# On Reducing Dynamic Web Page Construction Times

Suresha and Jayant R. Haritsa

Dept. of Computer Science and Automation,
Indian Institute of Science, Bangalore 560012, INDIA

**Abstract.** Many web sites incorporate dynamic web pages to deliver customized contents to their users. However, dynamic pages result in increased user response times due to their construction overheads. In this paper, we consider mechanisms for reducing these overheads by utilizing the excess capacity with which web servers are typically provisioned. Specifically, we present a caching technique that integrates fragment caching with anticipatory page pre-generation in order to deliver dynamic pages faster during normal operating situations. A feedback mechanism is used to tune the page pre-generation process to match the current system load. The experimental results from a detailed simulation study of our technique indicate that, given a fixed cache budget, page construction speedups of more than fifty percent can be consistently achieved as compared to a pure fragment caching approach.

## 1 Introduction

Web sites are increasingly shifting from a static web page service model to a *dynamic* web page service model in order to facilitate delivery of custom content to users [4]. While dynamic web pages enable much richer interactions than static pages, these benefits are obtained at the cost of significantly increased user response times, due to the on-demand page construction. Dynamic web pages also seriously reduce the performance of the Web server due to the load incurred by the generation process. In fact, it has been recently estimated that server-side latency accounts for 40 percent of the total page delivery time experienced by end-users [6]. Hence, performance and scalability are becoming major issues for dynamic web sites.

To address these issues, a variety of optimization techniques have been developed in the recent literature. These include dynamic content-aware full-page caching, content acceleration, client-side prefetching, database caching, and fragment caching [1–5]. Among these techniques, *fragment caching*, which reduces dynamic page construction time by caching dynamic fragments, is particularly attractive since it provides the following desirable guarantees [3, 4]: Firstly, it ensures the *freshness* of the page contents by maintaining an association between the cached dynamic fragments and the underlying data sources. Secondly, it ensures the *correctness* of the page contents by newly generating the page skeleton each time the dynamic page is requested.

On the down side, however, fragment caching has some limitations: Firstly, its utility is predicated on having a significant portion of dynamic fragments to be cacheable – however, such cacheability may not always be found in practice. Secondly, even when most fragments are cacheable, dynamic page construction is begun only upon receiving

the request for the page – therefore, the server latency may still turn out to be considerable.

In this paper, we consider the possibility of achieving significant reductions in server latencies, and thereby user response times, by resorting to dynamic page *pre-generation*, in conjunction with fragment caching. The pre-generation is based on having a statistical prediction mechanism for estimating the next page that would be accessed by a user during a session. The page pre-generation is executed during the time period between sending out the response to the user's current request and the receipt of her subsequent request. Note that in the case where the page prediction turns out to be right, the pre-generation effectively reduces the server latency to *zero*, which is the best that could be hoped for from the user perspective.

An unsuccessful pre-generation on the other hand represents wasted effort on the part of the server. This may not be an issue for web-servers that are under normal operation since these systems are usually over-provisioned in order to handle peak loads [8], and therefore some wastage of the excess capacity is not of consequence. But, during peak loads, the additional effort may further exacerbate the system performance. To address this problem, we incorporate a simple linear feedback mechanism that scales down the degree of pre-generation to match the current system load.

A related design issue is that we need to allocate space in the server cache to store the pre-generated pages. That is, the cache has to be *partitioned* into a fragment space and a page space, and the relative sizing of these partitions has to be determined.

Our hybrid approach of combining pre-generation with fragment caching ensures the freshness of content through either fresh computation or by accessing fragments from the fragment cache. Further, it ensures the correctness of pages by pre-generating pages specific to users. In a nutshell, our approach achieves both the *long-term benefit through fragment caching* and the *immediate benefit through anticipatory page pre-generation*.

Using a detailed simulation model of a dynamic web-server, we study the performance of our hybrid approach in terms of reducing dynamic page construction times, as compared to pure fragment caching and pure pre-generation approaches. Our evaluation is conducted over a range of fragment caching levels and prediction accuracies, for a given cache budget. The results show that under normal loads, we are able to achieve reductions in server latency by over fifty percent on average as compared to pure fragment caching, whereas under heavy loads, we do no worse. Further, the number of pages delivered with *zero server latency* is proportional to the prediction accuracy.

To summarize the contributions of this paper:

1. We propose a hybrid approach to reduce dynamic web page construction times.
2. We demonstrate that robust settings exist for the relative sizing of the cache partitions for pre-generated pages and fragments, respectively.
3. We incorporate a simple linear feedback mechanism to ensure that the system performance is always as good or better than that of pure fragment caching.
4. Our experimental results show that significant improvements in page generation times can be achieved through the hybrid approach as compared to fragment caching.

## 2 A Hybrid Approach to Dynamic Page Construction

In this section, we describe in detail our proposed hybrid architecture. Before discussing our new approach, we first provide background material on fragment caching and page prediction techniques.

**Fragment Caching:** A request for a dynamic web page corresponds to a script, which is essentially a set of code blocks. Each code block carries out some computation to generate a part of the required page, and results in an HTML fragment. An output statement after the code block places the resulting HTML fragment in a buffer. Once all the code blocks in a script have been executed, the resulting HTML is sent as a page to the user. If we know that a code block's output does not change for a sufficiently long time, then such a code block can be tagged as cacheable. When the script is executed, these tags instruct the application server to first check for the fragment in the fragment cache. If the requested fragment is found in the cache, the code block execution is bypassed and the content is returned from the cache. If not, the code block is executed and the fragment is generated freshly and also cached for future benefit. The cache contents are managed by a cache replacement policy and an invalidation mechanism. A cached object is invalidated whenever the underlying data source updates the data values on which the object is dependent. The details of fragment caching are available in [3, 4] – in the rest of this paper, we assume the use of the fragment caching technique proposed in [3, 4].

**Page Prediction:** There are several page access prediction models that have been proposed in the literature [7–12], based on information gained from mining web logs. These models can be classified into two categories: *point-based* and *path-based*. The point-based models predict the user's next request solely based on the current request being served for the user. On the other hand, the path based prediction models are built on entire request paths followed by users. The path based predictions use a path profile, which is a set of pairs, each of which contains a path and the number of times that path occurs over the period of the profile. The profiles can be generated from standard HTTP server logs and the accuracy of these models has been found to be high enough to justify the pre-generation of dynamic content [8] – in the rest of this paper, we assume the use of such a path prediction model.

### 2.1 Combining Page Pre-Generation and Fragment Caching

Our proposed hybrid model is an integration of anticipatory page pre-generation and fragment caching. A high level representation of the proposed hybrid architecture is given in Figure 1.

Here, for each individual user session, when a response for a request leaves the web server, the web server decides whether or not to pre-generate the next most expected page for the associated user, based on considerations such as the system's current load, the benefit of pre-generating a page, the type of user and so on. If the system decides to pre-generate a page for a particular user, it requests the page pre-generator to carry out

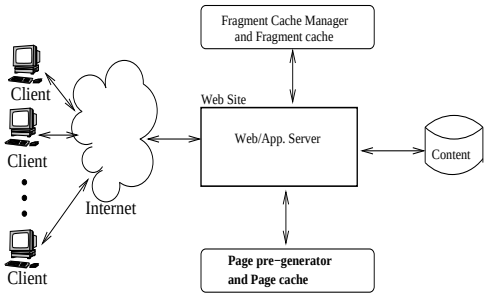


**Fig. 1. The Proposed Hybrid Model**

the generation for this page. When the web server receives the next page request from this user, it checks whether the page is already available with the page pre-generator. If the page is available, the page is immediately served. If not, the page is freshly computed by the web/application server as usual. The page pre-generator retains only the pre-generated pages of current active users.

Note that the user response leaving the web server will take some time to reach the user and the user will then take some time to click the next page. We expect that under normal operating conditions, this time delay is sufficient for the page pre-generator to complete the page generation process before the arrival of the next request of the same user. The implication is that in case of a correct prediction, the server latency in terms of page construction time is *brought down to zero*.

Further, note that the proposed solution is guaranteed to serve *fresh* content, since it is associated with the origin server. Moreover, it also ensures serving *correct* pages, since the page pre-generation is specific to the user session and is not generic across users. From a broad perspective, by fragment caching we are achieving the long-term benefit whenever the fragment is reused in course of time. Whereas by page pre-generation we are achieving the immediate benefit for the current user.

### 2.2 Server Cache Management

In a pure fragment caching approach, the server cache can be used solely for hosting these fragments. However, in our hybrid approach, we need to allocate space for hosting pre-generated pages as well. Therefore, we partition the cache into a *fragment cache* and a *page cache*.

**Cache Partition Sizing :** An immediate issue that arises here is determining the relative sizes of the fragment and page cache partitions. This issue is investigated in detail in our experimental study presented in Section 4 – our results there indicate that a 50-50 partitioning works well across a range of pre-generation accuracies and fragment cacheability levels.

**Cache Replacement Policies :** With regard to the fragment cache, we are not aware of any web logs that are available to track the reference patterns for fragments. This re-

stricts us to the use of simple techniques like Least Recently Used (LRU) for managing the fragment cache.

With regard to the page cache, we do not expect to require an explicit replacement policy since the utility of pages in the cache is typically short-lived – that is, until the arrival of the next request by the user – after this arrival, the page is immediately vacated from the cache. However, to address those uncommon cases where the page cache is completely filled with active pre-generated pages, we adopt the simple mechanism of blocking further page pre-generations until some of the existing pages expire.

An association between the fragments in the fragment cache and the pre-generated pages in the page cache is maintained by the page pre-generator. Whenever a fragment is invalidated, all the pre-generated pages associated with it are marked invalid.

### 2.3 Server Load Management

While page pre-generation is useful for reducing response times, it also involves expense of computational resources. This is acceptable under normal operating conditions, even if the page prediction accuracy is not good, since web-servers are typically over-provisioned in order to be able to handle peak load conditions [8], and we are only using this excess capacity. But, when the system is under peak load conditions, the wasted resources due to the mistakes made by the pre-generation process may actually exacerbate the situation, driving the system into *a worse condition*. To address this issue, we implement a simple linear feedback mechanism that modulates the pre-generation process to suit the current loading condition. Specifically, we periodically measure the system load, and if it exceeds a threshold value, the role of the page pre-generator is restricted in proportion to the excess load.

We have applied a simple linear feedback mechanism in our hybrid model to control the role played by the page pre-generator during the peak loads. Specifically, for each outgoing page response, the web server allows the page pre-generator to generate pages with probability $prob\_gen$ set as follows:

$prob\_gen = 1$ if ($current\_load < threshold\_load$)
$prob\_gen = \frac{maximum\_system\_load - current\_load}{maximum\_system\_load - threshold\_load}$ otherwise.

When the pre-generator is restricted, its assigned *cache partition* may become underutilized – therefore the size of the fragment cache is dynamically enlarged to cover the underutilization of the page cache.

## 3 Simulation Model

To evaluate the performance of the proposed hybrid model, we have developed a detailed simulator of a web-server supplying dynamic pages to users. Table 1 gives the default values of the parameters used in our simulator – these values are chosen to be indicative of typical current web-sites, with some degree of scaling to ensure manageable simulation run-times.

**Web-site Model:** The web site is modeled as a directed graph. Each node in the graph represents a dynamic web page. Each edge represents a link from one page to another

page. A node may be connected to a number of other nodes. The web-site graph is generated in the following manner: We start with a node called the root node, at level zero, and an initial fanout $FanOut$. Then, at each level $l$, for all nodes of that level, the next level nodes are created and linked, with a uniform random fanout ranging between $(0, FanOut - l)$. When a fanout of $0$ is chosen at a node, the generation process at that node is terminated. In order to model "back-links", we permit, in the process of linking a node to other nodes, even the *previously generated nodes* of the prior levels to be candidates. The percentage of back links is determined by the $BackLinks$ parameter.

**Web-page Model:** Each dynamic web page consists of a static part and a collection of identifiable dynamic fragments. A fraction $FragCacheable$ of these dynamic fragments are cacheable, while the remaining are not. The number of fragments in a page are uniformly distributed over the range *(MinFragNum,MaxFragNum)* and are selected randomly from the *FragPopulation* fragments. The cost of producing a fragment, $FragCost$, is taken to be proportional to its size which is uniformly distributed over the range *(MinFragSize,MaxFragSize)*.

**User Model:** The web site receives requests from the sessions of different users. The creation of sessions is assumed to be Poisson distributed [13] with rate *ArrRate*. Each session generates one or more page requests, in a sequential manner. The number of pages in a session are uniformly distributed over the range *(MinSessionPage, MaxSessionPage)*. Between the page requests of a session, a uniformly distributed user think time over the range *(MinThinkTime, MaxThinkTime)* is modeled.

**System Model:** We assume that the web-server has a cache for dynamic page construction, of size $CacheSize$. The fraction of the cache given to the PageCache is given by *PageCacheFraction*, with the remainder assigned to the fragment cache. The search times in the page and fragment caches are determined by the *CacheSearchTime* parameter. The accuracy of page access prediction is determined by the $PagePredict$ parameter. The fragments in the fragment cache are modeled to be invalidated randomly by the data source with an invalidation rate set by $InvalidRate$. The threshold load at which the feedback control mechanism kicks in is set by the $ThresholdLoad$ parameter.

## 4 Experiments and Results

Using the above simulation model, we conducted a variety of experiments, the highlights of which are described here. The performance metric used in all our experiments is the average dynamic page construction time, evaluated for various settings: *LOW (20%)*, *MEDIUM (50%)* and *HIGH (80%)* of the page prediction accuracy and the cacheability of the dynamic fragments, as a function of the session arrival rate and the fraction of the cache assigned to page pre-generation. Covering these variety of values permits the modeling of a range of real-life web-site environments. Also, the arrival rates are set so as to model both normal loading conditions as well as peak load scenarios.

**Table 1. Simulation parameter settings**

| | | | |
|---|---|---|---|
| MinSessionPage | 1 | MaxSessionPage | 19 |
| MinPageSize | 10KB | MaxPageSize | 30KB |
| MinFragNum | 1 | MaxFragNum | 19 |
| MinThinkTime | 1 second | MaxThinkTime | 9 seconds |
| MinFragSize | 1KB | MaxFragSize | 3KB |
| FragPopulation | 8000 | CacheSize | 2MB |
| PageCacheFraction | 0 to 100 percent | FragCost | 20 ms |
| FanOut | 10 | BackLinks | 20 percent |
| ArrRate | 0 to 5 sessions per second | InvalidRate | 1/ms |
| PagePredict | 20, 50, 80 percent | CacheSearchTime | 0.1 ms |
| FragCacheable | 20, 50, 80 percent | ThresholdLoad | 75 percent |

### 4.1 Suite of Algorithms

To put the performance of our approach in proper perspective, we compare it against the following three yardstick algorithms:

**Hybrid:** This is our new algorithm in which pre-generation and fragment caching are simultaneously used, and the cache is partitioned into a page and a fragment cache.
**Pure_FC:** This algorithm implements pure fragment caching (with no pre-generation).
**Pure_PG:** This algorithm implements pure page-generation (with no fragment caching).
**NO_GC:** Neither fragment caching nor page pre-generation is used here, and the cache does not come into play at all.

### 4.2 Experiment 1: Page Construction Times (Normal Load)

In our first experiment, we evaluate the dynamic web page construction times under normal loading conditions. Here, both the fragment cacheability level and the page prediction accuracy are set to MEDIUM (50 percent), and the cache memory is *equally* partitioned between the page cache and the fragment cache. For this scenario, Figure 2(a) gives the relative performance of the four dynamic web page construction algorithms as a function of the session arrival rate. We see here that:

- The HYBRID approach performs the best across the entire normal loading range. Further, it requires less than *half the time* to construct pages as compared to fragment caching, the policy that has been advocated in recent literature.
- The utility of caching and pre-generation are indicated by the significant improvement in performance that are provided by HYBRID, Pure_PG and Pure_FC, as compared to No_GC which is completely impervious to caching/pre-generation.
- While the performance of HYBRID, Pure_FC and No_GC is flat across the loading range, the Pure_PG approach begins to progressively do worse as the load is increased. This is because of the extra load that is imposed by the pre-generation process. In contrast, HYBRID, while also incorporating pre-generation, does not suffer from the problem because of its fragment caching component.

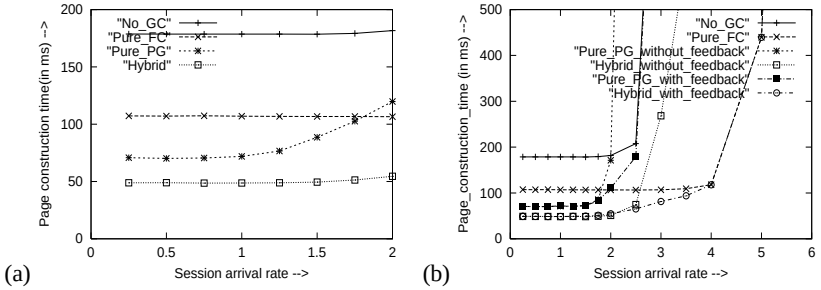


**Fig. 2. Page Construction Times : (a) Normal Load (b) Peak Load**

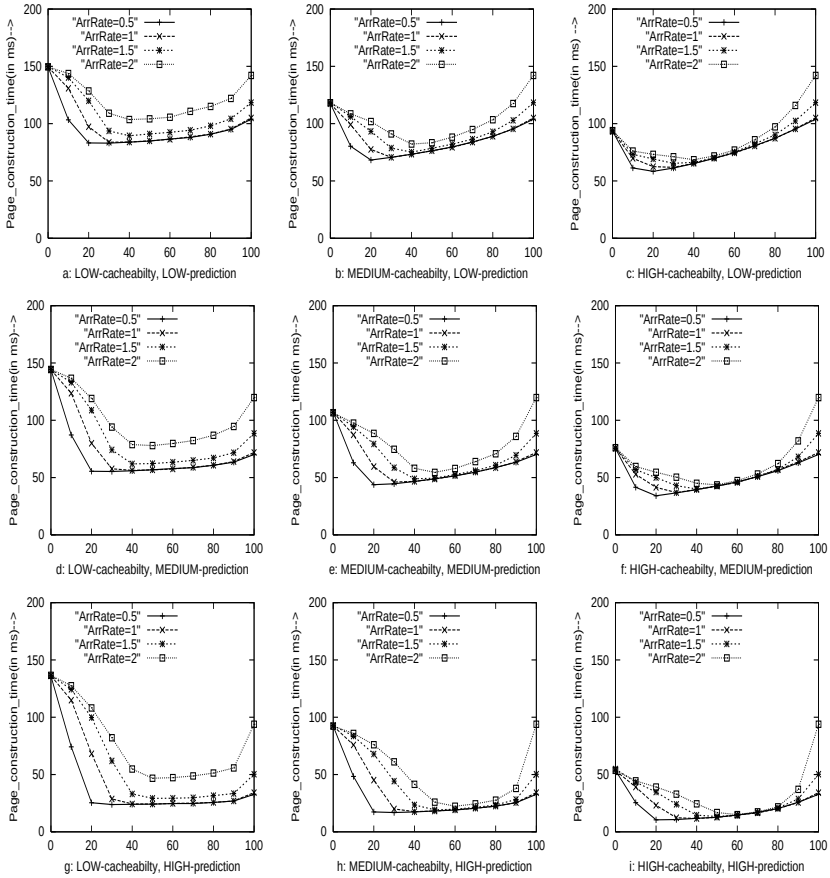


**Fig. 3. Cache Partitioning**

### 4.3 Experiment 2: Peak Load Performance

We now evaluate the performance under transient peak load situations which all web-servers experience from time to time. For this experiment, we present the performance of the HYBRID and Pure_PG approaches, both with and without the feedback mechanism, to evaluate the effectiveness of this mechanism. The page construction performance for this experiment is shown in Figure 2(b). We see here that:

- The HYBRID-with-feedback approach performs the best across the entire loading range. As the load moves into the peak-loading region, this approach progressively reduces the role of pre-generation, finally winding up eliminating it completely and becoming identical to Pure_FC.
- The benefits of feedback are clearly shown by comparing the with-feedback and without-feedback versions of HYBRID and Pure_PG.

### 4.4 Experiment 3: Cache Partitioning

We now investigate the performance impact on HYBRID of different cache partitionings – this is done over the entire range of fragment cacheability levels (Low, Medium and High) and page prediction accuracies (Low, Medium and High), resulting in nine different combinations. The results for all these combinations are shown in Figure 3, where we observe the following:

- All of them have a "cup shape" with the highest construction times being at the extremes (0% page cache and 100% page cache), and the lowest somewhere in between.
- For the LOW prediction scenario (Figures 3a-c), the best overall partitioning is about 40 percent page cache, while for the MEDIUM and HIGH prediction scenarios (Figures 3d-f and 3g-i), the best partitioning is 50 percent page cache and 60 percent page cache, respectively.
- While the best partitionings are a function of the prediction accuracy as mentioned above, using a value of 50 percent page cache is very *close to the best in all the graphs*. That is, with this setting we are assured almost-optimal performance across the entire range of web-server scenarios.
- Note that the setting of 0 percent page cache is equivalent to a *Pure_FC* approach. We observe that the performance of Pure_FC is strongly dependent on the fragment cacheability level.

## 5 Conclusions and Future Work

We have proposed a hybrid approach to reduce dynamic web page construction times by integrating fragment caching with page pre-generation, utilizing the spare capacity with which web servers are typically provisioned. Through the use of a simple linear feedback mechanism, we ensure that the peak load performance is no worse than that of pure fragment caching.

We made a detailed study of the hybrid approach over a range of cacheability levels and prediction accuracies, for a given cache budget. Our experimental results show that an even 50-50 partitioning between the page cache and the fragment cache works very well across all environments. With this partitioning, we are able to achieve over fifty percent reduction in server latencies as compared to fragment caching. In summary, our approach achieves both the long-term benefit through fragment caching and the immediate benefit through anticipatory page pre-generation.

Currently, we restrict the pre-generation to the single most likely page. In our future work, we plan to investigate the performance effects of pre-generating a set of pages, rather than just a single page.

**Acknowledgements:** This work was supported in part by a Swarnajayanti Fellowship from the Dept. of Science & Technology, Govt. of India.

## References

1. A. Eden, B. Joh and T. Mudge, "Web Latency Reduction via Client-Side Prefetching", *Proc. of the IEEE Intl. Symp. on Performance Analysis of Systems & Software, 2000.*
2. A. Iyengar and J. Challenger, "Improving Web Server Performance by Caching Dynamic Data", *Proc. of the Usenix Symp. on Internet Technologies and Systems, 1997.*
3. Chutney Technologies, Inc. "Dynamic Content Acceleration: A Caching Solution to Enable Scalable Dynamic Web Page Generation", *Proc. of the ACM SIGMOD Intl. Conf. on Mgmt. of Data, 2001.*
4. A. Datta, K. Dutta, H. Thomas, D. VanderMeer, K. Ramamritham and D. Fishman, "A Comparative Study of Alternative Middle Tier Caching Solutions to Support Dynamic Web Content Acceleration", *Proc. of the 27th VLDB Conf., 2001.*
5. Q. Luo, J. Naughton, R. Krishnamurthy, P. Cao and Y.Li, "Active query caching for database web servers", *Proc. of the 3rd Intl. Workshop on the Web and Databases, 2000.*
6. C. Huitema, "Network vs. server issues in end-to-end performance", Keynote address, Performance and Architecture of Web Servers Workshop, 2000.
7. I. Zukerman, D. Albercht and A. Nicholson, "Predicting Users' Requests on WWW", *Proc. of the 7th Intl. Conf. on User Modeling, 1999.*
8. S. Schechter, M. Krishnan and M. Smith, "Using Path Profiles to Predict HTTP Requests", *Proc. of the 7th Intl. World Wide Web Conf., 1998.*
9. Z. Su, Q. Yang, Y. Lu and H. Zhang, "WhatNext: A Prediction System for Web Requests using N-gram Sequence Models", *Proc. of the 1st Intl. Conf. on Web Information System and Engineering, 2000.*
10. Z. Jiang and L. Kleinrock, "Prefetching Links on the WWW", *Proc. of the IEEE Intl. Conf. on Communications, 1997.*
11. D. Duchamp, "Prefetching Hyperlinks", *Proc. of the 2nd USENIX Symp. on Internet Technologies and Systems, 1999.*
12. Z. Wang and J. Crowcroft, "Prefetching in World Wide Web", *Proc. of the IEEE Global Telecommunications Internet Mini-Conf., 1996.*
13. M. Andersson, J. Cao, M. Kihl and C. Nyberg, "Performance Modeling of an Apache Web Server with Bursty Arrival Traffic", *Proc. of the Intl. Conf. on Internet Computing, 2003.*