PPO-AN APPROACH TO PERSONALIZED WEB ACCELERATION

Shailesh K S [1] and P. V. Suresh[2]

[1]Research Scholar,SOCIS , IGNOU, Maidan Garhi, Delhi - 110068
[2]Director, SOCIS, IGNOU, Maidan Garhi, Delhi - 110068

ABSTRACT

Personalized web is used in all functional domains. Personalized web consists of private pages and pages which personalize the content, data and information based on user's context and preferences. Personalization engines use users' implicit and explicit feedback to personalize the user experience and information. While the personalization drives the user satisfaction levels, it also has performance side effects. Traditional web performance optimization methods fall short of improving the performance of personalized web due to privacy and security concerns and due to the dynamic nature of the data. In this paper we have tried to address this crucial issue by discussing various aspects of personalized performance optimization algorithms. We have discussed a novel approach using "Personalization performance Optimization" (PPO) framework that has resulted in 30% increase in page response times and 35% increase in cache hit ratio during our experiments.

KEYWORDS

Web Performance, Caching strategy, Performance Engineering, Personalized web acceleration Web Performance Optimization

1. INTRODUCTION

Web performance optimization is the key enabler for the business success. As most of the web is personalized, it would be imperative to ensure that personalized web is effectively used to have high impact on end users. As performance is one of the crucial success factors for the user satisfaction, we need to have a strong focus on performance optimization of personalized web.

Bulk of the existing literature covers the performance optimization of public web scenarios as part of Web performance Optimization (WPO). WPO involves best practices and techniques to increase the speed of web pages [1]. It impacts customer churn [2], Site traffic [3] and other factors Performance optimization would be even more challenging for personalized web. For personalized web, content/data varies based on user role and the context. As a result the performance optimization techniques for public web content cannot be effectively used. To address this key challenge we propose a novel Personalized Performance Optimization (PPO) framework and develop the algorithms to automatically pre-fetch the content/asset for personalized scenarios. We will look at the detailed examples of PPO and its constituent algorithms and the results of using PPO on a sample application.

DOI:10.5121/ijcsit.2017.9106

PAPER ORGANIZATION

In this paper we discuss related work in section 2. We then look at main performance optimization techniques and best practices for public web scenarios in section 3. In section 4 we discuss performance optimization of personalized web in greater detail. In this section we would also elaborate personalized performance optimization framework and algorithms for access graph and dependency graph construction along with pre-fetch and caching techniques. Conclusion and results are summarized in section 5. Scope for future work is discussed in section 6.

2. LITERATURE REVIEW AND STATE OF THE ART

The best practices [5] [6] [10] and thumb rules [7] [8] [9] provide good rules for optimizing the web performance. They provide various performance optimization techniques that can be adopted at various layers. For pre-fetching the paper [14] uses file correlation and papers [12,28] pre-fetches using the user's browsing patterns. The papers [16] and [18] uses mining of web logs and paper [19] uses web object lifetime and popularity for pre-fetching. The technique discussed in [24] segregates static and dynamic content for pre-processing and pre-fetching. The paper [25] discusses some of the problems with dynamic and personalized content generation and briefs dynamic content delivery models such as location based cache, object cache. PPM method discussed in [27] builds a history tree to predict target URLs.

Most of the existing techniques pre-fetch the content based on these conditions:

- Create a navigation graph based on access patterns.
- Attach the access probability in the nodes based on the access frequency
- Pre-fetch the resource based on access probability in the graph

The technique is effective for file pre-fetching and web page pre-fetching. For personalized web, most of the techniques use popularity and log analysis as main criteria to predict the resource that needs to be pre-fetched.

An architecture proposed in [22] proposes user management agent at web proxy layer to analyze user profiles and create HTTP response based on it.

EXISTING WPO TECHNIQUES

State of the art techniques focus majorly on public web scenarios. Some of the key web optimization techniques are Asset optimization techniques (minification of web assets, image compression), page compression techniques, asynchronous data loading, caching methods (data caching, content caching, page caching), CDN usage and minimization of HTTP requests (using on-demand/lazy loading)

SCOPE OF IMPROVEMENT TO CURRENT TECHNIQUES

The dynamics of personalization needs more sophisticated pre-fetching and caching techniques in order to make it more effective. Here are the high-level gaps we have noticed with existing techniques with respect to personalized web:

- Pre-fetching technique: Efficiently pre-fetching personalized content and scaling prefetching to high user load
- Caching: Caching personalized content

3. PERSONALIZED PERFORMANCE OPTIMIZATION (PPO)

So far we have seen the state of the art techniques in WPO for public web. The techniques discussed in previous sections would work well for public user scenarios. Web applications are increasingly offering personalized presentations. Personalized web provides most relevant contextual information, content and services based on various factors such as user attributes, demographics, pre-configured user preferences, user's social activities, user transaction history, user access device etc. The main intention of the personalized web is to provide a more engaging experience to the end user with useful content.

Performance optimization techniques for the public user scenario would not be fully useful in personalized web scenarios. The fundamental reason for this being the variance in content from user to user. It would not be possible to pre-fetch or cache all possible variations of content and assets for potentially huge number of users.

Hence in order to address this challenge we will present a novel performance optimization framework for personalized web scenario in this section.

PUBLIC VS PERSONALIZED SCENARIOS

Before presenting the framework, let us look at the main differences between the public and personalized scenarios applicable for performance optimization:

Table 1. Public vs. Personalized Scenarios

Public Web Scenario	Personalized Web Scenario
Data, content and assets are same for all access scenarios **Performance optimization impact:** This can be exploited by caching the content/assets for all users	Data, content and assets would vary based on various personalization factors. **Performance optimization impact:** Variation in content and assets would pose challenges in pre-fetching and efficient caching
Navigation paths would be relatively fixed with pre-determined navigation paths. **Performance optimization impact:** Asset pre-fetching can be smartly designed by easily anticipating the next navigable path	Dynamics of personalized content offer huge combinations of navigation paths **Performance optimization impact:** This would pose challenges in asset and content pre-fetching.
Content in public user scenarios have fixed update frequency **Performance optimization impact:** We could smartly devise caching strategies using cache TTL, cache refresh rate aligned with content update frequency	Dynamic content has varied update frequency without a fixed pattern **Performance optimization impact:** This poses challenges in devising an optimal caching strategy

As we can see from the above table, personalized web offers its own layer of complexity for performance optimization exercise.

PERSONALIZED PERFORMANCE OPTIMIZATION FRAMEWORK

We have proposed a novel performance optimization framework as depicted in figure 1.

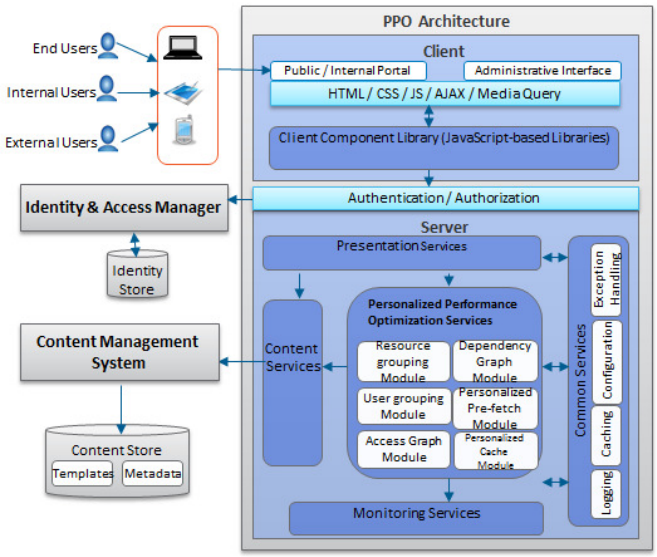

Figure 1: Personalized Performance Optimization Framework

Given below are the key functions of various modules of PPO. We would be discussing the working of each of these modules in greater detail while discussing PPO algorithms:

- **Monitoring services:** Various modules in this category track logged-in user actions. These services are mainly implemented using web analytics software to track user's social activities, page performance, user's navigation patterns and user's transactions. They would feed this information to PPO for building access graphs and for user grouping.
- **Resource grouping module** would group web resources (web content, images) based on similarity of their tagged metadata.
- **User grouping module** would dynamically construct user groups based on their similarity in access paths
- **Access graph module** would construct user access path based on information obtained from monitoring services
- **Dependency graph module** would identify the associated content ids using their tagged metadata
- **Personalized pre-fetch module** would pre-fetch the related web resources using access graph and dependency graph
- **Personalized cache module** would cache various content ids belonging to resource groups.

The PPO framework addresses the challenges we discussed in previous section through various components. Let us look at the way PPO works by looking at a user journey as described in Figure 2.

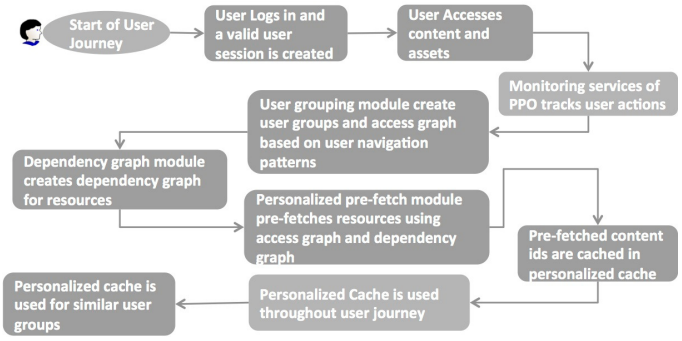

Figure 2: PPO usage in a user journey

When the user logs into a private web page, a valid user session is created. The user session consists of (at a minimal) user profile attributes, user roles and user preferences. User would access the content, data and navigates to various pages as a part of user journey.

As mentioned earlier, static content/asset caching cannot be used for personalized scenarios due to its dynamic nature. To address this, PPO groups users based on their navigation patterns and uses personalized caching to cache content ids. The underlying assumption is that users belong to same user group would be interested in same content and assets. Also the navigation paths used by users belonging to same group would be same. With this assumption it would be possible to create a manageable cache that can be used across various users.

PPO has various monitoring services which tracks and monitors various user actions such as click action, downloads, path traversals, social activity, user transactions and page performance. These monitoring services play a vital role in "understanding" user behavior and the insights are used to cache the information.

Using these monitoring services, PPO builds the map of personalized content/asset usage through these actions:

- User groups and access graph is constructed based on user navigation patterns
- Dependency graph is constructed for various web resources

Using dependency graph and access graph, PPO accelerates the performance of personalized web by pre-fetching and caching the content ids belonging to resource groups.

The information stored in personalized cache would be used throughout user journey. Content ids of various resource groups are also used for other users belonging to similar user group.

EXPERIMENT DESIGN

We have implemented PPO in an application with 30 pages. The web application was created for e-commerce domain with 30 pages spanning across 6 hierarchy levels. Top level pages such as landing page, home page have an average page size of 700KB and average size of other levels are 500KB. There are 5 distinct user groups (buyer, admin, reseller, guest and seller) who can access the application. Each user group has a distinct navigation pattern. For instance a typical buyer would navigate across 6-level page hierarchy as follows: login page, home page, search page, product details page, shopping cart page and check out page. Similarly each user role has a distinct navigation pattern.

Custom monitoring service was developed to track the user navigation on the server side. The monitoring service would track the navigation based on user group. The identified navigation pattern information and dependency graph would be used to pre-fetch the content from subsequent pages cache it in personalized cache for a specific role.

We created five distinct user groups (roles) and each group had 20 users. The navigation behavior and URL/resource access pattern for each user group was varied.

The experiment was conducted on 4 node-cluster with each node running 8GB RAM and 2.4GHZ CPU. In-built cluster-wide cache replication was used for this experiment

It should be noted that pre-fetching and caching would cause slight overhead initially based on the amount of data to be pre-fetched (we are going to look at this overhead in section E).

PPO ALGORITHMS

In this section let us look at various algorithms used in PPO. We mainly discuss 4 algorithms in this section: Access Matrix algorithm which constructs and manages access matrix; dependency graph algorithm to develop content/asset dependency tree; personalized pre-fetch algorithm which is the core algorithm for pre-fetching content ids and personalized cache algorithm for managing personalized cache.

TERMINOLOGIES USED IN ALGORITHMS

Firstly let us look at some of the common terms used in the PPO algorithms:

- Content chunk: It is the content fragment that is rendered on the page section. A content chunk is modular, independent and reusable content piece
- Popularity: Key metrics to determine content and asset popularity are number of views, number of downloads and number of shares. Metrics is mainly obtained from monitoring services
- Metadata: They are the tags which are associated with content and images which provide more information about the content

CONCEPTS USED FOR PRE-FETCHING AND CACHING

We will introduce two main concepts used by PPO algorithms below:

- **Content Ids:** These are the content chunks or content fragments within a web page which form the basic caching unit. In personalized scenario, it makes little sense to cache the entire page as such cache would be least likely be used across different users (due to huge variation in content). Hence we identify the content chunks/fragments, images which stay static across user sessions. When we decompose the personalized page into its constituent sections, we could identify the page sections that would stay common for various users; we term such common static content as content ids. For example in a product details page, we have product brief description chunk, product specifications chunk, product image and product promotion chunk. Out of these 4 content chunks, first three chunks (product brief description chunk, product specifications chunk, product image) would remain same across various users. A special metadata (such as contentIdFlag) would be tagged with the content so that pre-fetch algorithm would use it as marker during pre-fetching content ids for a specific resource group. Content of product promotion chunk based on user loyalty points specific for a given user. Hence first three chunks would be ideal candidates for caching. Besides reusability other main reason for not caching user specific information (such as product promotion chunk) is due to security and privacy reasons.

- **Resource groups:** For optimization purposes we group the contentids into "resource groups". Resource groups are mainly indicated by the URL pattern. For example resource group "products" is represented by URL pattern "*/home/products/*" Products resource group would include all content Ids with this URL pattern such as /home/products/productdetails/main_chunk.html, /home/products/mainimage.jpeg and such. Intuitively resource groups provide a logical grouping of content which would be accessed by users with similar information needs and interests. For example users who access product pages would also access product detail pages. Resource grouping module would map all the content ids for a given resource group offline.

- **User groups:** This represents clusters of users who have similarity in their navigation patterns. User group is determined dynamically. Monitoring services feed the access patterns of various users to PPO and PPO dynamically calculates the similarity and groups the users. The navigation patterns are at resource group level we will discuss this in more details in next section.

Using content ids, we could create a manageable cache in personalized scenario that could be reused across various users. Content ids would also help us achieve balance between performance (achieved through caching) and dynamic nature of content (achieved through partial cache refresh and using non-cached personalized content). Resource groups help us to create an optimized pre-fetch strategy. Creating a second level of abstraction for web resources and users would provide us following benefits:

- Keep the pre-fetching frequency manageable
- Provide efficient caching techniques
- Addresses the problem of content variety and user variety

High level steps in the PPO framework are depicted in Figure 3

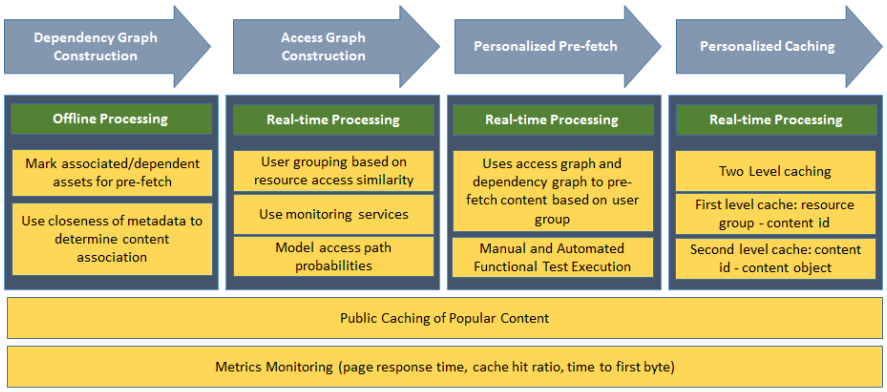

Figure 3: PPO High Level Steps

DEPENDENCY GRAPH CONSTRUCTION

A dependency graph is used mainly for identifying associated/dependent content ids. For instance in product details page, a product thumbnail image has a hard dependency on the full-size product image (as the user is most likely going to see the full-size image when they do a mouse-over or click the product thumbnail image). We can notice here that the dependency relationship is irrespective of the user role (all users are going to view the same thumbnail and full-size image). On similar lines we could establish dependency relationships between product specification content chunk and product specification document.

Dependency graph is mainly constructed based on relationship established through tagged metadata. Unlike the relationship in access matrix (where the access patterns and probability are calculated dynamically), the dependency relationship is static. The metadata tagged with content and assets is used by the algorithm for matching and identifying the dependent content. Degree of metadata matching would be the main criteria for identifying dependency relationship. Table 2 is subset of metadata for product thumbnail image and product full-size image:

Table 2: Metadata for thumbnail and full size image

Metadata Category	**Metadata values for Product Thumbnail Image**	**Metadata values for Product Full-size Image**
Product family	Book	Book
Product Line	Adventure	Adventure
Site page	Products/adventurebooks.html	Products/ adventurebooks.html
Asset Type	Thumbnail Image	Full-size Image

In the above example we can see that the metadata values for “product family”, “product line” and “site page” are exactly same for both product thumbnail image and product full-size image and only the “Asset Type” metadata is different. This results in high degree of matching and the

algorithm creates an association relationship between these two assets. In such scenarios if any one of the associated assets are fetched, other associated assets will be "marked" for pre-fetch.

ACCESS GRAPH CONSTRUCTION

The algorithm for access matrix is a variation of Grioen and Appleton algorithm used for OS files [11]. There are many differences in the way the relationship is established in our algorithm:

- The algorithm uses monitoring services to track user actions and feed the information to server for construction of access graph. This is required in order to understand the user access done on client side.
- Using the access information obtained from the monitoring services, users are grouped using cosine similarity algorithm.
- The algorithm create access matrix consisting of resource path, user cluster, next best resource and next best resource probability

We have seen in earlier sections that one of the key problems in personalization space is that of variety in content and users. It would not be possible to efficiently pre-fetch and cache the data at a user level, as it would result in frequent pre-fetch activity and quickly reaches cache size limitations.

User grouping based on similarity of their access paths: We create user groups using the similarity in their access paths using cosine similarity algorithm. To start with, historical user access data will be used for calculating similarity. User grouping would be continuously updated based on user access thereby improving the efficiency of the algorithm.

Let us see how we group the users with a small example. Table 2 given below are navigation frequency for three users. Number in each cell represents the access frequency to that resource group; for example user 1 had accessed the web resources in account resource group 3 times on an average in the session and accessed resources in product resource group twice on an average.

Table 3. User Navigation frequency based on Resource groups

	Product resource group (/home/products/)	Account resource group (home/accounts/)	Information resource group (/home/about/)
User 1	2	3	1
User 2	1	2	4
User 3	3	3	2
User 4	0	3	3

With this matrix we will use the cosine similarity which is given by formula

$$sim(\vec{a},\vec{b}) = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}| * |\vec{b}|}$$

Similarity is calculated based on angle between the vectors.
Cosine similarity of user 1 and user 2 is calculated as follows

Cosine Similarity (user 1, user 2) = dot(user 1, user 2) / || user 1|| || user 2||

dot(user 1, user 2) = (2)*(1) + (3)*(2) + (1)*(4) = 12

|| user 1|| = sqrt((2)^2 + (3)^2 + (1)^2) = 3.74165738677
|| user 2|| = sqrt((1)^2 + (2)^2 + (4)^2) = 4.58257569496

Cosine Similarity (user 1, user 2) = 12 / (3.74165738677) * (4.58257569496) = 12 / 17.1464281995 = 0.699854212224

Similarly cosine similarity of user 1 and user 3 is 0.9686 and cosine similarity between user 2 and user 4 is 0.9258

Based on cosine similarity we group user 1 and user 3 into one group and user 2 and user 4 into second group.

Access Graph Creation: Once user groups are created we would model the access graph from user group to resource group. Figure 4 provides this representation

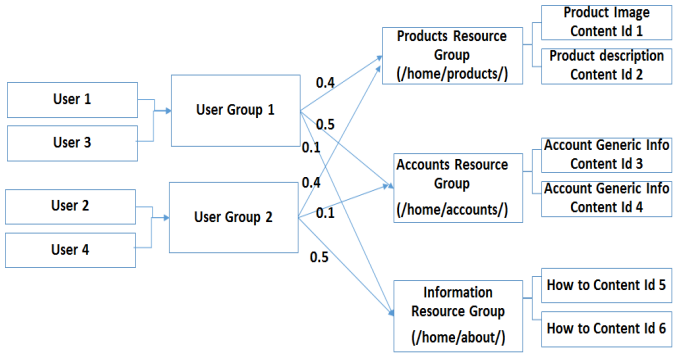

Figure 4: Access Graph based on user group and resource group

The access graph is constructed based on monitoring the navigation patterns of users in various groups. Weights on the edge between user group and resource group in the directed graph indicates the navigation probability. For a given user group, when the user accesses a web resource, monitoring service would record the user access pattern. Monitoring component at the server end would track the access requests within a given time window to determine the probability. Next fetch probability is continuously updated with each user request. Matrix representation of access graph is given in table 4

Table 2. Access Matrix

User group	Next best Resource group	Next-best Probability
User Group 1	Account resource group (/home/accounts/)	0.7
User Group 2	Information resource group (/home/about/	0.3

The matrix indicates that for user group 1, content ids in account resource would be most likely be used (with access probability 70%). Hence the pre-fetching algorithm would pre-fetch and cache the identified content ids within account group

PERSONALIZED PRE-FETCH ALGORITHM

Pre-fetch and caching algorithm is the core algorithm of PPO which anticipates and pre-fetches next-best resource and caches it. The personalized pre-fetch algorithm is given below:

```
void personalized_prefetch(user_id) {
 //If the user group is computed based on
 // earlier access path get the user group
 user_group = get_group_for_user(user_id)

 // For first time users determine user group
 if (user_group == null)
  user_group = compute_group_accesspath(user_id)

 //Get next best resource group from access matrix
 // based on next-best probability
 resource_group = get_next_best_resource_group(user_group)

 //get content ids for resource group in first level cache
 for each content_id in resource_group
  //if content id is not present in second level cache
  //fetch the content and cache it
 if (!cache_exists(content_id))
  populate_content_cache(content_id)

 //pre-fetch and populate dependent content
 for each dependent_content_id
 for content_id
 if (!cache_exists(dependent_content_id))
  populate_content_cache(dependent_content_id)
}
```

The personalized pre-fetch algorithm uses the access matrix and dependency graph to automatically pre-fetch the information at the server end at run time. Pre-requisites for the algorithm are as follows:

- User groups are created based on resource access pattern similarity
- Access graph and access matrix are constructed
- Dependency graph is constructed

HIGH-LEVEL STEPS OF THE ALGORITHM ARE AS FOLLOWS:

1. When the web user logs in, the pre-fetch algorithm determines the user group to which the logged in user belongs. Then it would look up in access matrix for determining the next-best resource group for the corresponding user group
2. If the content ids of the next-best resource group are already present in the personalized cache, then the algorithm exits. If the resource group content ids are not cached, it would pre-fetch the content ids for a given resource group. Content ids could be retrieved from back-end database, content management systems (CMS) or from digital asset management (DAM) systems. All pre-fetched content ids would be cached using multi-level cache. We will look at the cache structure shortly
3. For each of the content id pre-fetched in step 2, the pre-fetch algorithm would also pre-fetch the dependent content ids using dependency graph. Dependent content ids would also be cached in personalized cache.
4. Multi-level personalized cache would be used for subsequent requests.

For all subsequent requests, server renderer would first check in personalized cache before making back end calls.

PERSONALIZED CACHE STRUCTURE

Cache hit is one of the key metrics that determines the success of PPO. Hence cache should be designed to maximize cache hit ratio. Personalized cache is essentially a two-level hierarchical object cache for efficient management of cache operations.

We have used two level cache to efficiently manage the cached content ids:

- First level cache maps the resource groups to content ids
- Second level cache maps the content id to actual content
-

A sample first level and second level cache structure is depicted below:

Table 3 .First Level Cache

Resource group	Content ids
Product resource group	B3ECL93828, B099390176
Accounts resource group	B39DM94428, BIlS87UJ63

Table 4. Second Level Cache

Content Id	Content Object Value
B3ECL93828	<Content Object>
B099390176	<Content Object>

As mentioned earlier no user-specific information would be cached due to security and privacy concerns. A web page is a mix of static and dynamic content. We could take the complete advantage of the static nature of the content through this two-level cache.

Usage of personalized cache: Personalized pre-fetch algorithm determines the user group (based on access pattern) and determines the next-best resource group to be fetched. At this time, all the content ids belonging to the next-best resource group would be retrieved from first level cache and using these content ids actual content objects would be fetched from second level cache.

Popular content caching: Besides content ids belonging to various resource groups, we would also cache the popular public content (such as most viewed public image, most popular public content fragment and such). These public content would again be cached with their content id. These public content ids could be used for public pages (such as user registration page, welcome page, FAQ page and such)

Cache Invalidation: When we say the content as "static", it only means that it is less frequently updated than its dynamic counterpart. In order to ensure that we minimize the stale content, we need to devise appropriate cache invalidation procedures. One naïve approach is to set a constant time out value for each of the cached items (like the way we discussed in public web user scenario). This requires a good understanding of update frequency of the underlying content to achieve good balance between performance and content freshness. Even the best cache time out value would have some amount of freshness lag. Hence PPO uses on-demand cache invalidation. When the content is updated in underlying data sources (such as DBMS, CMS or DAM), the content update activity would trigger a cache invalidation service which removes the cache entry corresponding to the content being updated. This forces PPO to reload the cache with fresh entry.

RESULTS: DEFINITION AND MEASUREMENT OF PERFORMANCE METRICS

For assessing the performance impact of PPO we mainly use three metrics: page response time, time to first byte and cache hit ratio.

Page response time is the overall time taken for rendering the page DOM. Time to first byte (TTFB) is the time taken for first byte of the response to reach user agent (browser). Cache hit ratio indicates the utilization of cache for web requests. We study and compare these metrics with earlier scenarios at various user loads

For analysing and quantifying the impact of PPO, we monitored and measured the web pages that meet following criteria:

- Content of page (web content and static content) is based on the user role. This is to ensure that personalized cache is used for a given user session

- Each page has mix of text content and non-text content. Textual content is mainly web content fetched from content management system (CMS) and static assets (image, video, and document) is retrieved from Digital Asset Management system (DAM). A mix of content types would prove the effectiveness of PPO for various content types.

We loaded the system using Apache JMeter and we took the average value of the metric at various loads. Except for the user load other parameters such as system configuration, data volume and server configuration was kept constant. With all these factors being made constant we take the readings of page response time (total page load time) for the identified pages before and after using PPO.

IMPACT ON PAGE RESPONSE TIME

The main metric captured is the page response time measured by total page load time in seconds. This metric is chosen as it includes DNS lookup time, TCP connection time, time for first byte, full DOM load time and render time and hence gives the complete picture of impact of PPO. Page metrics could be measured through various tools such as Firefox Network monitor, Fiddler and various other browser extensions and plugins.

Each entry in the following table is the average of page load times for the candidate pages and we take average if 3 readings at each user load before and after PPO. We hypothesize that the mean page load times has decreased after PPO and we use paired-t test to prove our hypothesis:

Table 5. Page Load times before and after PPO

User Load	*Average E_A (Before PPO in seconds)*	*Average E_B (After PPO in seconds)*	$\Delta = E_B - E_A$
100	2.75	3.15	0.4
200	2.77	1.45	-1.32
300	1.78	1.52	-0.26
400	1.97	1.9	-0.07
500	2.63	1.76	-0.87
600	2.53	1.38	-1.15
700	2.69	1.27	-1.42
800	2.38	1.5	-0.88
900	2.55	1.34	-1.21
1000	*2.35*	1.8	-0.55
Mean:	**2.44**	**1.707**	**-0.733**
Standard deviations:	**0.331997323**	**0.54847364**	**0.599889805**

We could notice an obvious decrease in the page load time. The average page load time after using PPO is 1.707 seconds as compared to earlier 2.44 seconds which brings about 30% improvement in average page response time. With **mean= -0.733 and mean standard deviation** degree of freedom as 9, we use following formula to get the t-statistic:

t-statistic = [Absolute((□mean - □0) * SQRT(*n*)/*S*□)]

We get t-statistic as 3.863 and we could prove our hypothesis with 99.81% probability.

Note:

1. We noticed that initial page load time has increased slightly (from 2.75 seconds before PPO to 3.15 seconds after PPO). This is due to the time spent in pre-fetching and populating personalized cache for a given user role.
2. For the first time user access, we did not notice significant improvement before and after PPO. This observation can be attributed to the construction of access matrix, user grouping and personalized cache population

IMPACT ON TIME TO FIRST BYTE (TTFB)

TTFB is a measure of efficiency of server side computation. Table VIII is the measure of TTFB:

Table 6: TTFB before and after PPO

User Load	*Before PPO in seconds*	*After PPO in seconds*
100	0.56	0.72
200	0.55	0.38
300	0.36	0.41
400	0.42	0.36
500	0.48	0.33
600	0.51	0.32
700	0.58	0.30
800	0.55	0.39
900	0.52	0.31
1000	0.49	0.42
Mean:	**0.502**	**0.394**

IMPACT ON CACHE HIT RATIO

Cache hit ratio is the ratio of objects found in cache to total cache lookups. It is a measure of cache effectiveness and how well the cache was used for a given application scenario. We compared the cache hit ratio of personalized cache with a regular cache; regular cache was constructed by loading the full page content on demand for specific user with LRU cache invalidation criteria. The usefulness of regular cache was limited only to particular user that too for limited time period. As regular cache had the entire page content, entire cache entry had to invalidate even if one of its constituent elements changed. Figure 5 compares cache hit ratio at various user loads for personalized cache with regular cache

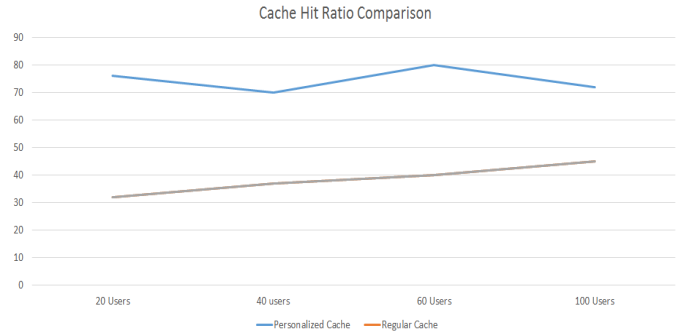

Figure 5: Cache Hit Ratio of personalized cache vs regular cache

We tested the PPO framework at varying user loads and measured the cache hit ratio. We compared the same user load with traditional cache and its cache hit ratio.

The graph clearly indicates a marked improvement in cache hit ratio of personalized cache created by PPO as compared to regular cache. The cache hit ratio for personalized cache is about 75% on average whereas it is about 40% on an average for regular cache. There is an average improvement of 35% in terms of cache hit ratio.

We tested the PPO for maximum 100 users who formed 6 user groups and 5 resource groups. Each resource group cached an average of 100MB of content ids in personalized cache.

COMPARISON OF PPO WITH EXISTING ALGORITHMS

One of the papers [26] discusses the real-time evaluation of prefetching algorithms. Good number of predictive pre-fetch algorithms uses web log analysis to model user's navigation behavior [16] [18]. Other popular pre-fetch algorithm is PPM detailed in [27]. However both log analysis method and PPM are mainly based on access history which is somewhat different from personalization scenarios.

We compared these two methods with PPO using "Cache hit rate" and "overhead rate". Table IX provides the comparison:

Table 7. Second Level Cache

	Cache Hit Rate	Overhead Rate
PPM	10%	23%
PPO	60%	34%

As per the documented analysis, Log-based analysis adds 6% additional overhead for the network traffic.

The main reason for PPO to achieve a high cache hit rate is due to the fact that it factors the personalized user behavior and user group behavior as compared to generic user behavior. Additionally it employs personalized cache for caching the data efficiently

THREATS TO VALIDITY

The hit ratio and overhead rate was calculated for the application with 30 pages. Though 30-page applications reflect a medium-sized real-world application, PPO should be tested on a large web application which typically consists of more than 100 web pages and associated web resources. The scalability, resource utilization and performance of PPO needs to be validated for large web applications. We used 5 user groups with total of 100 distinct users. As the personalized cache is mainly based on user group, we need to test PPO with large user group and a large set of distinct users.

Finally the behaviour of PPO needs to be examined on infrastructure with lower configuration.

4. CONCLUSIONS AND FUTURE WORK

In this research paper, we have proposed a novel personalized performance optimization framework for enhancing the performance of personalized web. As part of PPO we have also presented following algorithms

- Dynamically group the users based on similarity in their access paths
- Develop access path graph and dependency graph for user access path and web resources respectively
- Pre-fetch the relevant and personalized resources using access path graph and dependency graph
- Design of personalized cache for efficiently caching web resources
-

We tested the PPO for personalized scenarios and measured the metrics related to average page response time and cache hit ratio:

- Average page response time improved by 30% on an average for personalized scenarios
- Cache hit ratio was measured against traditional caching and there was an improvement of 35% in cache hit ratio on an average.

REFERENCES

[1] Shivakumar, Shailesh Kumar. (2014). Architecting High Performing, Scalable and Available Enterprise Web Applications. Morgan Kaufmann.
[2] For Impatient Web Users, an Eye Blink Is Just Too Long to Wait: http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html?_r=2
[3] Akamai Report: http://www.akamai.com/html/about/press/releases/2009/press_091409.html
[4] Speed Is A Killer – Why Decreasing Page Load Time Can Drastically Increase Conversions: http://blog.kissmetrics.com/speed-is-a-killer/.
[5] S. Souders – Even Faster Web Sites: Performance Best Practices for Web Developers; O'Reilly Media, 2009
[6] S. Souders – High Performance Web Sites: Essential Knowledge for Front-End Engineers; O'Reilly Media, 2007
[7] Best Practices for Speeding Up Your Web Site: http://developer.yahoo.com/performance/rules.html
[8] Web Performance Best Practices: http://code.google.com/speed/page-speed/docs/rules_intro.html

[9] WPO – Web Performance Optimization: http://www.stevesouders.com/blog/2010/05/07/wpo-web-performance-optimization/
[10] S. Stefanov– Web Performance Daybook; O'Reilly Media, 2012
[11] James Grioen and Randy Appleton. Reducing File System Latency using a Predictive Approach", Proceedings of the 1994 Summer USENIX Technical Conference, Cambridge MA, June, 1994.
[12] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World Wide Web latency. ACM SIGCOMM Computer Communication Review, 1996
[13] Gu, Peng; Wang, Jun; Zhu, Yifeng; Jiang, Hong; and Shang, Pengju, "A Novel Weighted-Graph-Based Grouping Algorithm for Metadata Prefetching" (2010). CSE Journal Articles. Paper 44.
[14] D. Kotz and C.S. Ellis, "Practical Prefetching Techniques for Multiprocessor File Systems," J. Distributed and Parallel Databases vol. 1, no. 1, pp. 33-51, Jan. 1993
[15] H. Lei and D. Duchamp, "An Analytical Approach to File Prefetching," Proc. USENIX Ann. Technical Conf., Jan. 1997
[16] Lee, H., An, B., & Kim, E. (n.d.). Adaptive Prefetching Scheme Using Web Log Mining in Cluster-Based Web Systems. 2009 IEEE International Conference on Web Services.
[17] Dahlan, A., & Nishimura, T. (n.d.). Implementation of asynchronous predictive fetch to improve the performance of Ajax-enabled web applications. Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services - IiWAS '08.
[18] Yang, Q., Zhang, H., & Li, T. (n.d.). Mining web logs for prediction models in WWW caching and prefetching. Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '01.
[19] Venkataramani, A., Yalagandula, P., Kokku, R., Sharif, S., & Dahlin, M. (n.d.). The potential costs and benefits of long-term prefetching for content distribution. Computer Communications, 367-375.
[20] Bouras, C., Konidaris, A., & Kostoulas, D. (n.d.). Predictive Prefetching on the Web and Its Potential Impact in the Wide Area. World Wide Web, 143-179.
[21] Xu, C., & Ibrahim, T. (n.d.). Towards semantics-based prefetching to reduce Web access latency. 2003 Symposium on Applications and the Internet, 2003. Proceedings.
[22] Frelechoux, L., & Kamba, T. (1997). An architecture to support personalized Web applications. Computer Networks and ISDN Systems, 29. Chicago
[23] Azarias Reda, Edward Cutrell, and Brian Noble. 2011. Towards improved web acceleration: leveraging the personal web. In Proceedings of the 5th ACM workshop on Networked systems for developing regions (NSDR '11). ACM, New York, NY, USA, 57-62.
[24] Douglis, F., Haro, A. and Rabinovich, M., 1997, December. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In USENIX Symposium on Internet Technologies and Systems (pp. 83-94).
[25] Ravi J, et al. A survey on dynamic Web content generation and delivery techniques. J Network Comput Appl (2009)
[26] B. de la Ossa, J. A. Gil, J. Sahuquillo, and A. Pont. 2007. Web prefetch performance evaluation in a real environment. In Proceedings of the 4th international IFIP/ACM Latin American conference on Networking (LANC '07). ACM, New York, NY, USA, 65-73
[27] T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. Proc. of the 4th International Web Caching Workshop, San Diego, USA, 1999.
[28] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. In ACM SIGMETRICS, 1999.