]. ESI is a specification, suggested by an industry panel headed by Akamai and Oracle, for an XML-based language that is used to enable assembly of a dynamic web page from smaller fragments. The fragments can be independently delivered and cached closer to the client. Instead of generating a full HTML page, the server generates ESI code fragments, each containing the original HTML code for this fragment with additional ESI directives. Incidentally, ESI requires a complete revision of the website's code, as the code has to be re-written to use ESI language directives. A sample website and fragmentation of it are shown in Figure 1.

The fragments are cached on specialized *edge servers*. The edge servers are also where each page is assembled, based on the ESI directions in the fragments. The edge servers are assumed to be much closer to the client than the main server, thus not only saving the assembly cost on the main server, but also reducing the overall network and main server's bandwidth requirement.

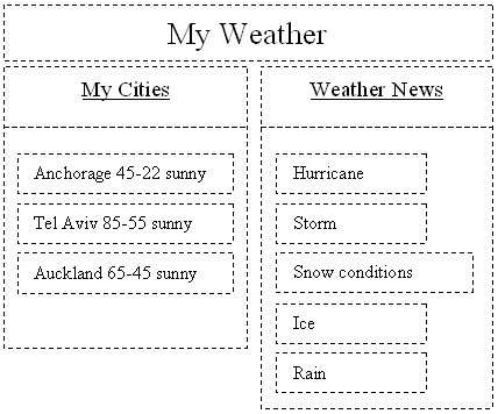


Figure 1 - MyWeather Page

As in [14], we differentiate between the ESI language and ESI-based systems. For the rest of this paper, "ESI" stands for our modeled ESI system, unless apparent from context we are referring to the language.

There are a number of possible architectures for an ESI system, differentiated by the number and position of the edge servers. Our modeled standard case involves thousands of edge servers spread across many different geographic locations[1], as shown in Figure 2. The exact number and location of those servers affects the performance in ways we discuss later. However, usually there is a need for *backend servers*, located closer to the main server, to compensate for the expected low hit rate at the edge servers (as the smaller client population for the edge servers entails poorer locality). The backend servers are part of the system we simulate in this paper, and the system architecture is detailed in Section 4.

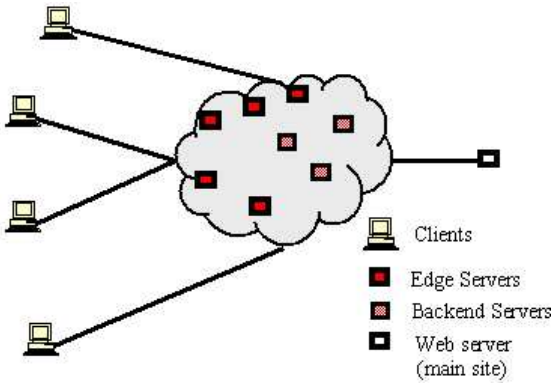


Figure 2 - ESI Architecture

3. A PAGE CONTENT MODEL

Evaluation of page caching schemes typically requires a page access model. In our case, we need a much richer model, in order to represent what fragments are accessed by clients *and* how they are to be assembled into full pages. Our model strikes a balance between simplicity and the richness necessary to capture the main tradeoffs. The model consists of three parts: a model for the underlying data of a resource, a model for the construction of a dynamic page, and a model for the physical system (presented in the next section).

Up to now we have used the term "resource" loosely; we now define it more precisely. A resource is a homogeneous pool of web pages represented by a single URL, or by a collection of URLs. For example, our MyWeather and My Yahoo are resources represented by a single URL; the group of Amazon book pages is represented by a collection of URLs[2]. The term *page* is used as an instance of a resource – a specific book's page on Amazon; a client's actual MyWeather page, etc.

A dynamic page is constructed by a selection from available items. These *data items* are usually chosen from different *groups*. For example, the data for MyWeather is divided into two groups: the Weather News group and the MyCities group. The weather news group contains a number of different news items. The MyCities group contains the data items for all the cities available for display. In another example, a simplified model of the resource "Amazon books" has one group (a books group). The data items in this group correspond to all the books in the Amazon catalog.

Each group also has a static part, like the header of the MyCities fragment for example. In our model this part can be personalized or generic; i.e., can be shared by other users (e.g., the static part of the weather news group) or applies to only one user (e.g., the MyCities static part).

[1] Two special cases that we do not discuss are placing the servers on the main site, and on the other hand, pushing the assembly all the way to the clients ([14]).

[2] http://my.yahoo.com, http://www.amazon.com

To model the creation of an instance of a dynamic page (e.g., some user's MyWeather page) we simulate a two-phase selection. The first phase is a selection of groups to appear on the page. In the second phase we simulate a selection of data items from each group. Below are two examples for the selection process.

To generate an instance of MyWeather page, first both available groups are selected (news and MyCities appear on every client's page). Subsequently, the data items from each of the two groups that appear on a specific user's MyWeather page are selected. Selecting the news items for a page is simple, since in this case all the users get the same news items. The selection of items from the MyCities group, however, is based on user preferences; for user A the selection includes the Anchorage, Tel Aviv and Auckland data items.

A construction of an Amazon book page is slightly different in practice, but is still accommodated by our model. Unlike the previous example where the page is constructed based on the user's pre-defined preferences (that correspond in part to the selection of groups and items), here the user "creates" his page implicitly, by choosing a specific book's URL. However, we can still model this process as a selection from the "books" group – there are many possible data items to be picked, and one of them is selected to appear on the simulated page.

We model the selection of groups and items using a Zipf-like distribution [17], where the likelihood that an item is picked is proportional to $1/i^{\alpha}$, and i is the item rank based on popularity (most popular item is first, i=1). The popularity curve is steeper for larger values of α. Many studies ([6,18,19,20,21]) suggest that web page requests follow Zipf-like distribution, although those studies do not agree on a single value for α. We extend this hypothesis to items and groups, and use a range of values for α to ascertain its effect on the model.

The parameters for the model are discussed in greater detail in [13], and are summarized here in Table 1.

4. THE SYSTEM MODEL

The physical system described in this paper consists of:

- Clients that make requests for the resource, and receive page instances in response.
- Thousands of edge servers at the "edge" of the network, i.e. a few hops away from the clients. The number is chosen to resemble current CDNs, and is varied in our simulation.
- A few "backend" servers or caches located "closer" to the main site. The backend servers can fulfill requests from the edge servers, supplying another layer of caching between the edge servers (that handle a smaller number of clients and thus show lesser locality) and the main server. The better locality of the backend servers helps to further reduce load from the main site's servers. For simplicity, we assume a tree structure of the physical system: a client always accesses the same edge cache; an edge cache always accesses the same backend cache.
- A web server at the main site.

Table 1 - Parameters of Page Model for Resource RS

Parameters	Short description
Global	(Global parameters)
$\mathbf{N_{RS}}$	Number of available groups
$\mathbf{g_{RS}}$	Distribution function describing the number of groups per page
$\mathbf{f_{RS}}$	Distribution function describing the relative popularity of the available groups
Group	(Parameters for a specific group j)
$\mathbf{k_{RS,j}}$	Number of items in the group
$\mathbf{g_{RS,j}}$	Distribution function describing the number of items picked for a page from group j
$\mathbf{f_{RS,j}}$	Distribution function describing the relative popularity of items from group j
$\mathbf{ItemSize_{RS,j}}$	Size in bytes of HTML/ESI code for items in group j
$\mathbf{StaticSize_{RS,j}}$	Size in bytes of HTML/ESI code for group j without any items
$\mathbf{TTL_{RS,j}}$	Time-to-live in seconds for items in the group

Table 2 - Architecture Model Parameters and Default Values

Parameter	Default Value
Number of distinct clients	1,000,000
Page requests per day	8,000,000
Number of edge servers	4000
Number of backend servers	4

For this paper, bandwidth usage is the principal metric. We evaluate the total traffic on the different links: the total traffic (in bytes) sent by the server per day, as well as the total aggregate traffic per day received by all clients, and total traffic per day between the edge and backend servers. Metrics such as user-perceived latency and server CPU loads are, of course, closely related to the number of bytes transferred on each link, but are not explicitly evaluated in this paper. Instead, we use a simple weighted-cost model of the traffic on each link when we compare the two systems, and discuss the other measure qualitatively in Section 10.

Factors such as CPU constraints, disk size and memory constraints on the edge, backend and main site servers are not modeled. All the caches modeled in our system are presumed to be of infinite size. We will present cache usage information for some of the simulations, to confirm that usage is modest.

The basic parameter values in modeling the architecture are detailed in Table 2.

5. SIMULATION

We use our model to simulate difference resources (or web applications). We consider one resource at a time, and show that the performance for different resources will be

considerably diverse, due to the dependency on page model parameters. We have chosen to perform a separate evaluation for a number of specific, yet representative, resources as opposed to evaluating a "mean" behavior for a large number of mixed resources. This strategy lets us better understand why each scheme performs as it does. The representative resources we consider are:

- An online bookstore's book-page resource, similar, for example, to a book page on Amazon. This resource is an example of a resource with comparatively large, high TTL fragments and relatively little personalized content.
- A stock portfolio page, representing a resource with a large static portion and a personalized group selecting many small data-item fragments (many stocks on each page).
- My Page, a My Yahoo-style resource that combines many highly personalized as well as generic groups, and also both low and high TTL groups and items.
- Our hypothetical resource, MyWeather, represents a simple dynamic page, with a relatively large static portion, and two statistically different groups – the news, which is the same for all clients, and a personalized MyCities fragment.

Our software simulates the client requests and the way requests are processed at the different caches levels and at the main server. For each request, a page skeleton is generated using the given resource's parameters, listing all groups and items that appear on the page. This skeleton is used to generate an item request pattern at the simulated caches. The client request arrival rate follows a Poisson distribution. The simulated time period in each case was 24 hours, with a prior 10-hour "dry run" before collection of statistics starts to allow the system to reach a steady state.

6. ESI SIMULATION RESULTS

As mentioned above, the main metric we examine is the amount of traffic on the different links. This metric is obviously directly related to the hit rate for page fragments on the edge servers. In our model there are two types of fragments: fragments that represent data items and fragments that represent static parts of the page. The latter (unless personalized) appear on most pages, and thus are almost always cached. For the following discussion, *hit rate* is therefore defined for data items only.

Before showing our simulation results, we derive two equations that yield useful insights, even though they are based on some simplifications. The equations are based on the work of Breslau et al in [20]. Assuming that item requests are independent, and that all items are homogeneous in size and time-to-live parameters, we show in [13] that when $1 << \lambda \cdot TTL << N$ the hit rate in each cache is expected to follow $HR_1 = \sum_{i=1}^{N}(1-(1-p(i))^{\lambda \cdot TTL}) \cdot p(i)$ (**Equation 1**), where $p(i)$ is the probability for item i to appear on any page (i.e., the item's popularity); λ is the **item** request arrival rate for a single cache, TTL is the time-to-live value for items and N is the number of items available to choose from. We also show that Eq. 1 can be approximated in closed form using $HR_2 = \Omega \ln(\lambda \cdot TTL \cdot \Omega)$ (for $\alpha=1$) or $HR_2 = \frac{\Omega}{1-\alpha}(\lambda \cdot TTL \cdot \Omega)^{\frac{1}{\alpha}-1}$ (for $0<\alpha<1$) (**Equation 2**), where $\Omega = (\sum_{i=1}^{N}\frac{1}{i^{\alpha}})^{-1}$ is the Zipfian distribution constant.

This analysis suggests we can expect a logarithmic increase in a cache's hit-rate with an increase of the arrival rate λ or the TTL. This assumption is not always true for the edge servers because of the lower arrival rate λ ($1 << \lambda \cdot TTL$ does not hold). In fact, we witnessed in the simulation that the item hit-rate at the edge caches grows linearly with λ and with the TTL value when their product is small. However, we show below that the equations are good approximations for the *overall system hit-rate*: the percentage of all item requests satisfied by some cache from both levels (edge and backend), calculated as *(number of hits at backend caches + number of hits at edge caches) / (number of total item requests).* If we look at the entire cache system as a single cache, then Eq. 1 and 2 are estimations for the overall system hit-rate.

6.1 Simulation Results – Online Bookstore's Book Pages

A "Bookstore book" resource can describe the collection of all individual product pages on a web site, for example, music-album pages on allmusic.com[3], book pages on Amazon, or similar resources. See Table 3 for a summary of the main parameter values. This resource has two groups that appear on each page instance. The first group's static part models the page header, static links etc. The items in

Table 3 - Default parameter values bookstore resource

Number of groups	2
Group 1	Appears on every page Static portion size: 10Kb Item size: 17Kb Items per page: 1 Item pool size: 400000 Item time-to-live: 1 hour α = 0.8 (see graphs where α=1.1)
Group 2	Appears on every page Static size: 3Kb Personalized and un-cacheable

[3] The model parameters of Table 3 are loosely based on this resource, found at http://www.allmusic.com

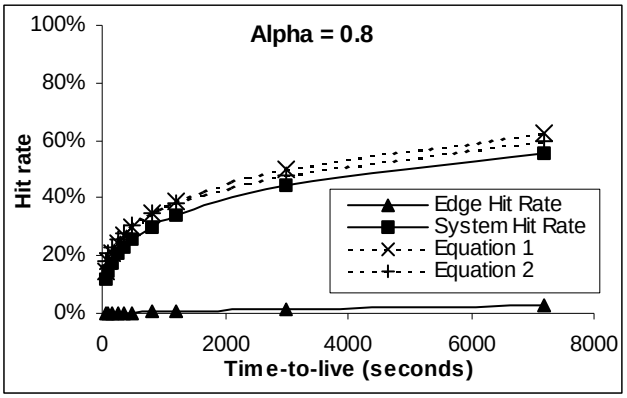


Figure 3 - Edge and backend servers' item hit-rate vs. item TTL, bookstore-style resource (simulation results and estimations)

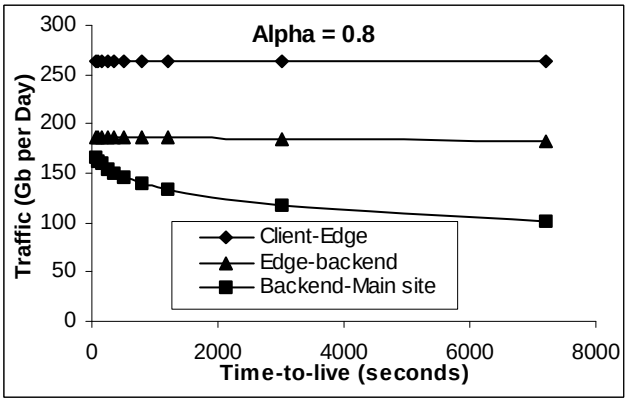
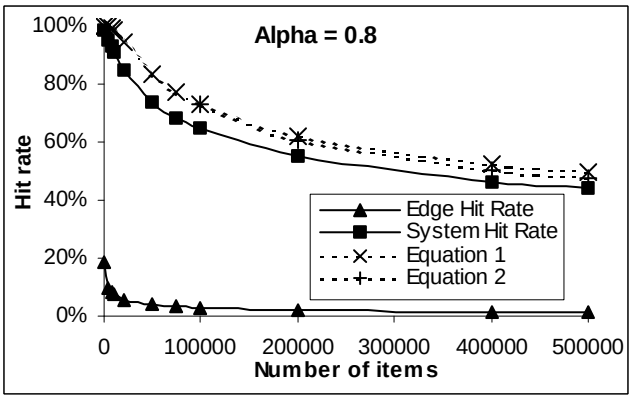


Figure 4 - Traffic vs. time-to-live, bookstore-style resource



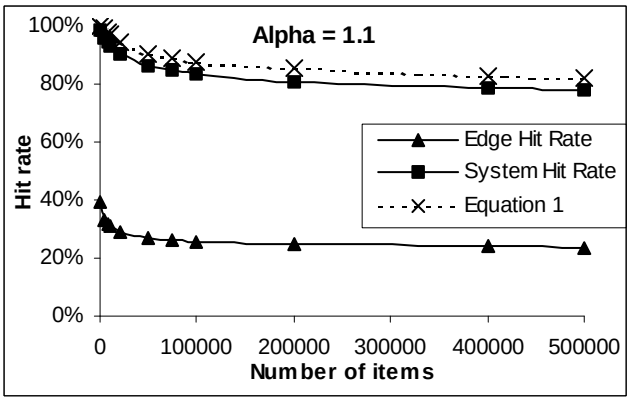


Figure 5 - Edge and backend servers' data-items hit-rate vs. number of available items, bookstore-style resource (simulation results and estimations

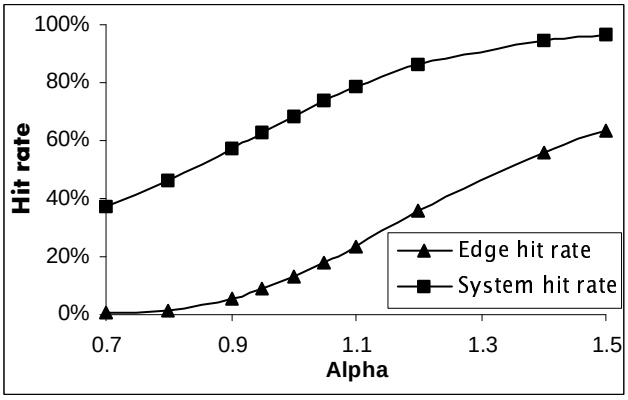


Figure 6 - Hit-rate vs. value of α, bookstore-style resource

this group are the items (e.g., books) available for this resource; the selection in the group is for one item per page out of the collection of available items. The second group on the page includes some personalized user information, such as a user name, specialized recommendation, shopping cart etc. Notice that for this resource, all data items belong to a single group; therefore, equations 1 and 2 are directly applicable.

Figure 3 shows the hit-rate for the data-item fragments (i.e., the book fragments) vs. the items' TTL value. We show the system hit rate and the edge-cache hit rate for a single α value (the affect of α will be shown in Figures 5 and 6) as the TTL ranges from 60 seconds to 7200 seconds (two hours). In addition to the simulation results, Figure 3 shows the estimates for the system hit-rate using Eq. 1 and Eq. 2. We can see that both equations are relatively accurate, demonstrating the logarithmic nature of the TTL effect on the hit rate.

According to Figure 4, that shows the total traffic on each link as a function of the TTL, the edge servers are useful in caching the static fragments of the page despite the low item hit-rate, witnessed by the savings for the edge-backend link. On the client-edge link there are no savings since in our modeled system, clients receive fully-assembled pages.

Figure 5 illustrates the effect of the number of data items available for selection, N, on the hit-rates. The Figure shows the curves for edge hit-rate, system hit-rate, and the approximations for system hit-rate using Eq. 1 and Eq. 2 (the latter is missing from figure 5b since Eq. 2 is not valid for $\alpha>1$). Here we show results for two different α values. The low end of the item count, 1000 items, is representative for, e.g., a newspaper-articles resource. The higher end (500,000 items) is characteristic for product pages on a medium-scale online store. We can see from Figure 5 that ESI performance improves for a lower number of items; however, for larger values of α this effect is not as critical.

Figure 6 summarizes the effects of the slope of the Zipf-like curve (determined by the value of α) on the edge servers

and system item hit-rates. The value of α ranges from 0.7 to 1.5 - values that were witnessed in numerous web-caching studies ([19], [20], [21] and others). Especially noteworthy is the magnitude of change in edge hit-rates, from close to 0% to about 60%. To give an idea, for 400,000 items, when α=0.8 the first three items account for 3.23% of all requests; when α=1.1, the first three items account for 23.12% of the requests.

For the basic settings of the bookstore resource, the simulation resulted in traffic savings of 30% for the edge-backend link, and 56% for the backend-main site link. The static parts of the page are the main contributor to the savings on the edge-backend link, as they are frequently served from the edge caches. The backend servers do a better job than edge servers in serving the data items, accounting for most of the additional 26% savings. Finally, the edge cache usage for this resource was around 1.2Mb per cache, consisting mostly of data items.

In section 9 we present more variations on the model parameters, including experiments with different request arrival rates, which are analogous to varying the number of caches in the system.

6.2 Simulation Results – Other Resources

We evaluated three resources in addition to the Bookstore resource. Due to space limitations, we only discuss here the key differences from the Bookstore resource (an extended discussion is found in our technical report [13]). For these new resources, a given user always sees a fixed set of groups and data items. For example, when users ask for their portfolio, they see the same stocks every time. Of course, the stock values may still change between requests. The first resource is indeed a brokerage portfolio resource (e.g., users' personal financial portfolio web page), described in Table 4. The second resource is the above-mentioned MyWeather. Table 5 lists the basic parameters for this resource. The third is the most complicated among our resources, the personalized My Page, a simplified model of a MyYahoo-style resource. My Page parameters are summarized in Table 6.

Table 7 lists the simulation results, as average savings and edge server cache size requirements (assuming evacuation of stale fragments only), for all the resources we simulated. The cumulative traffic on the edge-backend link and backend-main server link is measured in the simulation, and the percentages in Table 7 represent the reduction in traffic on these two links from the total traffic between server and client without ESI (the client link has no savings in ESI; it is not listed in the table).

Some comparative observations about Table 7 follow. The portfolio resource exhibits greater savings for both types of links compared to the bookstore resource, suggesting better caching at the edge servers. Since the relative size of the static portion of the two resources is similar, the improvement stems from better caching of the data items, in this case the stock fragments. The reason for that is twofold; first, there are fewer data items to choose from in the group; second, every page request translates into many data-item requests since a page has 25 stocks on it on average. The reduction is despite the lower TTL for the data items in the portfolio resource (5 minutes compared to 1 hour). The low TTL does affect the hit-rate at the edge; but the backend servers, having a much larger request arrival rate, are more effective in caching data items. The cache size usage for the portfolio resource is very low, and is in fact dominated by the cost of caching the personalized static part of the page.

For MyWeather we see even more savings than the portfolio resource. Compared to the portfolio resource, MyWeather has, in both groups, a smaller number of data-

Table 4 - Default parameter values, portfolio resource

Number of Groups	2
Group 1 – stock entries	Appears on every page Static size: 2Kb (personalized) Item size: 1Kb Items per page: 1-50 Item pool size: 10000 Item time-to-live: 5 minutes α = 0.8
Group 2 - header	Appears on every page Static size: 18Kb No items (static part only)

Table 5 - Default parameter values, MyWeather resource

Number of groups	2, both appear on every user's page
Group 1 – MyCities	Static portion size: 3Kb Item size: 2Kb Items per page: 1-10 Item pool size: 5000 Item time-to-live: 1 hour α = 0.8
Group 2 – Weather News	Static portion size: 30Kb (incl. page header and footer) Item size: 3Kb Items per page: 5 Item pool size: 5 Item time-to-live: 5 minutes

Table 6 - Default parameter values, My Page resource

Groups	14 groups Number of groups of page: 4-14 Group selection zipfian parameter α=0.7 Average page size: 42Kb
Group names	News modules, movies, email notification, personal stocks, yellowpages module, personalized weather…

Table 7 - Result summary

	Savings edge-backend	**Savings backend-main site**	**Edge cache usage**
Bookstore	30%	56%	1.2Mb
Portfolio	48%	95%	0.5Mb
MyWeather	84%	98%	1.3Mb
My Page	62%	75%	1.5Mb

items with higher TTL. Especially, the data items in the 'news' group are few, and the selection from them is much more restricted. The result is the resource being very well cached in both edge and backend caches, as can be seen in Table 7. As a result of a larger TTL and a larger size of the personalized static part, the cache usage is higher than for the portfolio page. The cache contents in this case are split more evenly between data item fragments and personalized fragments.

My Page bandwidth savings from Table 7, although not as significant as the savings for the portfolio or MyWeather resources, are still better than the bookstore resource's savings, despite the large amount of personalized information. Notice that the savings on the edge-backend link are greater than for the portfolio resource, but it is the opposite case for the backend-main site savings. This difference is caused by the larger relative portion of personalized information for the My Page resource. While data items can be cached better using cooperative caching (i.e., backend servers)[12], cooperative caching does not add any benefits for the personalized parts of the page. The cache usage for My Page is the largest, mainly due to the many personalized static fragments, but also due to a large number of data items cached.

7. CLASS-BASED DELTA ENCODING

Class-based delta encoding (DE) exploits the fact that two different versions of the same dynamic page will often bear great similarity. A "Condenser" is a machine (or a number of machines) located between the organization's web server

and the Internet. The Condenser serves as a proxy for client access to the web server, forwarding the requests from clients to the web server, and applying DE to responses from the web server to the clients. Class-based DE allows the Condenser to exploit spatial locality, in this case similarity between different pages from the same resource viewed by different clients.

When the web server receives a request, it produces the complete page, and delivers it to the Condenser. The Condenser holds a number of candidate base files that can simply be described as previously generated pages. The Condenser chooses one of these base files, encodes the current page as a delta from the chosen file, and sends the delta to the client along with the base file's identifier. The details concerning how base files are selected for use in servicing a particular request appear in [1].

The main benefit of class-based DE is the reduction in traffic, since the base files are static resources and can be cached on traditional network caches (and on end-users' machines). In many cases, caching will reduce the traffic out of the Condenser to deltas, which are usually very small compared to the original page size. Notice that DE does not reduce the computational load on the web server, since every request propagates all the way to the server, and the server generates the entire page for each request.

7.1 System Model

The physical system model for DE simulation is very similar to the model for ESI described in Section 4. The only difference is the omission of the backend servers from the system model. Class-based DE can utilize regular proxy caches, installed by independent bodies such as ISPs and are usually found at the "edge" of the Internet, and not at the backend. In summary, the DE system includes clients, proxy caches (instead of "edge caches" as they appear in the ESI model), and a main site with a web server and a Condenser.

Some new parameters are needed for the simulation of class based DE, like the number of possible base-files generated by the Condenser. We do not have an estimate for this number, which is dependent on the clustering algorithm and the resource. Therefore, instead of predicting the number of base files created, we used a range of values, studying the effect on performance. In addition, we used a time-to-live parameter for base files, because the algorithm can replace them over time.

Another parameter is p_{same}, the probability of a client getting a delta from a base file it already has cached ("same" base file). The value of p_{same} is determined by the resource type. For example, for a resource like My Page, the client essentially accesses the same page every time. It is likely for the Condenser to use the same base file for the reply, unless this base file has timed out. Therefore, in this case, p_{same} is close to 1. On the other hand, consider a resource like the online bookstore book pages. Each request a client makes for a page will be subject to a process of finding the new best base file by the Condenser. The Condenser is not likely to pick a base file the client has already cached, so p_{same} is close to 0.

8. DELTA ENCODING SIMULATION

To simulate DE, we first generated pages with content, as opposed to the page skeletons we used so far. Recall that the page content model (Section 3) describes the components of the page, but not their textual content. Two methods were used for creating this content:

1. Download actual pages from the web that fit the model. This is the option taken for the bookstore-style resource.
2. If the model is not based on an existing resource, like My Page, we generate page instances. First, we manually generate templates for all the groups and items of the resource. Each template provides the HTML text that is constant for each group, and placeholders for fields that vary between instances. For example, the template for a Weather item contains placeholders for the city name and temperatures. Each template is generated to match the model parameters such as size. We then use these templates to generate item instances, replacing the placeholders with text, words or numbers generated by a random text generator. To produce full pages, we run our simulation to output page skeletons

(containing identifiers for the constituent groups and items), which are then populated with group and item instances.

We used the following procedure to estimate the expected size of deltas d_{base}. We generated 2000 instances of each resource to serve as content pages, and we selected a subset S ($|S| = 20$) of pages to represent the Condenser's base files. Then, for each page P in the initial 2000 pages, we applied the *vcdiff* algorithm to compute the delta between P and each S page (*vcdiff* [16] is based on the *vcdelta* [11] algorithm studied comparatively in web-context in [7], and used in [8, 1]). For each page P we picked the minimum delta size between P and any page from S. The average over all minimum values for pages P is the value d_{base} we used for the simulation.

In the rest of this section we present simulation results for the on-line bookstore resource, and touch on the performance difference between it and My Page resource. The significant difference shows the effect of a high p_{same} value versus low p_{same} value, especially on the client traffic. We later propose a variation on the algorithm for resources with a low p_{same}, and evaluate its potential benefits.

8.1 Simulation Results – Online Bookstore

The online bookstore resource represents a collection of URLs, in this case, all the books in the store's database. A user can access a different page (i.e., a URL of a different book) every time. For this class of resources there is no guarantee for the user to get a delta from the same base file on subsequent requests ($p_{same} \approx 0$). As a result, the traffic to clients is usually greater than the traffic without class-based DE, since the base file and delta are both retrieved for a total size greater than the original page size. On the other hand, traffic from the main site is still reduced since the base files are requested more often and as a result are better cached on proxy caches. The size of delta between pages used in the simulation was d_{base}= 1922 bytes, generated as described above using real pages downloaded from allmusic.com.

Figure 7 shows the aggregate client traffic and total main site traffic as a function of the number of base files. The figure also shows the original volume of traffic without class-based DE. We can see that the client-side traffic almost always exceeds the original traffic but there are considerable savings on the server side. The hit-rate for base files goes down from almost 100% (10 base files) to around 52% (1000 base files), and the effect on traffic is seen in the figure.

In summary, with resources of this type, where a client is likely to download a different base file on every access, class-based DE usually reduces only main-site traffic and not client traffic. In that, the reduction is highly dependent upon the hit rates for base files on the proxy caches.

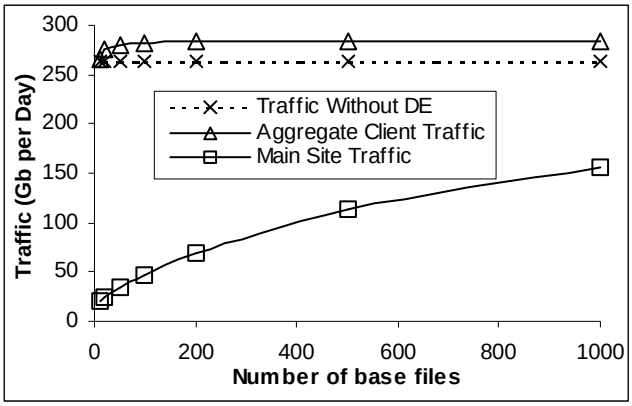


Figure 7 - Online bookstore, class-based DE traffic vs. number of base files

8.2 Simulation Results – My Page

For this type of resource we assume a client is likely with some high probability p_{same} to get the same base file on subsequent accesses, since the page viewed is not fundamentally different from one access to the other. We used d_{base}=1050 bytes, generated as described above, as the delta size for the simulation.

The high p_{same} generates much larger client-side traffic savings. DE saves 66% of aggregate client traffic under the basic settings for this resource, while the Bookstore resource offers no savings for clients at all. On the server side, the reduction in traffic amounts to 87%. The extended results of this simulation are found in our technical report [13]. Some variations on the basic settings are done in section 9.1.

8.3 Same-Base reply for Online Bookstore

In this section we introduce a simple modification that may improve class-based DE performance for the online bookstore resource and other resources where the clients with high probability receive a different base file on every access. We propose simply to set a threshold for a "*same-base reply*": The client request includes information, possibly in a cookie, about the most recent base file in its cache. The DE Condenser first computes the delta of the new client page from the base file in the client's cache. If the delta is lower than a certain pre-defined threshold, the Condenser sends the computed delta to the client, which in turn applies it to the base file in its cache. In this case, the process of classifying the new page and finding a base file candidate on the Condenser is avoided. Note that we have the client send only the identifier of the most recent base file because identifying the entire set of its cached base files would require multiple delta computations on the Condenser.

This scheme is of course sensitive to the distribution of the delta sizes between pages. We used the set of 2000 downloaded pages to compute ds_{all}, the distribution of delta sizes for any pair of pages. The results are displayed in

Figure 8. We used the ds_{all} distribution to compute the probability that a delta will be lower than the threshold, and the average size of those deltas. For the case where the threshold was exceeded, we plugged in the average delta size of the regular scheme, d_{base}.

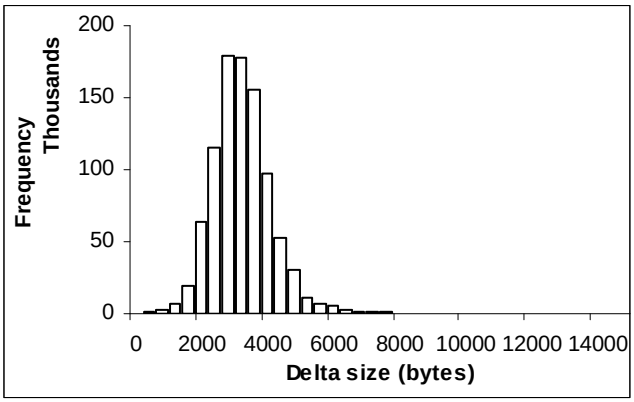


Figure 8 - ds_{all} distribution for online bookstore resource

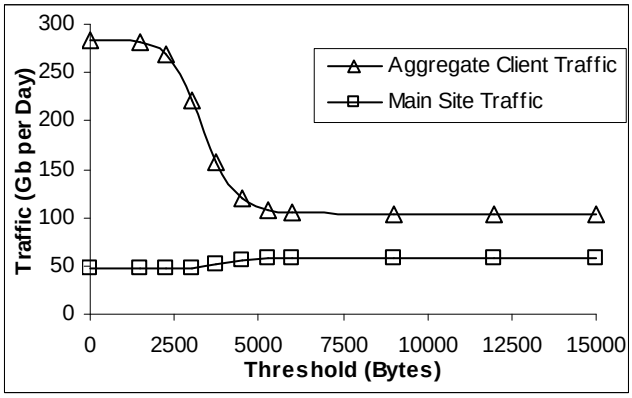


Figure 9 - Traffic vs. threshold for Same-base Reply

Figure 9 shows the aggregate traffic to clients, and total traffic at the main site, as a function of the threshold set for the algorithm (the number of base files for this simulation is 100). For comparison note that the scheme is equivalent to the regular DE scheme when the threshold is set to 0. We can see this scheme reduces the aggregate client traffic since many times we replace downloading a new base file and a small delta, with downloading a slightly bigger delta, but without a base file. At the same time, traffic from the main site increases, mainly because the main site is supplying larger deltas in a lot of cases. The increase is less due to the reduction in the number of base-file requests (since caching was already keeping this traffic to minimum).

The breakdown of client traffic is a little different with same-base reply. The clients now receive greater deltas (transferred from the main site) instead of smaller deltas and base files (the latter are transferred, hopefully, from a closer proxy cache). Thus, even though the aggregate client traffic is lower, the reduction in latency may not be as low – fewer bytes are transferred, but they travel a longer way on average. A more careful study of the system's latency factors would be needed to show that same-base is indeed superior to the regular scheme.

9. COMPARISON OF ESI AND CLASS-BASED DE

This section presents a quantitative and qualitative discussion of ESI and class-based DE. The quantitative discussion is not straightforward, since the performance of the two techniques is dependent on parameters that may not be equivalent in each system. Furthermore, physical models for the two systems are somewhat different, with a cascaded architecture for ESI, and only one level of caching for DE. Thus, in some respects an ESI-DE comparison is like comparing "apples" with "oranges." Nevertheless, we believe a comparison is useful because both solutions have the same ultimate goal: the reduction of traffic, and caching of content useful for dynamic pages. We overcome the physical system difference by eliminating the backend caches for ESI from the model; we will mention the additional savings using backend servers when relevant.

The main metric for evaluation continues to be network traffic. We present traffic measures for the different links (clients/edge servers, edge servers/main site, and also clients/main site). In addition, we offer a simple, weighted cost formula that combines traffic measures on the different links. The weights account for varying costs associated with the links. Finally, we also show the cache space usage for both schemes.

9.1 My Page Resource

Table 8 sums the reduction in traffic and cache usage for the My Page resource with ESI and class-based DE. As we mentioned in Section 8, in DE with this resource a client is likely to get the same base file on every access. Two points are worth repeating: First, our modeled ESI system does not introduce savings on the client link since the page is assembled as a whole on the edge servers. Second, there are further savings to ESI, when using backend caches, which are not part of the system model for this part (see Table 5 for the complete ESI results).

Table 8 - My Page comparison

	Savings client link	**Savings server link**	**Edge cache usage**
ESI	0%	62%	1.5Mb
DE	66%	87%	3.2Mb

Figure 10 shows the variation in traffic on the client and main site links, for ESI and DE, as a function of the request arrival rate. The measure we use is bytes per request (total traffic in a day divided by the total number of requests) to better show the impact of a higher load (where overall traffic grows but traffic-per-request may drop). In both cases the traffic per request on the main site drops with the arrival rate due to better caching, but for DE the **client** traffic (per request) is reduced too. Since clients cache base

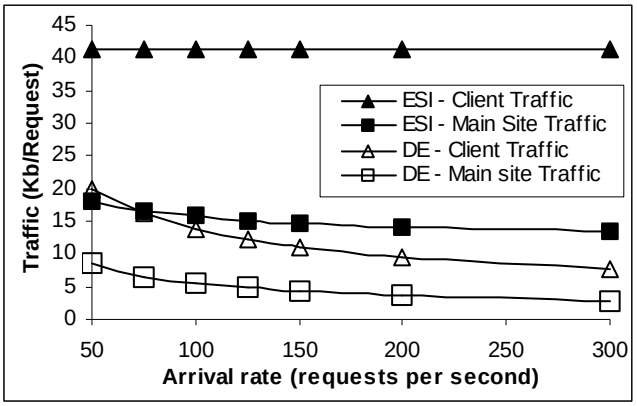


Figure 10 - My Page resource, traffic for ESI and class-based DE vs. request arrival rate

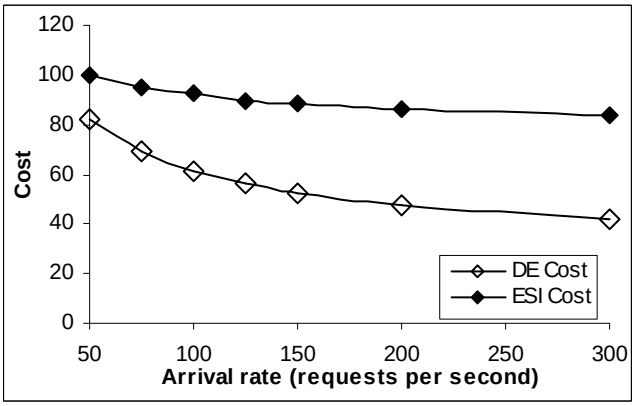


Figure 11 - My Page resource, cost of ESI and class-based DE vs. request arrival rate

files, and are likely to use the same base file on every access, the benefits of caching grow with the request rate. In conclusion, higher request rate entails better caching for both systems on the edge caches, but also means more benefits for client base-file caching in DE.

Notice that with DE, clients receive data from the main site "directly" (deltas), as well as base files from the proxy cache. The DE client curve of Figure 10 shows this aggregate traffic. On the other hand, with ESI, clients only receive data from the edge server, i.e., ESI client traffic in Figure 10 is exclusively edge-server-to-client traffic (of course, the edge server may get required data from main server when necessary). The client to main server traffic occurring with DE may be less desirable because it has to travel a longer distance (incurring more latency) and because it generates load (delta computation) at the server.

To take into account this and other differences, we propose a traffic cost model where each type of traffic is given a different weight. In particular, we use the cost function Cost = C_1×client-server traffic + C_2×cache-server traffic + client-cache traffic. Factor C_2 represents the cost ratio between cache-server and client-cache traffic. A high value for C_2 means that server traffic is more costly or less desirable than client traffic. Factor C_1 is only valid for DE since in ESI there is no "server-client" traffic. A high value of C_1 indicates that server-client traffic is more costly than other server traffic, because it incurs computational costs or because of larger latencies.

To illustrate, Figure 11 compares the ESI and DE costs for a scenario where C_1=25 and C_2=5. We chose a relatively large C_1 value to reflect what we think are the significantly higher costs of computation and latency of server-to-client traffic. With a C_2 value of 5, one byte of cache-server traffic is as costly as 5 bytes of client-cache traffic, perhaps because the cache is much closer to the client (this may not always be the case; these numbers are just illustrative). For this scenario, Figure 11 allows us to compare the desirability of each scheme. In this case we see that DE is preferable (lower overall cost) and the gap grows (DE even more desirable) as the arrival rate increases.

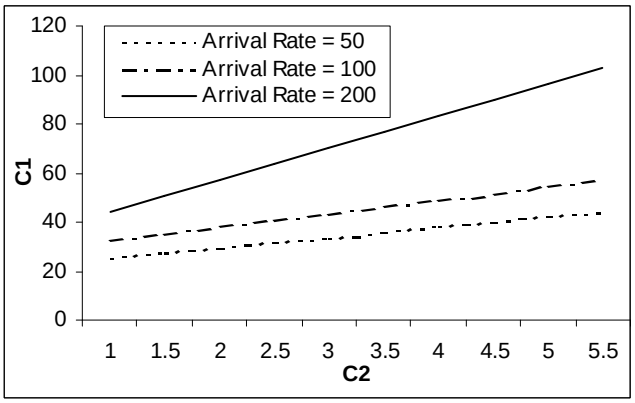


Figure 12 - My Page, values for C1, C2 where DE cost is equal to ESI cost

Figure 12 shows when DE is superior to ESI, as a function of the C_1 and C_2 cost factors, for the different arrival rates. The area under each line is where DE cost is *lower* than ESI cost for this arrival rate. For example, when C_2=3, DE is the better scheme as long as C_1 is less than or equal to 44 (for arrival rate = 1). Notice the high value of C_1 needed to equalize the cost for a given C_2. In other words, when the cost of DE (generating a full page, computing a delta, transmitting to the client and applying the delta) are dominant, the schemes are comparable.

A variation in the number of users-per-cache does not have a considerable influence on performance for this resource, neither for ESI nor for class-based DE, and due to space limitation is omitted from this subsection. We will just mention that a 5-fold increase in the number of users-per-cache brings only a slight decrease in server traffic for both systems.

9.2 Online Bookstore Resource

Table 9 summarizes the savings and cache usage for an online bookstore type resource. Remember, for this resource, in DE, a client is likely to get a **different** base file on every access. Thus, the negative results for the client link savings with DE reflect more generated traffic than the

regular scheme or ESI. On the server side, however, DE reduces traffic more than ESI. If we add backend servers, ESI server traffic reduction goes from 30% to 56%, still less beneficial than DE.

Table 9 - Online bookstore comparison

	Savings client link	Savings server link	Edge cache usage
ESI	0%	30%	1.2M
DE	-8%	82%	3.3M

Figure 13 shows the traffic per request on the client and server links as a function of request arrival rate. Since in this case, for class-based DE, almost every client request forces the client to get a new base file. Therefore, the clients' traffic does not benefit from increase in arrival rate like we have seen for My Page. However, both ESI and DE main site traffic per request is again reduced as arrival rate increases, due to better caching.

Figure 14 illustrates the overall cost for the same example values of C_1 and C_2 (25 and 5, respectively) as a function of the request arrival rate. The more distinct improvement in DE (compare to the improvement in ESI) is due to better caching on the proxy caches as the request rate grows, reducing the expensive traffic from the servers. For ESI, however, the hit-rate does not improve as dramatically with the arrival rate, mainly due to the much larger number of items in the "pool" (corresponding to 400,000 books). For higher request rates, or for other resources, maybe with fewer data-items, the savings can possibly grow more considerably. For this resource, in both systems, the effect of the number of users-per-cache (or number of caches) is analogous to the effect of the arrival rate on performance.

Finally, Figure 15 shows when DE outperforms ESI, as a function of the C_1 and C_2 factors, for three different arrival rate values. Again, the area underneath each line is where DE cost is lower. Note the ratio between C_1 and C_2 needed for equality is much lower in this case than in My Page case. For instance, when the arrival rate is 1 and C_2=3, we only need C_1=15 for both schemes to perform equally.

10. ADDITIONAL CONSIDERATIONS AND CONCLUSIONS

Table 10 summarizes some of the conclusions and additional considerations for comparing class-based DE to ESI. A major concern is the computational costs and latencies introduced by both schemes. Class-based DE requires generation of the entire page for each request. Moreover, the page then needs to be delta-encoded, which includes the process of finding a good base file and computing the delta. On the client side, the delta should be applied to the base file accounting for more delay. ESI, on the other hand, requires only assembly of the fragments of the page on edge servers into a full page that is served to the client. The assembly does not introduce new computational costs since it had to be done by the web

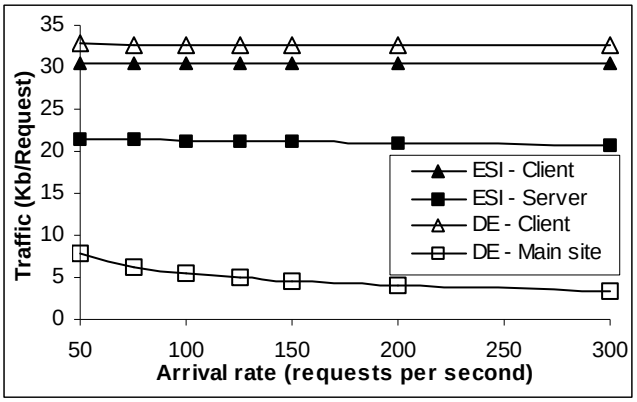


Figure 13 – Online Bookstore resource, ESI and class-based DE traffic vs. request arrival rate

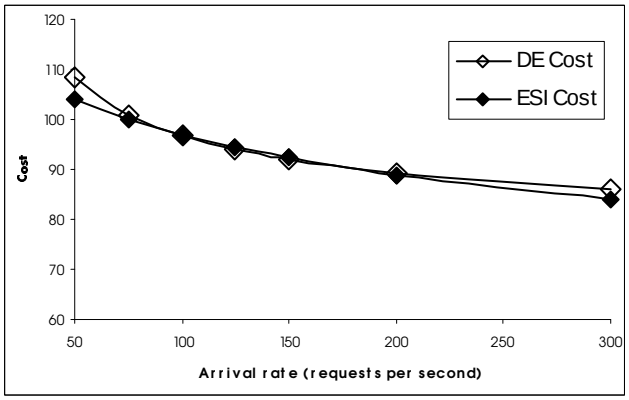


Figure 14 – Online Bookstore, cost of ESI and class-based DE vs.. request arrival rate

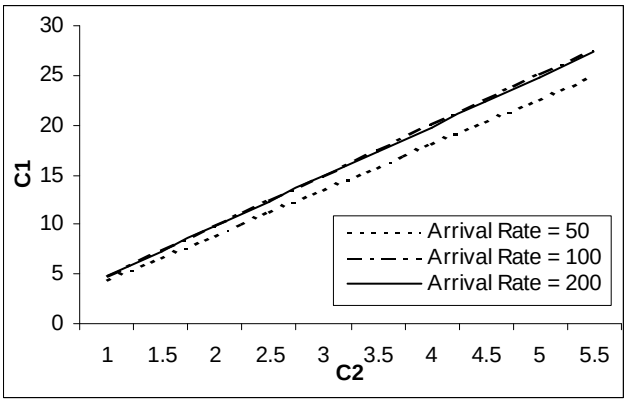


Figure 15 – Online Bookstore, values for C1, C2 where DE cost is equal to ESI cost

server even without ESI. The web server benefits twice under ESI: not only does it not have to assemble the page; in many cases it is required to deliver only small parts of the page.

Another consideration is the transparency of both systems. While class-based DE as offered in [1] requires installation of hardware or software, usually near the web server, it does not require any change to web-pages code, and works transparently with existing network infrastructure of proxy caches and clients. ESI, on the other hand, requires changes to web-pages code, as ESI code must be added over the

original HTML. In addition, ESI requires specialized edge servers (i.e., the services of a CDN provider) since the assembly directives are not implemented on proxy caches.

Table 10 – Summary: Excellent *, Good +, Bad -, Sometimes ~

	ESI	DE
Reduces server traffic	+	*
Reduces client traffic	-	~
Reduces load on web server	*	-
Performance dependent on web page structure	Yes	Yes
Performance dependent on characteristics of data	Yes	No
Benefits greater when popularity rises	Yes	Less
Requires main site hardware/software installation	No	Yes
Requires web-page code changes	Yes	No
Requires network infrastructure (CDN services)	Yes	No
Can exploit information available from CDN for page construction	Yes	No

The distribution of work in ESI between the main site and the CDN enables the web site to use information that is available to the CDN provider only, or not maintained by the main site, such as physical location of the clients. This can be exploited in the page construction (e.g., automatically inserting relevant weather) using the ESI language.

In conclusion, we showed that both ESI and DE can reduce traffic and improve caching for dynamic page delivery. However, the benefits of the techniques are highly dependent on the resource. For example, ESI is beneficial when there are a small number of items on the page, the item popularity is skewed, and their time-to-live is high. DE, while always good for reduction of main site traffic, may not always reduce client traffic.

11. ACKNOWLEDGEMENTS

We would like to thank Peter Danzig for helpful discussions, Phong Vo for support with the *vcdiff* software, and the reviewers for their helpful comments.

12. REFERENCES

[1] Konstantinos Psounis. Class-based delta-encoding: a scalable scheme for caching dynamic Web content. 22nd International Conference on Distributed Computing Systems Workshops, 2-5 July 2002, Vienna, Austria.

[2] Jim Challenger, Arun Iyengar, Karen Witting, Cameron Ferstat, and Paul Reed. A publishing system for Efficiently Creating Dynamic Web Content. IEEE INFOCOM 2000.

[3] Pei Cao, Jin Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98).

[4] ESI 1.0. http://www.w3.org/TR/esi-lang.

[5] Fred Douglis, Antonio Haro, and Michael Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. USENIX Symposium on Internet Technologies and Systems, 1997.

[6] Chris Roadknight, Ian Marshall, and Debbie Vearer. File Popularity Characterization. Second Workshop on Internet Server Performance (WISP), 1999.

[7] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential Benefits of Delta Encoding and Data Compression for HTTP. ACM SIGCOMM, 1997.

[8] FineGround Networks. Breaking New Ground in Content Acceleration. http://www.fineground.com/pdf/FGCWhitepaper.pdf.

[9] Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic Deltas for WWW Latency Reduction. USENIX Technical Conference, 1997.

[10] B.C. Housel and D.B. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. Second Annual International Conference on Mobile Computing and Networking, 1996.

[11] James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. Delta Algorithms: An Empirical Analysis. ACM Transactions on Software Engineering and Methodology, 7:192-214, 1998

[12] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and and Henry M. Levy. On the scale and performance of cooperative Web proxy caching. The 17th ACM symposium on Operating Systems Principles (SOSP í99), 1999.

[13] Mor Naaman, Hector Garcia-Molina, and Andreas Paepcke. Evaluation of Delivery Techniques for Dynamic Web Content (Extended Version). Technical Report Number 2002-31. Stanford University, 2002. Available at http://dbpubs.stanford.edu/pub/2002-31.

[14] Michael Rabinovich, Zhen Xiao, Fred Douglis, and Chuck Kalmanek. Moving Edge-Side Includes to the Real Edge - the Clients. USENIX Symposium on Internet Technologies and Systems. 2003.

[15] Jim Challenger, Paul Dantzig and Arun Iyengar. A Scalable System for Consistently Caching Dynamic Web Data. 18th Annual Joint Conference of the IEEE Computer and Communications Societies, New York, New York, 1999.

[16] David G. Korn and Kiem-Phong Vo. Engineering a Differencing and Compression Data Format. USENIX, 2002.

[17] George Kingsley Zipf. Relative frequency as a determinant of phonetic change. Reprinted from the Harvard Studies in Classical Philology, XL, 1929.

[18] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing Reference Locality in the WWW. PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems, 1996.

[19] Steven Glassman. A caching relay for the world wide web. The First International World-Wide Web Conference, 1994.

[20] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. Infocom, 1999.

[21] V. N. Padmanabhan and L. Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. ACM SIGCOMM, Stockholm, Sweden, 2000.