Content Distribution for Publish/Subscribe Services

Mao Chen, Andrea LaPaugh, and Jaswinder Pal Singh

Department of Computer Science, Princeton University
35 Olden Street, Princeton, NJ 08544, USA
{maoch,aslp,jps}@cs.princeton.edu

Abstract. Caching and content delivery are important for content-intensive publish/subscribe applications. This paper proposes several content distribution approaches that combine match-based pushing and access-based caching, based on users' subscription information and access patterns. To study the performance of the proposed approaches, we built a simulator and developed a workload to mimic the content and access dynamics of a busy news site. Using a purely access-based caching approach as the baseline, our best approaches yield over 50% and 130% relative gains for two request traces in terms of the hit ratio in local caches, while keeping the traffic overhead comparable. Even when the subscription information is assumed not to reflect users' accesses perfectly, our best approaches still have about 40% and 90% relative improvement for the two traces. To our knowledge, this work is the first effort to investigate content distribution under the publish/subscribe paradigm.

1 Introduction

The information needs of content consumers form a key to driving content delivery over the Internet. Typically, these information needs are determined based on access patterns and pre-determinations of popular resources.

Many web-based notification services are based on users' subscriptions, which are statements of interest. The stated interest can therefore also be used as a basis for caching and content distribution. Little exploration of this use has been done.

An example is the notification services at news sites. A user indicates the categories or keywords of the news of interest; the news site notifies the user with a list of titles when it publishes news that matches the user's subscription. If the user wants to read an article in the list, the user requests the actual content from the origin site. These types of services are usually known as publish/subscribe applications.

In the literature, most work on publish/subscribe systems examines event routing and efficient matching. However, content distribution in publish/subscribe services is an important module that has not been adequately studied. Content delivery is usually ignored because the existing publish/subscribe applications assume subscribers are only interested in short messages rather than large-size contents. However, this assumption does not hold for many applications, such as news delivery, in which the object of interest (to which the notification might only carry a link) may embed long

M. Endler and D. Schmidt (Eds.): Middleware 2003, LNCS 2672, pp. 83–102, 2003.

texts, images and video/audio streams. In addition to this need, the time decoupling in publish/subscribe services, which means information producing and consuming occur asynchronously, creates an opportunity for early content distribution. Of course, because of the constraints on the storage at subscriber-side machines and on the Internet bandwidth, it is neither realistic to store all the matched contents at the subscriber side until users read them nor efficient to leave all the contents at the publisher side until users request them, so dynamic content distribution strategies must be developed.

This paper presents a set of content distribution strategies for publish/subscribe systems. The different approaches can be classified along two axes, which also expose the key design issues: (i) *when* is the opportunity for placing a page into a cache; (ii) *how* (on what basis) the placement and replacement decisions are made at evaluation time. The two major possibilities for *when* are (a) at match time, i.e. when a page is determined to match certain subscriptions and (b) at access time, as in traditional caching systems. The two major possibilities for *how* are (a) based on access patterns only, as in traditional caching systems, and (b) based on subscription information and matching.

A major challenge in such a study is that of developing workloads. No real-world publish/subscribe workloads are available for such studies. We therefore have developed workloads based on studies of observed access patterns at busy sites, extrapolating from there to publish/subscribe workloads. In particular, to study the performance of our approaches, we simulate the news delivery to subscribers who are geographically distributed. The publishing pattern and the access dynamics are simulated according to a study on one of the busiest media sites, MSNBC [24].

In our experiments, the performance metrics are: (i) the hit ratio in the local proxy servers, since the major goal of this work is to reduce the response time perceived by end-users, and (ii) the traffic overhead, which is measured using the network traffic for transferring contents from the publisher site to the proxies of subscribers. Using a purely access-based caching approach as the baseline, our approaches improve the hit ratio dramatically while keeping the traffic overhead comparable.

The major contributions of this paper are as follows:

1. Presenting the first study of content delivery and caching that uses publish-subscribe information;
2. Proposing and comparing a set of solutions for content delivery in publish-subscribe services, based on subscription information as well as access patterns;
3. Experimentally demonstrating the benefit of our approaches in reducing the response time to end users without extra overhead in network traffic;
4. Developing realistic workloads for evaluation, a major challenge given that publish/subscribe workloads are not generally available;
5. Building a simulator to study content delivery in globally distributed servers.

The rest of this paper is organized as follows. Section 2 outlines the architecture of a content delivery system that this paper addresses. Section 3 presents several information delivery mechanisms in the system. Section 4 discusses a news delivery workload and a simulator that are used to evaluate the approaches. The simulation results are demonstrated and analyzed in section 5. Section 6 discusses related work. Section 7 draws conclusions and indicates future directions for this research.

2 A Publish/Subscribe System with Content Distribution Engine

Publish/subscribe is an asynchronous communication paradigm for information producers and information consumers. The producers and consumers are globally distributed and do not have to know each other. Information consumers declare their information needs to a publish/subscribe system that notifies the subscribers when published information matches the users' subscriptions.

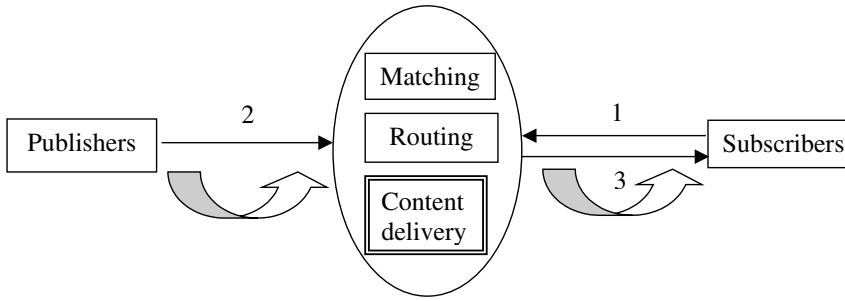


Fig. 1. Architecture of a publish/subscribe system

Figure 1 outlines a conceptual architecture of a publish/subscribe system. Publishers and subscribers (end-users) are connected via the publish/subscribe system. Notification services are usually implemented through three basic communication streams as labeled in the figure:

1. Users subscribe, announcing their interests to the system;
2. Producers publish contents to the system;
3. The system notifies the users whose subscriptions match the contents.

A typical system consists of a matching engine and a routing engine. After a piece of content is published into the system (flow 2), the matching engine finds the users who are interested in the events according to their interest profiles, and the routing engine delivers events to those users. These engines may be centralized or distributed.

This paper adds a content distribution and caching (or content delivery) engine, which is not discussed in the literature. After step 3, the notified users may choose to read the actual content of the event; the content distribution module in figure 1 is in charge of deciding when to deliver which content to the subscribers.

In this study, the caching/content-delivery servers are deployed as proxy servers close to the end-users. Each proxy server connects to a group of clients/subscribers. A proxy server aggregates its users' subscriptions and processes notifications for its users. It also serves as the cache that is consulted when one of its users accesses content. In this paper, the terms proxy, proxy server and server are used synonymously for content distribution server.

3 Content Delivery Strategies

As discussed earlier, content delivery strategies can be classified along two axes, namely *when* and *how* pages are evaluated for placement in caches.

Regarding *when*, the content distribution engine has two obvious opportunities to deliver content from a publisher to a proxy server. In the first case, the publisher proactively forwards a page to a proxy for potential placement when the matching engine determines that the content of the page matches the subscriptions of some users at the proxy. This approach, which we call the *push-time* strategy, assumes that the subscribers are likely to request the page later after receiving the notifications. The second scenario, which we call the *access-time* strategy, is like traditional caching and is based on the fact of rather than the prediction of users' accesses to a page.

Since a server is limited in physical storage capacity, when a page is delivered to a server in either the push-time or the access-time approaches, the server may need to replace some content at the server if the server's cache is already full. Replacement is based on values given to pages. Regarding *how* the value is determined, it can be done based on subscription and matching information or on actual access information.

This paper focuses on the following interesting combinations of when and how evaluations are made for page placement in caches:

1. Access-time strategy based on usage pattern only
2. Push-time strategy based on subscription and matching information only
3. Push-time and access-time strategy based on subscription and access pattern

The first combination is the traditional caching approach and is the baseline used in this study. The second case is a simple pushing mechanism driven by matching. This paper puts the emphasis on the third class of schemes that exploit both placement opportunities and both types of information about the value of pages.

3.1 Access-Time Strategy Based on Access Pattern

This paper uses a new caching replacement algorithm called Greedy-Dual* (GD*) [17] as the baseline algorithm, since it yields higher hit ratio than LRU, Greedy-Dual-Size (GDS) [7] and LFU-DA in an experimental study [17]. GD* determines the value of a page $V(p)$ based on the access frequency, the access recency, the cost to fetch a page, and the size of the page, as represented in equation 1:

$$V(p) = L + \left(\frac{f(p) \cdot c(p)}{s(p)} \right)^{1/\beta}. \quad (1)$$

Where

L: inflation value to capture the access recency
$f(p)$: number of accesses on the page
$c(p)$: cost to fetch a page from the publisher
$s(p)$: page size
β: balance factor of popularity and temporal correlation

In our implementation, the reference count of a page is discarded when the page is evicted, as in the In-Cache LFU algorithm [17]. As suggested by [7], our implementation uses the network distance to the origin publisher to measure the cost to fetch a page for a given proxy, where the network topology of proxy servers and the publishers is a random graph built using BRITE [6]. The constant parameter β is set manually as discussed in the experiment section (section 5).

On an access hit of page *p*, the replacement algorithm GD* increases the reference count *f*(*p*) and re-evaluates the page based on the current inflation value *L*. On a cache miss, all the pages in the cache are sorted by values and are evicted from the least valuable one, until there is room for the requested page; the inflation value *L* is set to be the value of the page that is evicted last. The following is the pseudo-code of GD*:

```
Replacement algorithm in GD*
  L←0.0
  For each request in turn:
    The current request is for page p:
      If p is already in memory
        Increase f(p);
      Else
        While there is not enough room
          L ← min {V(k) ∈ pages in the cache}
          Evict q s.t. V(q) = L
        Bring p into cache;
      V(p) ← L+( f(p)•c(p) / s(p) )^(1/β) ;
end.
```

3.2 Push-Time Strategy Based on Subscription Information and Matching

For a page that matches some subscriptions aggregated on a proxy, the number of the subscriptions indicates the number of requests of the page in future. The value of a page in a push-time replacement is based on the number of end-users' subscriptions that match the page, the cost and the size of the page as described in equation 2.

$$V(p) = \frac{f_S(p) \cdot c(p)}{s(p)}. \tag{2}$$

Where

$f_S(p)$: the number of subscriptions matching the content of page p

$c(p)$ and $s(p)$ have the same meaning as in equation 1

If the cache of a destination proxy is full in a push-time placement, the new page is evaluated and compared to the existing pages in the proxy. The pages whose values are less than that of the new page are candidates for replacement. The candidate pages are sorted by value and are evicted one by one until the available storage on the proxy is large enough for the new page. This placement algorithm using the subscription-based push-time strategy is denoted as SUB in this paper.

When an event is generated and routed to a destination proxy, SUB may decide not to store the new page if the total size of all the candidate pages (the pages whose values are smaller than that of the new page) is less than that of the new page. As a push-time only strategy, on a cache miss, SUB fetches the requested page from the publisher and forwards the page to the user without caching it in the local server.

3.3 Push-Time and Access-Time Schemes Based on Subscription Information and Access Pattern

We developed several approaches within the category that performs both push-time and access-time placement. The approaches run two independent placement modules at push-time and access-time, and are distinct in whether the same replacement algorithm is used in the two modules and in how a server's cache is configured.

Single Cache and Single Replacement Method. Using this type of approach, the push-time and the access-time placement modules share a server's cache and use the same replacement algorithm. Like SUB, whether to store a page on a server is purely based on the value of the page. As a consequence, the push-time placement does not store a new page in the local server if the page's size is larger than the total size of the candidate pages for eviction; on a cache miss, the replacement module discards the requested page immediately after forwarding it to the user if the page's value is not high enough to reside in the server's cache.

Within this framework, the evaluation function used in the replacement algorithm can analyze and combine the subscription and the usage information in several ways.

GD-based Approaches*. Since GD* provides a general framework to combine several factors related to a page's value, it is used as the basis to incorporate the subscription information. Two evaluation functions are developed based on GD*.

For a given page, the number of end-user subscriptions that match the page indicates the amount of references of the page in the future, while the number of accesses in the past exhibits the usage pattern of users. A direct way to combine the prediction and the history information is adding the two numbers together. The evaluation function based on this idea is as in equation 1 after replacing the frequency factor $f(p)$ by the sum of the number of subscriptions and of the accesses as in equation 3. This approach is referred to as Subscription-GD*-1 or *SG1* in this paper.

$$f(p) = s + a. \tag{3}$$

Where

s : the number of subscriptions matching page p

a : the number of accesses of page p

SG1 ignores the relationship between the references and the subscriptions of a page. Ideally, if every subscriber reads any page that matches his/her subscription exactly once, the difference between the number of subscriptions and that of past

requests is equal to the number of future references of a page. Based on this idea, an alternative uses the following equation to calculate $f(p)$, while keeping the other factors the same as in equation 1. The alternative is called Subscription-GD*-2 or *SG2*.

$$f(p) = s - a. \quad (4)$$

Where
s : the number of subscriptions matching page p
a : the number of accesses of page p

Frequency-based Approach. Using GD* as the framework, SG2 integrates the estimation of a page's reference frequency in the future with the access recency in the past. However, there is lack of proof about the correlation between the two factors. Therefore a new evaluation function that relies only on the frequency prediction is developed. This evaluation function is defined in equation 5. The approach using this evaluation method is referred to as *subscription-request* or *SR* in this paper.

$$V(p) = \frac{f(p) \cdot c(p)}{s(p)}. \quad (5)$$

Where
$f = s - a$

Single Cache and Dual Replacement Methods. The approaches that combine the subscription and the usage pattern into a single evaluation function are based on some assumptions on the relationship between the two patterns. For example, SG2 and SR assume every user requests all the pages that match his/her subscription exactly once.

An alternative is to use independent replacement algorithms as well as evaluation functions at push-time and access-time. Namely, GD* is applied in an access-time replacement, while SUB is used in a push-time placement. Since the two replacement algorithms use either access analysis or subscription information, the two types of information are used separately in different placement modules. This approach is referred to as Dual-Methods or *DM* in this paper.

Dual Caches and Dual Replacement Methods. A potential problem within *DM* is that a page that is in hot use will be replaced in a push-time placement if the number of subscriptions matching the page is not large enough. Or, on the other hand, a new page with a high future use indicated by subscription matching will be replaced on a cache miss just because of the few references up to the replacing time. This problem is due to the overlapping operations of the push-time and access-time placement modules in the same cache space.

To deal with the problem associated with *DM*, the cache on a proxy can be divided into two portions that are used by the push-time and the access-time modules independently. Under this scheme, the cache portion used by the push-time module is called *Push-Cache* or *PC* in this paper, while the portion used by the access-time module is called *Access-Cache* or *AC*. The new mechanism is denoted as *Dual-*

Caches or *DC* in the following discussions. Like *DM*, *Dual-Caches* runs GD* in the access-time module and SUB in the push-time module. But different from *DM*, each replacement algorithm runs only on the corresponding portion of a proxy cache.

Dual Caches with Fixed Partition (DC-FP). A simple way to handle a dual-cache is keeping a fixed partition on the storage. This approach is denoted as *Dual-Caches with Fixed Partition* or *DC-FP* in this paper.

When a page is pushed into a server, the push-time module tries to store the page into *PC* using SUB. When serving a request, *DC-FP* first checks *PC*. If the requested page is in *PC*, the page is moved from *PC* to *AC*; meaning that the page should be henceforth evaluated based on the access pattern and be compared with other referenced pages in *AC* in the cache replacement algorithm. Otherwise, GD* is called to handle the replacement on *AC* as a standard caching algorithm.

Dual Caches with Adaptive Partition (DC-AP). A fixed partition in a dual-cache lacks flexibility in adjusting the effectiveness of pushing and caching according to the content publishing dynamics and the access dynamics. For example, when a page in the push cache *PC* is requested, the page is moved to the access cache *AC*, which may trigger a replacement in *AC* if *AC* is full. However, the storage in *PC* that was cleared by moving out the requested page is unused at least until the next new page is published. In such a case, a better strategy is to reassign the storage of the requested page to the access cache. On the other hand, some storage in the access cache can be "devoted" to the push cache if there is no room to store a new page in *PC* **and** several old pages in *AC* have not been referenced for a while.

The approach that labels the storage of each page according to the publishing and request patterns is called *Dual-Caches with Adaptive Partition* or *DC-AP* in our discussion. Using *DC-AP*, when a page cannot be stored into *PC* based on *SUB* at push time, the push-time placement module checks the pages in *AC*. If some pages in *AC* have not been referenced since the last replacement in *AC*, these pages are assumed to be less important than the new page and thus become candidates for eviction. The storage of those pages is labeled as belonging to *PC* and is used to store the new page. The placing algorithm of *DC-AP* is as follows:

```
Placing in DC-AP
  Page P is pushed to the server:
    Run SUB on the push cache PC;
    If SUB fails to store P
      S ← pages in AC that have not been accessed since
the last replacement in AC;
      If Size(S) ≥ Size(P)
        While the available storage in PC < Size(P)
          P_m s.t. V_{P_m} = Min{V_i : i ∈ S};
          Label the storage of P_m as PC Label;
        Store P in PC;
end.
```

Recall that *DC-FP* moves a page from *PC* to *AC* when the page in *PC* is accessed for the first time. This "Moving" operation may trigger the replacement in *AC*. To avoid the unnecessary replacements in *AC*, in the same scenario, the locating algorithm in *DC-AP* labels the storage of the page as *AC*, assuming that the pushing frequency in *PC* is relatively low as compared to the replacement frequency in *AC*.

Recall that the Dual-Methods (*DM*) strategy labels each page with two values and considers each value only in the corresponding module. In contrast, *DC-AP* labels each page with a 2-tuple (*o*, *v*) at any time, where *o* indicates the module that should process the page and *v* refers to the page's value under the corresponding operation. Both elements *o* and *v* are updated with time.

In *DC-AP*, the fraction of the storage assigned to each portion can be any value between 0 and 1. If either *PC* or *AC* dominates the storage on a server, pushing or caching consequently dominates the content distribution at that server. To avoid the possible imbalance in the effects of the two modules, a boundary should be set on the fraction of storage that can be assigned to each portion. A variant of *DC-AP* sets the upper boundary and the lower boundary on the fraction that can be *PC* at any given time. We call this variant *Dual-Caches with Limited Adaptive Partition* (*DC-LAP*). In *DC-LAP*, the re-partition in the placing and locating algorithms is performed only when the new partition does not violate the boundary setting.

3.4 Summary of Strategies

The above caching and content delivery approaches can be classified based on when and how content is delivered from the publisher to a proxy server. Table 1 categorizes all the approaches discussed in this paper.

Table 1. Categorization of content distribution schemes

When / How	Access	Subscription	Access + Subscription
Access-time	GD*		
Push-time		SUB	
Access-time + Push-time			SG1, SG2, SR, DM, DC-FP, DC-AP, DC-LAP

As discussed in the literature [27], the performance of the cache replacement algorithms depends highly on the traffic characteristics of accesses. It is worth pointing out that although we use *GD** as the framework in forming our push-time and access-time combined schemes based on subscription as well as usage patterns, our approaches can be also incorporated with other cache replacement algorithms.

4 Workload and Simulator

Because of the difficulty of obtaining meaningful commercial data, the approaches discussed in this paper are validated using a simulator. The simulator is built based on analysis and observations about the real-world data in the literature.

We chose news delivery to be the application scenario for our simulation because of its challenges. News publishing and reading have high temporal dynamics. In addition, news pages are of significant size, especially for multimedia news (e.g., video, audio, and images). Gadde et al. indicate that content distribution is more beneficial when a large number of popular objects have large sizes and high update frequencies [15]. Therefore, news delivery can demonstrate the power of the content distribution strategies that this paper focuses on.

The simulator in figure 2 assumes a single publisher as a news site and a group of proxy servers, each of which connects to a set of users who are close to the server. The input of the simulator includes a publishing stream, a request sequence at each proxy-server, and the subscriptions collected from the end-users at each proxy-server. The publishing and request streams are temporal sequences, while the subscription information is assumed to be static. Our workload is for a 7-day simulation.

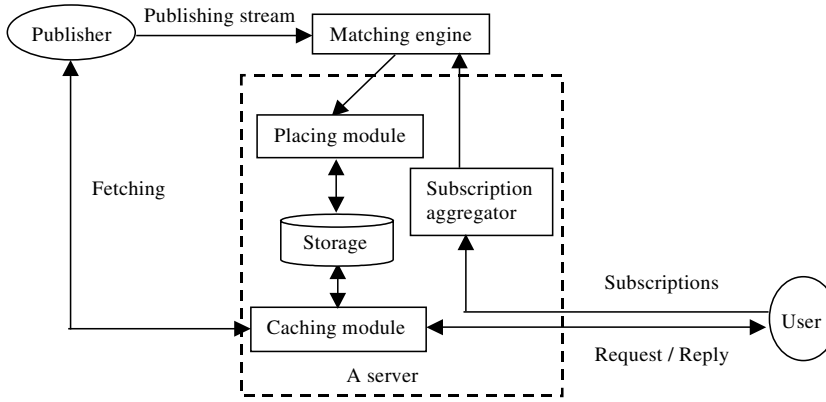


Fig. 2. Architecture of the simulator

As in the models presented in [5, 29], our workload parameterizes the request rate, the document rate of change, the total number of the information objects, and the popularity distribution for objects. As an extension, we model the sizes of pages and simulate a more realistic scenario in which every server has a limited storage capacity. Furthermore, we incorporate subscription distribution that has not been addressed in the literature. Finally, the formulae and the parameters used to build our workload are based mainly on a set of observations and analyses of the content generation and access patterns at the publishing server MSNBC ([23]), rather than on observations at a proxy as in most trace analyses.

4.1 Publishing Stream Generation

As observed in [24], the total number of pages published in 7 days is about 30,000, and about 24,000 pages are modified versions of 2,400 out of 6,000 distinct pages. Our publishing sequence consists of 30,147 pages in total.

In [24], 5% of the modification intervals are less than 1 hour and 5% are greater than 1 day, while others are between 1 hour and 1 day. Based on that, we generate the modification intervals of the 2400 pages using a step-wise random number distribution, assuming a fixed modification interval for any updated page. The first publishing time of the 6000 original pages are randomly chosen from the period (0, 7 days). The generation times of the 24,000 modified versions are then decided based on the modification intervals and the generation times of the first versions.

The sizes of the pages are generated using a log-normal distribution [3][1]. All the pages are assumed to be cacheable in our simulation.

4.2 Request Stream Generation

Scaling Down the Number of Requests. The MSNBC site receives about 25 million requests every day [24], so a 7-day trace should contain about 175 million requests. To scale down the simulation, we consider only 100 proxy-servers as representatives of all the servers sending requests to the publisher from all over the world. In [24], a 5-day trace includes the requests from several hundreds of thousands of institutional domains, hence we assume the 100 servers issue about 1/1000 of all the requests to the site. In this way, the request rate in our trace is scaled down to around 195,000.

The request generator uses Zipf's Law[2] with a homogeneity parameter α of 1.5 to model the popularity distribution of the pages, as observed in [24]. The popularity ranks are randomly assigned to the pages with the assumption that popularity is independent of the publishing time and the size of the page.

Deciding Request Times. The reference times are generated based on the correlation between a page's age and the probability that the page is requested. According to the observations in [24], most news pages are requested when they are fresh, but popular pages are still referenced even if they have been generated for a long time.

The request generator groups the pages into four classes according to their popularity so that the request rate drops about one order of magnitude from one class to the next. For a page in any given class, the probability for the page to be requested at a given time is inversely correlated to the page's age. The more popular a page is, the stronger the negative correlation between the access probability and the page's age is.

---

1 $p(x)=\frac{1}{x\sigma\sqrt{2\pi}}e^{-(\ln x-\mu)^2/2\sigma^2}$, $\mu=9.357; \sigma=1.318$.

2 $R_i = 1/i^{\alpha}$, the request rate on the page with rank i .

Splitting Requests by Server. As observed in the literature [24, 28], the frequently referenced pages are usually accessed by more organizations. As the first step, the server assignment procedure decides the maximum number of servers requesting a given page in a day as a function of the page's popularity using equation 6.

$$S_i = 100 \cdot \left(P_i \Big/ P_{\max} \right)^{0.5}. \tag{6}$$

Where

P_i : the popularity of page i

$P_{\max}$: the maximum popularity of all the pages

For the first day that a page is requested, S_i servers are randomly chosen from the 100 servers to make up a pool of potential servers. After that, every request to that page in that day is randomly assigned to one of the S_i servers.

As observed in [24], the server group requesting a page in one day and that in the next day overlap. Assuming the overlapping ratio is 60%, 40% of the candidate servers for a page in one day are replaced by the servers that are not in the current pool when generating the candidate servers for the page in the second day.

Generating Request Traces with α = 1.0. While news delivery is the focus of our validation study, the performances of our content delivery strategies for more general scenarios are also of interest. As a comparison, another request trace is built using a more popular α value of 1.0 in the Zipf's Law popularity distribution. The trace built using 1.5 as α is called NEWS, and that using 1.0 is called ALTERNATIVE.

4.3 Subscription Generation

Since the subscriptions are static, the only subscription information of interest is the number of subscriptions matching every page at every server.

This paper assumes that the users only request pages based on notification. For a page i, the ratio of the requests to the number of subscriptions matching i at a server j is called *subscription quality*, denoted as $SQ_{i,j}$. When any user reads a page at most once, $SQ_{i,j}$ is the probability for a subscriber of a page to actually request the page. The number of subscriptions matching a page at a server can be inferred using equation 7, given an *estimate subscription quality* (SQ). SQ being 1 is the special case that every user will in fact access any page matching the user's interest. More generally, users may only access a subset of pages that match their stated interest ($SQ < 1$).

$$SF_{i,j} = P_{i,j} \Big/ SQ_{i,j}. \tag{7}$$

Where

$P_{i,j}$: number of requests of page i from server j

$SQ_{i,j}$: a random number in $[2 \cdot SQ - 1, 1]$ if $SQ > 0.5$ or in $[0, 2 \cdot SQ]$ if $SQ \le 0.5$

5 Experiments

5.1 Metrics and Experiment Setup

The motivation of this study is to reduce the response time perceived by the end users. A high hit ratio in a local server generally means a smaller response time hence the *global hit ratio* (*H*) on the 100 servers is the major performance metric as follows:

$$H = \frac{\sum_{i=1}^{100} H_i}{\sum_{i=1}^{100} R_i}. \quad (8)$$

Where

H_i : the number of hits on server i

R_i : the number of request on server i

An elegant content delivery strategy should improve *H* without introducing a big overhead into the network traffic between the publishing site and proxy servers. The networking cost is measured using the amount of contents transferred between the publishers and the servers in terms of the number of pages or the number of bytes.

Our simulation experiments model a quite realistic scenario in which the storage capacity of each cache is limited. The storage capacity of a server's cache is set based on the unique bytes requested at the server in the whole simulation. In the experiments, the performances of the methods are tested under three settings for cache capacity: 1% of the total number of unique bytes requested by a server, 5%, and 10%.

According to experimental results [17], the parameter β (see equation 1) that balances the long-term popularity and the short-term temporal correlation in GD* may be different from trace to trace. On the other hand, when β is learned on-line from the past accesses seen at different times, β is quite stable for a given trace. To decide the suitable value of β, GD* and the two GD*-based approaches SG1 and SG2 are evaluated by varying β from 0.0625 to 4, under three capacity settings for both traces.

In the following experiments, the value of β is set in such a way that the hit ratio of the given algorithm achieves the highest hit ratio. Namely, β is 2 in the three methods for the trace NEWS; for ALTERNATIVE, β is 2 in GD* and SG1 when the capacity setting is 5% or 10% and 1 for 1%, while the value of β is always 0.5 in SG2.

5.2 Comparing Dual-Methods and Dual-Caches

Figure 3 compares Dual-Methods (DM), Dual-Caches with Fixed Partition (DC-FP), Dual-Caches with Adaptive Partition (DC-AP), and Dual-Caches with Limited Adaptive Partition (DC-LAP) for trace NEWS. DC-FP uses a 50%-50% partition. DC-AP starts from a 50%-50% partition but adjusts the partition dynamically. DC-LAP is like DC-AP but bounds the fraction of the pushing cache between 25% and 75%.

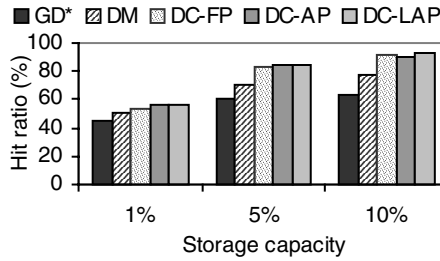


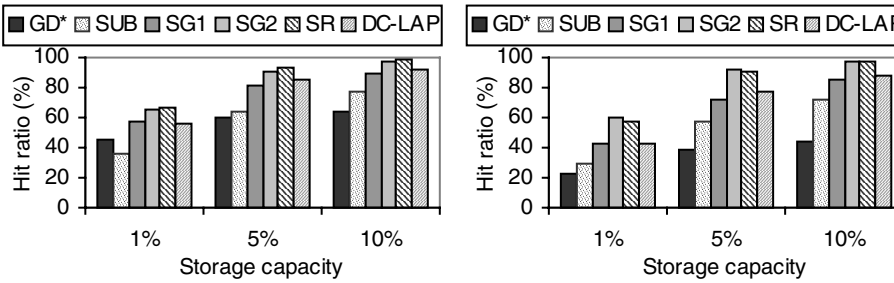
Fig. 3. Hit ratios of Dual-Methods and Dual-Caches algorithms (NEWS)

All the Dual* approaches have better hit ratio than GD*, but DC-LAP outperforms DM and other Dual-Caches approaches in all the cases. The observations hold for the trace ALTERNATIVE and for SQ < 1. Therefore, DC-LAP is chosen as the representative of the Dual* family in the following comparison experiments.

More adaptive approaches DC-AP and DC-LAP only yield marginal improvement over DC-FP. Recall that DC-AP assumes a difference between the publishing rate in the push-cache and the reference rate in the access-cache. Therefore, we conjecture that the little improvement using DC-AP and DC-LAP is due to the high publishing frequency and high re-access frequency in our traces.

5.3 Overall Hit Ratio with Perfect Subscriptions

Figure 4 compares the hit ratios of the major algorithms in this paper in the ideal case that the subscription information perfectly reflects users' request patterns ($SQ = 1$).



(a) For NEWS trace (b) For ALTERNATIVE trace

Fig. 4. Hit ratios of all the methods ($SQ = 1$)

The only case in which any of our new approaches that incorporate subscription-based pushing are worse than the access-based caching-only scheme GD* is when the cache capacity is low (1%). Then GD* outperforms the simple subscription-based pushing-only scheme SUB for the trace NEWS. NEWS has a set of very popular pages and thus exaggerates the performance of the caching-only algorithm.

While the hit ratio increases with the capacity setting for any method, the relative performance ranks of the approaches are quite stable under different capacity settings. All the other new approaches shown in the figure outperform SUB under any setting.

SG2 and SR, which use the estimation of the number of requests of a page in the future, provide the highest hit ratios. The temporal analysis in SG2 does **not** provide extra benefit to SR, which exhibits the difficulty of combining the analyses of history information and of future usage. SG1 has a lower hit ratio than SG2 and SR, which implies the importance to take into account the relation between the subscriptions and the accesses of a page. DC-LAP has a hit ratio similar to SG1, and yields around 4% higher hit ratio than SG1 only when the storage capacity is high (5% or 10%).

When α becomes smaller (ALTERNATIVE trace), the hit ratio of the caching approach GD* is much lower than that when α is high (NEWS trace). The degradation in hit ratio results from a more uniform popularity distribution implying fewer repeated references to the same page. However, the relative improvements using subscription-based pushing-enhanced methods are much higher when α is 1.0 than when α is 1.5, as summarized in table 2. The much higher gains for ALTERNATIVE mean that the push-time placement module benefits the non-homogeneous request streams (characterized by low α) more.

Table 2. Relative improvement over GD* (%) (capacity = 5%)

α	SUB	SG1	SG2	SR	DM	DC-FP	DC-LAP
1.5	6	34	50	54	17	37	40
1.0	47	84	133	133	34	93	96

5.4 Influence of Subscription Quality

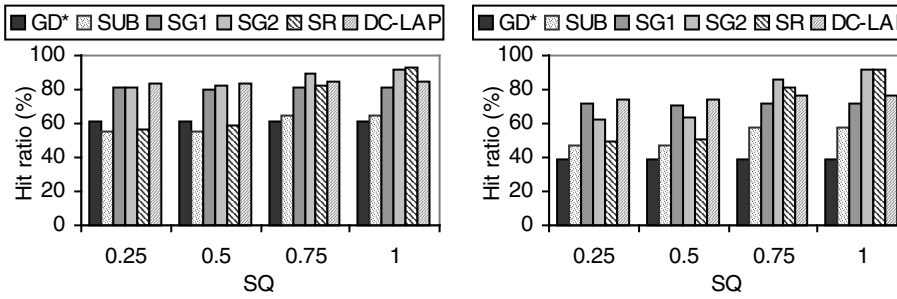
At the 5% capacity level, figure 5 reveals the effect of subscription quality (SQ) that is defined in equation 7 in section 4.3. All the approaches are affected by SQ, except for GD* which does not use the subscription information at all.

SR, which is one of the best approaches in the ideal case, is most affected by SQ and its superiority disappears quickly as SQ decreases. Both SG1 and DC-LAP are not sensitive to SQ, and they are similarly good approaches as SQ decreases. DC-LAP has about 3% higher hit ratio than SG1 when SQ is as low as 0.25.

One major distinction between the results for the two traces is the behavior of SG2. By incorporating the analysis of access patterns, SG2 outperforms SR by remaining highly effective when SQ varies. For NEWS trace, when $SQ \leq 0.5$, whether one is using the sum (SG1) or the difference (SG2) of the number of subscriptions and that of accesses becomes less important because the number of subscriptions dominates the frequency factor in equation 1. For ALTERNATIVE trace, however, the hit ratio of SG2 drops more quickly and it is even worse than SG1 when SQ is 0.25 or 0.5.

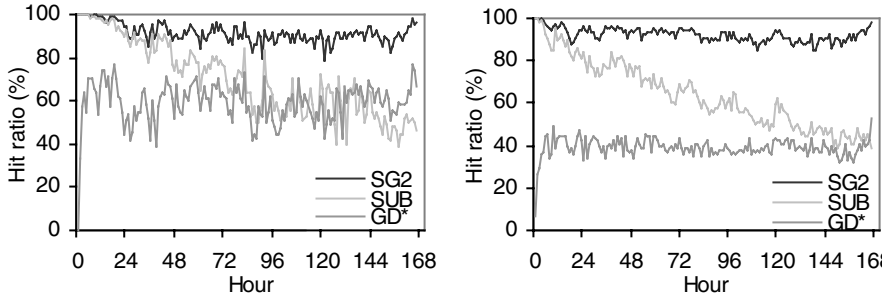
One possible reason for the above distinction is that since the request frequencies of pages are getting more similar with smaller α, the subscription frequency domi-

nates the frequency factor in the evaluation method in SG2. Therefore, the accuracy of subscription information becomes more important for SG2.



(a) For NEWS trace (b) For ALTERNATIVE trace

Fig. 5. Hit ratios of the algorithms with different subscription qualities (capacity = 5%)



(a) For NEWS trace (b) For ALTERNATIVE trace

Fig. 6. Average *H* hourly (SQ = 1, capacity = 5%)

5.5 Hit Ratio versus Time

Figure 6 demonstrates the average *H* of three algorithms in every hour, given that the subscription quality is 1 and the cache capacity setting is 5%. SG2, the best push-time and access-time placement approach in general, is compared against the subscription-based pushing-only method SUB and the access-based caching-only method GD*.

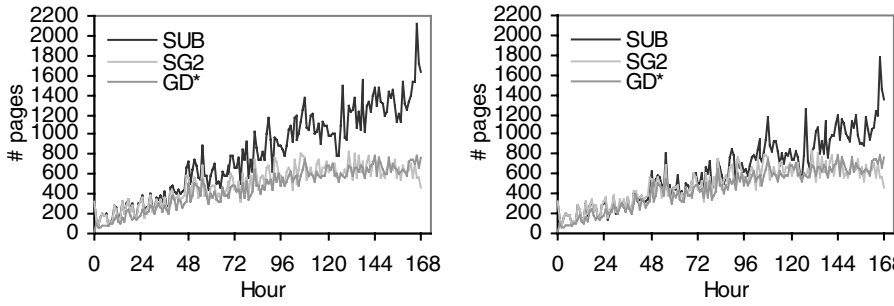
After the first couple of hours, GD* behaves stably. At the beginning, SUB has a high hit ratio by proactively pushing contents before users request them. However, the hit ratio of SUB drops with time because SUB only uses static subscription information but does not adjust the pushing policy according to the usage pattern. SG2 keeps a high hit ratio by combining the subscription and access pattern in placement.

5.6 Traffic Overhead

The push-time module can use either of the following two schemes to push contents:

1. *Always Pushing*: the push-time module always transfers a page to a server when the page is generated and matched to the subscriptions from the server; the server then decides whether to store the page in its local cache based on the replacement algorithm. The bandwidth is wasted if the server decides not to store the page.
2. *Pushing When Necessary*: the push-time module notifies the server of the meta-information such as page size when a page matches the subscriptions from the server; then the server performs its evaluation to decide whether to store the page and sends the result to the push-time module; if the reply is "will store it in cache", the push-time module notifies the publisher to forward the page to the server. This scheme is designed to reduce unnecessary pushing from the publisher to the server.

Figures 7 shows the traffic for pushing pages and fetching pages on cache misses versus time, considering each of the above two pushing schemes. The amount of traffic in terms of the number of pages is measured when using GD*, SUB and SG2.



(a) *Always-Pushing* scheme (b) *Pushing-When-Necessary* scheme

Fig. 7. Traffic in number of pages for two traces (SQ = 1, capacity = 5%, NEWS trace)

The traffic overhead of GD* does not change with pushing scheme hence it can be used as the baseline to compare the two pushing schemes. Interestingly, SG2 is not sensitive to pushing scheme, which implies that SG2 is biased toward new pages. In any case, the traffic overhead of SG2 is comparable to GD*. SUB always introduces the highest traffic overhead, because it suffers from fetching-on-miss due to its low hit ratio. The difference between the curves of SUB and GD* is smaller when using Pushing-When-Necessary than when using Always-Pushing, which means the former benefits SUB a lot in reducing the traffic overhead. The above observations hold for both traces when considering number of pages or number of bytes.

6 Related Work

Siena [8] is a distributed publish/subscribe system that makes use of the coverage relation of messages and subscriptions to achieve scalability. Efficient matching is important to publish/subscribe systems, either in a centralized or a distributed matching scheme [2, 13]. As a complement to topic-based and content-based systems, type-based publish/subscribe enables the integration of middleware and language [11]. The publish/subscribe paradigm can support event notification [14], communication in sensor networks [21], etc. To our knowledge, the storage management problem in content-intensive publish-subscribe services has not been investigated sufficiently.

Web caching passively keeps the most useful information in a capacity-limited proxy server. Many caching replacement algorithms have been presented in the literature. The GreedyDual-Size algorithm combines factors such as temporal locality and popularity as well as fetching cost and page size in caching replacement [7]. GreedyDual* is a generalization of GDS and balances the effects of long-term popularity and short-term reference correlation in a reference stream [17].

Prefetching is used to proactively pull information from an original site to a proxy server [10], or from a proxy to a browser cache [17]. Fan et. al. [17] propose a prefetching mechanism by mining the reference dependency between pages. Browsing agents can pull the pages that link to the current page and/or are similar in contents [9, 22]. Combining access pattern and link structure together, Duchamp proposes a mechanism to prefetch the popular embedded components [10].

Besravros [4] proposes a server-initiated pushing algorithm that places the most popular pages at the layer closer to the end users in a hierarchical caching system. Based on geographical information, Gwertzman and Seltzer [16] present a system that pushes the popular pages to the proxy servers that request the pages frequently.

Caching, prefetching and pushing are mainly based on inferred user interest. In a publish/subscribe application, user preference is stated in their subscriptions and pre-known by the publish/subscribe system. Our work addresses how to exploit the stated user interest as well as the inferred interest in content distribution.

The push-time placement algorithms in this paper belong to replication technique according to the definition in [26]. More importantly, our content delivery is on behalf of the content producers as for content delivery networks (CDN).

Most commercial products and research in CDN focus on hashing-based request redirection to achieve load balance among servers and thus reduce the response time [1, 19]. Gadde et al. [15] indicate a natural limit to the benefits of redirection-based hierarchical CDNs, since the hit ratio in proxy caches increases dramatically as ISPs serve larger user communities. This paper addresses server-based populating that helps to improve the hit ratio even when passive caching achieves its upper limit.

Regarding the placement problem in CDN, the previous work has mainly concentrated on optimum solutions for space-constrained problems. The optimum solution in an overlay network with a graph topology has been proved to be NP-hard, while there exist polynomial solutions for other topologies like trees [20]. The optimum solutions usually assume precise global and stable information known in advance, and thus are infeasible for many web-applications.

Kangasharju et al. [18] propose four heuristics, but the best one needs global knowledge about the network topology, the reference distribution and the content distribution at different times. Qiu et al. [25] propose several heuristics to choose *M* replica sites from *N* candidates for a given site, assuming a relatively stable reference pattern at the candidate sites. For bandwidth-constraint placement, Venkataramani et al. [30] present a solution whose expected response time is within a constant factor of the optimal placement if the information objects have uniform size. However, the algorithm is not designed for a highly dynamic environment in which the object update rate is high and demand-readings proceed in parallel with publishing.

This paper focuses on the coordination between a publisher and a proxy server; hence the placement decision at each proxy server is based on local knowledge only. Therefore, our solutions are suitable for a highly dynamic scenario, which is distinctive from the placement algorithms based on global and static information.

7 Conclusion

We have proposed several content distribution mechanisms for content-intensive publish/subscribe systems. Our approaches combine push-time and access-time content delivery based on subscriptions as well as access patterns. The simulation study demonstrates great improvement in hit ratio by applying our best approaches as compared to the access-based caching method, even if the subscription information does not match requests perfectly. The improvement in hit ratio translates into a reduction in user perceived response time. The traffic introduced by adding the pushing module is not significantly more than that needed to fetch pages on cache misses when using caching only. Our approaches benefit request streams with both regular-popularity and news-based distributions, even benefiting the former more.

Future work is on extending the content delivery schemes to more general scenarios in which not all requests to pages are driven through notification services.

References

1. Akamai. http://www.akamai.com.
2. Altinel, M. and Franklin, M. J. Efficient Filtering of XML Documents for Selective Dissemination of Information. In Proceedings of VLDB 2000, 2000.
3. Barford, P. and Crovella, M. Generating Representative Workloads for Network and Server Performance Evaluation. In Proceedings of ACM Sigmetrics'98, 1998.
4. Besravros, A. Demand-based Document Dissemination to Reduce Traffic and Balance Load. In Proceedings of SPDP'95, 1995.
5. Breslau, L., Cao, P., Li, F., Phillips, G., and Shenker, S. Web Caching and Zipf-like Distributions: Evidence and Implications. In Proceedings of IEEE Infocom '99, 1999.
6. BRITE. http://www.cs.bu.edu/brite/
7. Cao, P. and Irani, S. Cost-Aware WWW Proxy Caching Algorithms. In Proceedings of USENIX Symposium on Internet Technology and Systems, 1997.

8. Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. Design of a Scalable Event Notification Service: Interface and Architecture. Tech. Rep. CU-CS-863-98, Department of Computer Science, Univ. of Colorado at Boulder, Sept. 1998.
9. Chi, E. H., Pirolli, P., Chen, K., and Pitkow, J. Using Information Scent to Model User Information Needs and Actions on the Web. CHI 2001, Vol. 3(1), 490-497.
10. Duchamp, D. Prefetching Hyperlinks. In Proc. of USENIX Symp. on Internet Technologies and Systems, 1999.
11. Eugster, P.T., Guerraoui, R., and Sventek, J. Type-based publish/subscribe. Technical Report, Swiss Federal Institute of Technology, June 2000.
12. Fan, L., Cao, P., Lin, W., and Jacobson, Q. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. SIGMETRICS, 1999.
13. Fabret, F., Jacobsen, H. A., Llirbat, F., Pereira, J., Ross, K. A., and Shasha, D. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In proceedings of SIGMOD, 2001.
14. Fitzpatrick, G., Kaplan, S, Mansfield, T., Arnold, D., and Segall, B. Supporting public availability and accessibility with Elvin: Experiences and Reflections. ACM TOC 11(3): 447 – 474, 2002.
15. Gadde, S., Chase, J., and Rabinovich, M. Web Caching and Content Distribution: A View From the Interior. In Proceedings of WCW '00, 2000.
16. Gwertzman, J. and Seltzer, M. An Analysis of Geographical Push-Caching. 1997.
17. Jin, Shudong and Bestavrous, A. GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams. Computer Comm., vol. 24(2), pp. 174-183, Feb. 2001.
18. Kangasharju, J., Roberts, J., and Ross, K. W. Object Replication Strategies in Content Distribution Networks. In Proceedings of WCW'01, 2001.
19. Karger, D., Lehman, E., Leighton, F. T., Levine, M., Lewin, D., and Panigrahy, R. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In Proceedings of the ACM STOC, 1997.
20. Krishnan, P., Raz, D., and Shavitt, Y. The cache location problem. IEEE/ACM Transactions on Networking, 8(5): pages 568-582, October 2000.
21. Huang, Y.-Q. and Garcia-Molina, H. Publish/Subscribe in a Mobile Environment. MobiDE 01.
22. Lieberman, H. Letizia: An Agent That Assists Web Browsing. Proceedings of the 1995 International Joint Conference on Artificial Intelligent, 1995.
23. MSNBC. http://www.msnbc.com/news
24. Padmanabhan, V. N. and Qiu, L.-L. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In Proceedings of ACM SIGCOMM 2000.
25. Qiu, L., Padmanabham, V. N., and Voelker, G. M. On the placement of web server replicas. In Proceedings of 20th IEEE INFOCOM, 2001.
26. Rabinovich, M. Issues in Web Content Replication.
27. Wang, J. A Survey of Web Caching Schemes for the Internet. ACM Computer Communication Review, 29(5):36-46, October 1999.
28. Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Brown, M., Landray, T., Pinnel, D., Karlin, A., and Levy, H. Organization-Based Analysis of Web-Object Sharing and Caching. In Proceedings of USITS'99, 1999.
29. Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. On the Scale and Performance of Cooperative Web Proxy Caching. In Proc. of SOSP, 1999.
30. Venkataramani, A., Weidmann, P., and Dahlin, M. Bandwidth Constrained Placement in a WAN. In Proceedings of ACM PODC 2001.