

MCWDF: Micro Chunk Based Web Delivery Framework

Shailesh Shivakumar, IGNOU, New Delhi, India

Venkata Suresh Pachigolla, IGNOU, New Delhi, India

ABSTRACT

Segregating the web page content into logical chunks is one of the popular techniques for modular organization of web page. While chunk-based approach works well for public web scenarios, in case of mobile-first personalization cases, chunking strategy would not be as effective for performance optimization due to dynamic nature of the Web content and due to the nature of content granularity. In this paper, the authors propose a novel framework Micro chunk based Web Delivery Framework which proposes and uses a novel concept of “micro chunk”. The micro chunk based Web Delivery framework aims to address the performance challenges posed by regular chunk in a personalized web scenario. The authors will look at the methods for creating micro chunk and they will discuss the advantages of micro chunk when compared to a regular chunk for a personalized mobile web scenario. They have created a prototype application implementing the Micro chunk based Web Delivery Framework and benchmarked it against a regular personalized web application to quantify the performance improvements achieved by micro chunk design.

KEYWORDS

Content Chunk, Micro Chunk, Personalization, Personalized Web Acceleration, Web Content, Web Performance, Web Performance Optimization

INTRODUCTION

As web is becoming a greater influence on the community, there are various initiatives to enhance the web experience. Web pages are now made highly responsive, interactive and personalized and it is available on all mobile devices. Modern web architecture needs to address multiple concerns and cater to various stakeholders. Internet-facing web sites should be responsive, interactive and should achieve optimal performance (Galletta et al., 2004). The success of online strategy depends on its usability and performance (Schmiedl et al., 2009). The web application should satisfy quality attributes such as performance, scalability, extensibility and flexibility (Shivakumar, 2014).

Web pages that display enterprise information would mainly get its web content from content management systems (CMS). We could broadly categorize such web pages as public web which renders web pages for public users and private web which renders the web content for personalized/ logged-in scenarios. There are number of techniques such as layered architecture, caching, service-oriented-architecture, asynchronous resource loading, content compression that can be used to achieve better scalability and performance (Souders, 2009).

DOI: 10.4018/IJITWE.2018010101

Web content chunking (also known as HTML fragmenting) is one of the popular techniques wherein we logically segregate the page content into multiple chunks (or fragments) (Griffin et al., 2005). The chunks are modular logical entities which are independent and have reuse potential. A content chunk is a logic piece of cohesive HTML content on a web page which can exist independently and describes a semantic concept (Challenger et al., 2005).

For instance in a products web page, the product brief description information could constitute a chunk and product specifications information could form another chunk. A chunk consists of rich content and multi-media content. A chunk for the web content will be created from its origin system which is normally a content management system (CMS). The content chunk created will be tagged with metadata so that it can be appropriately retrieved on the correct web page. Chunk-based web page architecture addresses concerns such as scalability, performance and reusability (Shivakumar, 2014).

While content chunking works as an effective strategy for public web scenarios, personalized web poses its own set of challenges. In a personalized web scenario, a chunk contains the content applicable for a given context (such as user attributes, preferences, device, and language, location, and security roles and such). This means that it is not possible to statically author these content chunks in CMS and cache it globally. This further impacts the scalability and performance attributes. Web sites are adopting hyper personalization strategy wherein all content is heavily personalized for the user and the context. The trend is most noticed in e-commerce sites such as Amazon.com, Flipkart.com wherein the logged-in user would see personalized product recommendations, personalized offers, personalized notifications and such personalized content. In such scenarios there will be seldom static content chunks which can be applied to all users posing challenges to the page performance.

One of the ways to effectively achieve good performance in personalized scenario is to identify the static chunks for a personalized page and globally cache it. This partially addresses the problem as the dynamic and personalized chunks still cannot be cached and they have to be authored thereby impacting the overall page performance.

In this paper we introduce a novel concept called “micro chunk” and micro chunk based Web Delivery Framework (MCWDF) which addresses the challenges in authoring, management and performance for personalized web scenario. A micro chunk is highly granular dynamic content chunk which is created and managed at the run-time with minimal overhead on page performance. Micro chunk is created mainly based on context parameters such as user attributes, device name, security roles and other filters. As micro chunk caters to dynamic content scenarios, it can be effectively used in personalized web pages with minimal overhead in content authoring.

The objective of this paper is to elaborate the novel concepts related to micro chunk and MCWDF and how it solves the dynamic content challenge in personalization scenarios.

Based on our experiments we were able to notice 90% improvement in page response time by usage of micro chunk when compared to regular content chunk for personalized mobile web scenarios.

PAPER ORGANIZATION

In the remaining portions of the introduction section we will look at the necessity and novelty and motivation of micro chunk, core concepts of micro chunk along with related work and the significance of the work. We will discuss the complete details of the micro chunk, micro chunk based web delivery framework, caching design and the authoring and publishing aspects of micro chunk in the “Method” section. In “Results” section we will look at the benchmarked results of performance numbers at various user load and content metrics. Finally we will discuss the significance of results, explanation of the main findings, and impact on content authoring and future scope of improvements in “discussion” section.

CASE FOR MICRO CHUNK

Most of the modern web applications are personalized. Almost every content chunk on a personalized page has a personalized and contextual information. This brings in two set of differences from the regular content chunk/HTML fragment based approaches: The chunk as a whole is not dynamic; it is partially dynamic and partially static. For instance consider the content chunk:

Your policy is due for renewal on 01-Jan-2017 and your new annual premium would be \$250.

The pieces (*01-Jan-2017* and *\$250*) in the above content fragment are dynamic and the rest is static. As a result of this mixed nature of the chunk it is not possible to cache it fully nor can we afford to refresh a complete page for updating this content. The dynamic nature of the content also poses challenges for authoring these types of chunks. We see a good number of these granular content chunks in modern mobile-first web sites.

CORE CONCEPTS OF MICRO CHUNK BASED WEB DELIVERY FRAMEWORK (MCWDF)

The key concepts using in MCWDF is detailed below:

- **Content Chunk:** A content chunk (also known as HTML fragment) is a logical piece of cohesive HTML content on a web page which can exist independently and describes a semantic concept;
- **Micro Chunk:** A micro chunk is a modular and dynamic component of a content chunk. Micro chunk is constructed at the run time at the micro chunk server based on context parameters and uses light-weight services to retrieve the dynamic values;
- **Context Parameters:** These parameters represent the variables which can fully describe the run time environment. Session attributes, user attributes, user preferences, device, security roles, current page are some of the examples of context parameters;
- **Micro Chunk Server:** A micro server is responsible for constructing the micro chunk based on context parameters. The micro chunk server would manage the micro chunk through its entire lifecycle including authoring, updating, caching and purging the micro chunk. The micro chunk server also manages all the service invocations needed to manage the micro chunk and acts as a facade for the web page;
- **Micro Chunk Service:** The light-weight services which can be used to obtain the micro chunks and they are hosted on micro chunk server. Micro chunk services would take the context parameters to create the micro chunk.

RELATED WORK

The core concepts discussed in the paper are related to "...chunking content in dynamic scenarios..." and "...improving performance and management of dynamic content in personalization scenarios..." We will look at state of the art in these two focus areas.

Firstly, the generic concept of chunking is used for web page construction for many purposes.

The concept of chunk is used to reduce network traffic (Zhang et al., 2015), and to reduce download time in mobile devices (Jang et al., 2014). Chunked data is also widely used in CDN (Content Delivery Network) for performance optimization (Zhu et al., 2016). Griffin et al. (2005) uses chunk concept for identifying and removing duplicates and Davis et al. (2006) uses chunking method for device specific rendering and appropriate positioning. Brodie et al. (2004) uses keywords for chunk detection. Maghoul et al. (2008) uses chunking concept to differentiate static chunk from

dynamic chunk and pre-fetch appropriately. ESI (Edge side include) provides language for defining web page components and assembling it at edge caches using chunking concept.

Content fragments are also used to handle dynamic content (Khaing & Thein 2005). Ramaswamy et al., 2004 proposes an algorithm to automatically detect the fragments based on its caching and reusability potential. Christos et al., 2004 and Ioannis et al., 2004 propose user driven personalization which uses fragments based on page's internal structure. Challenger et al. (2005) and Challenger et al. (2000) provide an efficient update of dynamic content in a web page and dynamic content publishing using fragments. Chan & Woo (1999) efficiently encode structurally similar web pages of a site using content chunks.

As far as dynamic content performance is concerned, papers (Challenger et al., 1999) and (Challenger et al., 2000) discuss dynamic content caching and Cao et al., 1998 uses active cache for caching dynamic web content. Candan et al. (2002) discusses invalidation algorithms of dynamic chunks and exploits the difference between cache invalidation and view maintenance to delivery fresh content. Datta et al. (2002) proposes fragment based performance improvement. Mohapatra & Chen (2001) uses web graphs using fragment for managing quality of service in dynamic content.

Once the chunks are categorized as static and dynamic, caching policies can be accordingly applied. The web graph technique discussed in Mohapatra & Chen (2001) handles dynamic content by adding various attributes such as QoS (quality of service), security and these attributes are used to refresh the dynamic content.

Bruck et al. (2012) uses micro content for efficient learning on mobile devices and the paper (Nagler et al., 2007) discusses micro content for e-learning. In these two techniques, content is broken down into smaller and easily-digestible learning content.

As papers (Ramaswamy et al., 2004) and (Challenger et al., 2005) are closely related to the subject matter of this paper, we will elaborate the differences between the topics discussed in those papers and our proposal. The core idea of the paper (Ramaswamy et al., 2004) is the automatic fragment detection based on its reuse potential and its lifetime and personalization characteristics and caching it. Challenger et al. (2005) presents technique to establish relationship between pages and its composite fragments represented by object dependence graph. It also presents algorithm to detect and update changed fragments and publish them. The content fragments discussed in these papers are at coarse grained and the updates are done at the chunk level. Based on the amount of content present within the fragments, the web page incurs data transfer overhead. In the proposed micro chunk based approach, we propose a light-weight fine-grained content where content updates are highly manageable with least data transfer overhead. Papers (Ramaswamy et al., 2004) and (Challenger et al., 2005) would also encounter scalability challenges when the web pages need to handle huge amount of private and personalized content scenarios as it leads to high frequency content updates whereas our micro chunk based approach is tested for scalability in high traffic personalized scenarios. The content fragments of (Ramaswamy et al., 2004) and (Challenger et al., 2005) are also not designed to handle a mix of static and dynamic values within a single content fragment. In the case of a micro chunk, we could efficiently handle the dynamic values embedded within a single micro chunk.

To summarize, state of the art techniques use the content fragments to handle dynamic content using following methods:

1. Identifying the reusable chunks and segregating static chunks from dynamic chunks;
2. Use dynamic fragments to automatically push the updated content to page through dynamic fragments;
3. Design the cache differently for static and dynamic chunks.

KEY MOTIVATIONS FOR MICRO CHUNK

The concept of content chunk and fragment needed a fresh look in the context of hyper-personalized modern applications which are developed with mobile-first strategy. The traditional definitions of

content chunks or fragments (Zhang et al., 2015), (Zhu et al., 2016) would not be fully relevant for a personalized modern application. Firstly, the amount of content chunk has reduced to cater to mobile and Omni-channel requirements. Secondly and more importantly the content fragment would no longer fit into a pure definition of a “static chunk” or “dynamic chunk” (Maghoul et al., 2008); it is mix of static content along with dynamic values. So, we need a granular content fragment which could cater to dynamic and personalized scenarios and the concept of micro chunk is designed for this purpose. While the traditional content chunks were pre-dominantly driven by CMS or HTML content, the content in modern web applications get the information from various sources. So, micro chunk server would use micro chunk assembler and invoke all necessary services to create the micro chunk.

The cache management of micro chunk would also be different from that of regular content chunks (Challenger et al., 1999), (Challenger et al., 2000), (Mohapatra & Chen, 2001). Hence micro chunk cache uses two-level caching mechanism with Micro chunk update monitor doing an on-demand cache invalidation when the source content changes.

The main gaps with the state of the art techniques with respect to personalized mobile-first scenario are as follows:

1. In a personalized mobile-first web scenario, most of the content chunks would contain a mix of static and dynamic data and it would not be easy to neatly separate static content from dynamic content;
2. Personalized data is more dynamic than a regular dynamic data. A personalized data is specific for an individual user. We could explain the difference between the regular dynamic data and personalized data through a dashboard example. A public dashboard of an employee portal would render the regular dynamic data by retrieving the web content from CMS; whenever the content in CMS changes the dashboard would get the updated dynamic data from the CMS. A personalized dashboard rendered for a logged-in employee would specifically display the content for the employee such as employee’s leave balance, profile details and such. This personalized dashboard is a classic example of personalized data. State of the art techniques are designed to manage and cache regular dynamic data but falls short of handling the personalized data. None of the caching techniques mentioned earlier can efficiently handle the personalized data due to security and privacy concerns. State of the art caching techniques discussed earlier would only cache the fully static data in the global cache. If the dynamic data (such as CMS content or database records) are used across various user sessions, they will be stored in the cache (Challenger et al., 2000). However, the dynamic data applicable for the users (personalized data) will not be cached by the state of the art caching systems;
3. State of the art techniques also do not automatically consider context parameters such as user’s device, time, location and such to render the personalized content chunk;
4. Due to the challenges with applicability of content chunk with personalized data, the performance and scalability would be impacted due to the increased run-time overhead in retrieving and processing the dynamic content. The impact would be more with high volume of dynamic data and during peak user traffic as the server would take more time in processing and rendering the personalized content without the help of cache (Shivakumar, 2014).

Our novel concept of micro chunk is designed to handle all types of personalized and context parameters for optimal page rendering.

In the next section, we will briefly discuss the drawbacks of content chunks in the context of personalized page scenarios.

COMPARISON WITH ALTERNATE TECHNIQUES

The main techniques which are closer to MCWDF are regular content chunk and custom services aggregation. Let us look at the key differences between Micro chunk and these two techniques.

Table 1 provides high level differences between a normal content chunk and a micro chunk.

The functions of a micro chunk server can loosely be compared to that of a custom services layer. Micro chunk server that creates the micro chunks does more activities than a services layer such as assembling, caching and managing a micro chunk. The micro chunk server exposes micro services to the presentation components and aggregates services needed for developing a micro chunk.

SCOPE AND SIGNIFICANCE OF THE WORK

The scope of this research work covers the following:

- Performance optimization of a mobile-first web page in personalized scenarios using micro chunk concept. This improves upon the existing content chunk based technique to handle dynamic content scenario;
- Improving the scalability for the personalized and dynamic web content scenarios.

The research paper proposes a novel micro chunk based web delivery framework which can be used in personalized web and dynamic web scenarios.

Method

In this section, we will look at the detailed design of micro chunk based web delivery framework (MCWDF) along with caching and authoring scenarios for micro chunk.

ANATOMY OF A MICRO CHUNK

Let us look at the concept of micro chunk using an insurance dashboard page we have developed as a demo application. Figure 1 shows the micro chunks on a personalized insurance dashboard which is developed for mobile web.

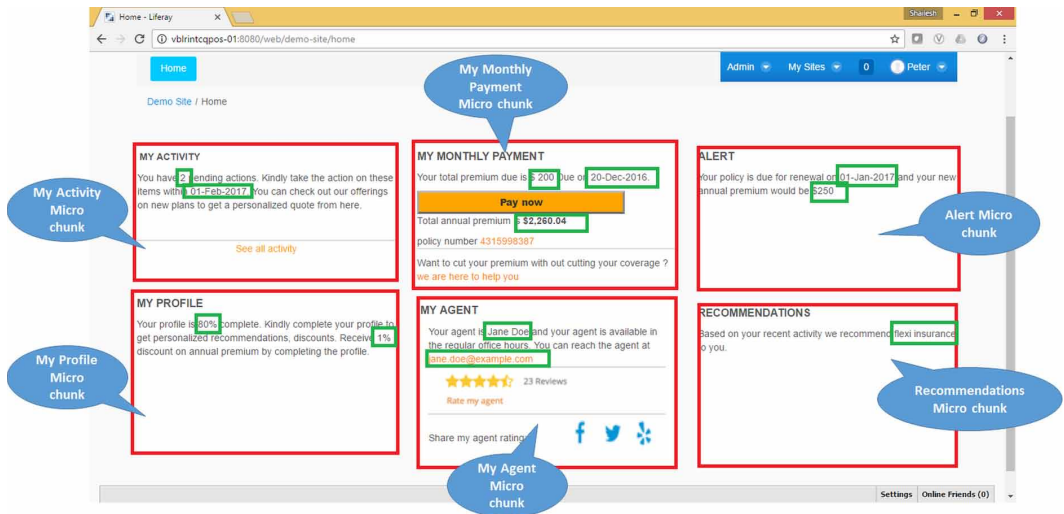
As we can see the personalized dashboard has a heavy mix of static content and dynamic values. It would not be possible to cache the content as-is due to privacy and security concerns and dynamic nature of the data. We can also notice that there is not much content chunk which are fully static. Almost all content has personalized information to some extent.

A micro chunk can be created to handle a variety of dynamic personalized information which depends on user information and context information.

Table 1. Content chunk vs Micro chunk

	Content Chunk	Micro Chunk
Key Tenets	<ul style="list-style-type: none">• Cohesiveness• Semantic independence and reusability• Static nature	<ul style="list-style-type: none">• Modular• Dynamic and context based content
Content Granularity	Relatively coarse grained	Fine-grained granularity
Data Content	Normally static which can be used across all contexts	A mix of static and dynamic and driven by context parameters
Reuse potential	Can be fully reused as-is in static scenarios and public pages Cannot be reused in private web pages which need personalized data	Can be reused across public and private scenarios

Figure 1. Micro chunks of mobile enabled personalized dashboard page



The key tenets of a micro chunk are: Independence (nil or minimal dependency on other content), Dynamic nature (should contain dynamic and personalized information), Reusability (the content within a micro chunk should be reusable), shorter content length (content sentences ranging from 1-5) and Modularity (logical grouping of content).

In the dashboard example, we have identified 6 micro chunks and the dynamic values within each micro chunk is highlighted within a box. Each micro chunk identified satisfies 5 key tenets listed above. For instance, the “Alert micro chunk” contains the logical grouping of policy alert information and is both modular and independent. The micro chunk contains the dynamic information 01-Jan-2017 and \$250 which is personalized for logged-in user Peter. The alert message verbiage and format can be reused across all users with varying dynamic values applicable for each user. The micro chunk can be effectively used for enabling the mobile web and implement mobile first strategy. The rendition of the responsive dashboard page on a mobile device is shown in Figure 2.

In the next section, we will discuss in detail about the MCWDF and the role of micro chunk in rendering the personalized page.

MICRO CHUNK BASED WEB DELIVERY FRAMEWORK ARCHITECTURE

We are proposing MCWDF that uses micro chunk to efficiently manage the content on mobile-first personalized pages. MCWDF is responsible for creating the micro chunk, its associated JSON and manages its lifecycle along with the caching of micro chunk.

The logical architecture of MCWDF is depicted in Figure 3.

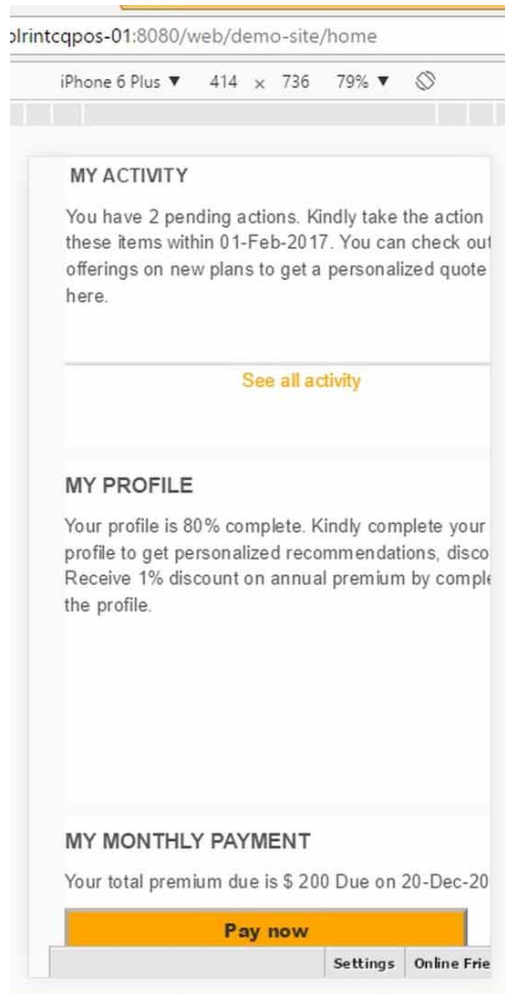
The MCWDF consists of components that are part of web page and components which are part of micro chunk server.

Web Page Level MCWDF Components

There are basically three framework components that are a part of web page.

Micro chunk JSON (JavaScript Object Notation) defines a micro chunk. The micro chunk JSON will be loaded as part of initial page load. Each micro chunk is identified by a uniqueid and contains the details of a micro chunk such as content type (text or image or video), serviceurl (that provides the service end point to the micro chunk service based on context parameters). The context parameters

Figure 2. Mobile simulation of the dashboard page



are essentially personalization tokens which provide the system-specific and application-specific context values such as user roles, device, account id and such. The micro chunk preprocessor would send these values to the micro chunk service. Let us look at two micro-chunks in JSON format to understand its design and utility in Figure 4.

Each micro chunk has a name is identified by a “*uniqueid*” which is a unique identifier. The “*contenttype*” indicates the type of the content rendered by the micro chunk and “*serviceurl*” represents the service end point which provides the needed dynamic information for this micro chunk. We can also pass all context parameters such as userid, device, page and such.

The *contextparams* could be “generic” which contain non-user specific details such as page name, device id or language. “private” context parameter provides user specific personalization tokens such as user id, account id and such.

In Figure 4, “user_promotion_message” micro chunk has a uniqueid of 12345 and invokes the “getpromomsg” service URL.

HTML tags is basically a HTML div element with a pre-defined style class “*mcwdf-microchunk*”. The HTML tags provides the place holder for injecting the dynamic micro chunk at the run time. The HTML DIV embeds the name of the micro chunk. The micro chunk preprocessor is responsible for

Figure 3. Micro chunk based web delivery Framework

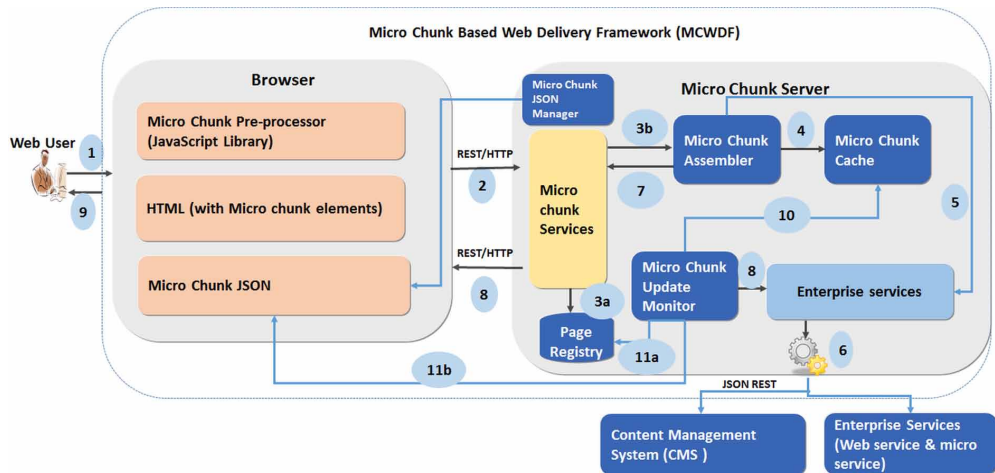


Figure 4. Micro chunk JSON

```
{
  "microchunks" : [
    {
      "name": "user_promotion_message",
      "uniqueid": "12345",
      "contenttype": "text",
      "serviceurl": "http://localhost:9091/getpromomsg",
      "contextparams": {
        "generic": {
          "page": "{page_id}"
        },
        "private": {
          "userid": "{user_name}",
          "account": "{account_id}"
        }
      }
    },
    {
      "name": "user_account_info",
      "uniqueid": "672345",
      "contenttype": "image",
      "serviceurl": "http://localhost:9091/getacctinfo",
      "contextparams": {
        "generic": {
          "device": "{device_id}",
          "lang": "{lang_id}"
        },
        "private": {
          "userid": "{user_name}",
          "account": "{account_id}"
        }
      }
    }
  ]
}
```

populating the div with the dynamic micro chunk value obtained from the micro chunk service. The update happens asynchronously and whenever there is an update to the micro chunk. An example of the HTML is depicted in Figure 5.

In Figure 5, the div identified by style class “*mcwdf-microchunk*” embeds {{ user_promotion_message }}. The place holder {{ user_promotion_message }} will be replaced by the actual dynamic content value at the run time by micro chunk preprocessor.

Micro chunk preprocessor is a custom JavaScript library that uses the micro chunk JSON to invoke the corresponding micro chunk services and injects the dynamic micro chunk content obtained from the micro chunk service. The micro chunk would be inserted at the HTML element identified by the micro chunk name or micro chunk uniqueid as depicted in Figure 5.

Given below are the process steps that would occur to embed the micro chunk on the web page:

Step 1: Micro chunk preprocessor would scan the entire web page to list all the micro chunk HTML tags identified by style class “*mcwdf-microchunk*”. As depicted in Figure 5, it would find a micro chunk HTML tag with {{ user_promotion_message }} placeholder.

Step 2: Micro chunk preprocessor would then look for the details of the user_promotion_message micro chunk in the micro chunk JSON. From Figure 4, it would identify the “getpromomsg” as the service URL for this micro chunk.

Step 3: Micro chunk preprocessor would invoke the “getpromomsg” service URL passing all the needed generic (page id = dashboard) and private (user id = Peter and account = 1234) values. Upon successful invocation of the “getpromomsg” service, the micro chunk preprocessor would receive the micro chunk. The obtained content for the micro chunk is “Congratulations Peter! You can avail \$200 discount on your next purchase.”

Step 4: The micro chunk preprocessor would inject the obtained micro chunk into the {{ user_promotion_message }}. The HTML snippet after the micro chunk injection is given in Box 1.

Box 1.

```
<div class="borderless-container" style="">
  <div class="body">
    <div class="journal-content-article">
      <div class="mcwdf-microchunk" >
        Congratulations Peter! You can avail $200
discount on your next purchase.
      </div>
    </div>
  </div>
</div>
```

Figure 5. HTML for rendering micro chunk

```
<div class="borderless-container" style="">
  <div class="body">
    <div class="journal-content-article">
      <div class="mcwdf-microchunk" >
        {{user_promotion_message}}
      </div>
    </div>
  </div>
</div>
```

The micro chunk rendition algorithm is given below in Algorithm 1.

Algorithm 1

```
void render_microchunk(pageid) {  
    //Micro chunk preprocessor would scan the entire web page for  
    // mcwdf-microchunk tags and compiles the list  
    List microchunk_tag_list = getMicroChunksOnPage(pageid);  
    //Process all the micro chunks on the page  
    for each microchunk_tag in microchunk_tag_list do  
        //Extract the micro chunk name from the microchunk_tag  
        Microchunk_name = get_microchunk_name(microchunk_tag);  
        // look up the micro chunk JSON to get the micro chunk service  
        for  
            // the micro chunk name  
            Microchunk_service = get_microchunk_service(microchunk_name);  
            //Get other parameters needed to invoke the micro chunk service  
            Microchunk_params = get_microchunk_params(microchunk_name);  
            // Invoke the micro chunk service with all needed parameter  
            values.  
            Micro_chunk = assemble_microchunk (Microchunk_service,  
            Microchunk_params)  
            //Inject the micro chunk into the corresponding HTML tag  
            Update_HTML (Microchunk_name, Micro_chunk);  
        end do;  
    }  
}
```

Server Level MCWDF Components

The key components of the micro chunk server are as follows.

Micro chunk services which act as the service end point for the micro chunk requests. Once the request for micro chunk is received along with context parameters, the requesting page and its context values will be added to the page registry and the request is transferred to the micro chunk assembler. If the page id is already present in the page registry the status flag will be checked to get the content from cache.

Once it receives the micro chunk from the micro chunk assembler, it would create the response for the same. For textual micro chunk, the textual micro chunk content will be embedded within the micro chunk whereas for an image micro chunk the URL of the image would be sent as part of the response.

Micro chunk assembler is responsible for identifying all the upstream services needed to create the requested dynamic micro chunk. Initially the assembler checks for the existence of micro chunk in the micro chunk cache using the micro chunk's uniqueid as the cache key. If the corresponding micro chunk is not cached, the micro chunk assembler invokes corresponding services from enterprise services layer and creates the dynamic micro chunk. A single micro chunk may need invocation of multiple services. For creation of "user_account_info" micro chunk the assembler would invoke 2 services: firstly a content service to get the base static content of the micro chunk which gets the content: "Your policy is due for renewal on #due_date and your new annual premium would be #due_amount".

Then the assembler invokes two micro services to get the dynamic values for #due_date and #due_amount for the said customer.

Only the base static version of the micro chunk is cached against the micro chunk's uniqueid and the context parameters.

Micro chunk assembler handles both textual micro chunk and non-textual micro chunk (such as images). All micro chunk would be stored in the micro chunk cache.

Enterprise services would interact with enterprise interfaces such as content management system (CMS) and other upstream enterprise services. Enterprise services layer could also invoke micro services needed for the micro chunk.

Page registry contains the mapping of the micro chunk ids and the page URLs on which those micro chunks are used.

Micro chunk update monitor registers for the updates from CMS and enterprise services. Each of the system of records (such as CMS, enterprise services) would provide a registration interface which would be used by the micro chunk update monitor. When the data changes in the system of record, the system of record would notify the micro chunk update monitor of the change and the update monitor would purge the micro chunk cache for the corresponding micro chunk and would update the page registry. If the content is updated subsequent requests for the micro chunk on those pages would go to the source systems for the content.

Micro chunk JSON manager is mainly responsible for creating the initial version of the micro chunk JSON that is used to by the micro chunk preprocessor of the page. The micro chunk JSON manager provides a web interface for the administrator to configure various parameters of a micro chunk such as uniqueid, serviceurl, contextparams and others.

The micro chunk assembly algorithm is given below in Algorithm 2.

Algorithm 2

```
void assemble_microchunk (Microchunk_service, Microchunk_params) {  
    //Add or update the page registry  
    Add_update_page_registry(Microchunk_params);  
    // initialize micro chunk  
    Micro_chunk = null;  
    //Start assembling all the needed params for the micro chunk  
  
    // check for existence of cache for the corresponding micro chunk  
    If (check_microchunk_cache(Microchunk_params)) != null) {  
        Micro_chunk = get_from_microchunk_cache (Microchunk_params);  
        Return micro_chunk;  
    }  
    // Invoke all necessary services for creating the micro chunk  
    Micro_chunk = invoke_services (Microchunk_params);  
    //the service will constantly monitor all the systems and will  
    invalidate  
    // micro chunk cache if the source data changes.  
    Monitor_for_micro_chunkchanges();  
}
```

At a broad level, micro chunk server essentially plays the role of a facade to invoke needed services and manage the micro chunk. An intermediate content services layer could serve this purpose; however micro chunk server is a sophisticated platform that can manage the micro chunk's lifecycle more efficiently through following mechanisms:

- The Micro chunk JSON manager of the micro chunk server creates the initial version of the micro chunk JSON for each page. This is the initial stage of the micro chunk flow wherein we define all the parameters needed for a micro chunk such as private parameters and generic parameters. The

micro chunk JSON will be used by the micro chunk preprocessor to invoke the corresponding service and populate the micro chunk on the page at the run time;

- The execution of the micro chunk is handled by micro chunk services and micro service assembler. These components will internally use micro chunk cache and all necessary enterprise services to create the micro chunk for a given context;
- The micro chunk update monitor will continuously monitor the source systems (CMS, enterprise services) for any data change and flushes the micro chunk cache and page register once the data changes. This ensures that the page always get the fresh micro chunk data.

CACHING DESIGN

The micro chunk cache is a two-level cache. First level stores the mapping of micro chunk's uniqueid to a hash value computed using the generic context params (such as device id and lang id). The hash value is computed using all the "generic" context parameters for a micro chunk. The cache entries for "user_account_info" is given in Table 2.

The <object> in the second level cache would be the actual micro chunk content identified by uniqueid 672345: "Your policy is due for renewal on #due_date and your new annual premium would be #due_amount".

MICRO CHUNK AUTHORIZING AND PUBLISHING

A separate authoring template would be created for managing micro chunks. As micro-chunks are modular and reusable across many pages, authors would be able to manage micro chunks centrally. As part of micro chunk creation process, author needs to specify the micro chunk name with other metadata tags such as language, device, role and other applicable personalization parameters. There could be multiple variants of the same micro chunk varying based on language, device and user role. Authors can add the static text of the micro chunk and indicate the dynamic variables. Here is the sample authoring for "user_account_info" micro chunk:

Your policy is due for renewal on #due_date and your new annual premium would be #due_amount".

Dynamic values #due_date and #due_amount are resolved at run time by the micro chunk assembler based on the context parameters.

The micro chunk is published as a JSON which can be invoked through REST web service by providing the uniqueid and context parameters.

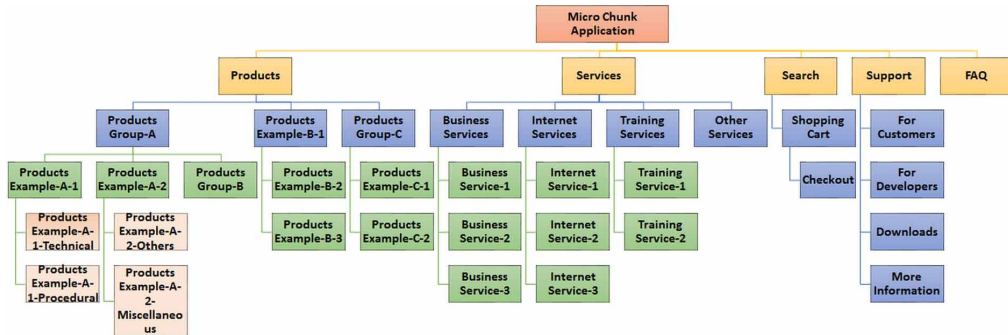
RESULT

We implemented the MCWDF in a prototype application, *Micro chunk Web App*, with 40 pages with 3 hierarchy levels whose hierarchy level is shown in Figure 6. The dashboard page depicted in Figure 1 is the home page of the Micro chunk web app. We created micro services to get the dynamic data needed for the micro chunk. The micro chunk server in the Micro chunk Web App leveraged the micro

Table 2. Two-level micro chunk cache

Cache Key	Cache Value
672345	m6e428e5n346cew78bfd5a005477521
m6e428e5n346cew78bfd5a005477521	<object>

Figure 6. The 3-level hierarchy of micro chunk web app



services and each micro chunk used an average of 1.4 micro service to create a fully dynamic micro chunk content. Each page has an average 4.5 micro chunk in the micro chunk web app.

EXPERIMENT DESIGN

For measuring the performance improvement, we created another web application named as *Dashboard web app*. In this case we used database services and content services directly invoked from the page to create the dynamic data on the web page. The content required for dashboard web app was authored as regular content chunk with dynamic values being added at the run time by the presentation engine. Table 3 provides the differences between the dashboard web app and micro chunk app.

We have maintained the page count of 40 for both micro chunk web app and dashboard web app same so that we can benchmark against similar content load.

PERFORMANCE COMPARISON OF MICRO CHUNK WEB APP VS DASHBOARD WEB APP

We used Apache JMeter to load both the applications and captured the average response times. The performance metrics at the load of 100 concurrent users is given in Table 4.

Table 3. Micro chunk web app vs Dashboard web app

Category	Micro Chunk Web App	Dashboard Web App
Page content assemble	Through micro chunk using MCWDF	Through direct invocation of web services and content services
Content authoring	Authored as micro chunk	Regular content chunk
Page count	40	40

Table 4. Performance metrics for micro chunk app and dashboard app

Application	Average First Byte Time (S)	Average Page Load Time (S)	Average Total Number of Requests per Page	Average Asset Load Time (Image, CSS, JS) (S)
Micro chunk web app	0.125	1.088	4	1.039
Dashboard web app	1.142	1.999	20	1.849

We took the average times for pages to arrive at the numbers shown in Table 4.

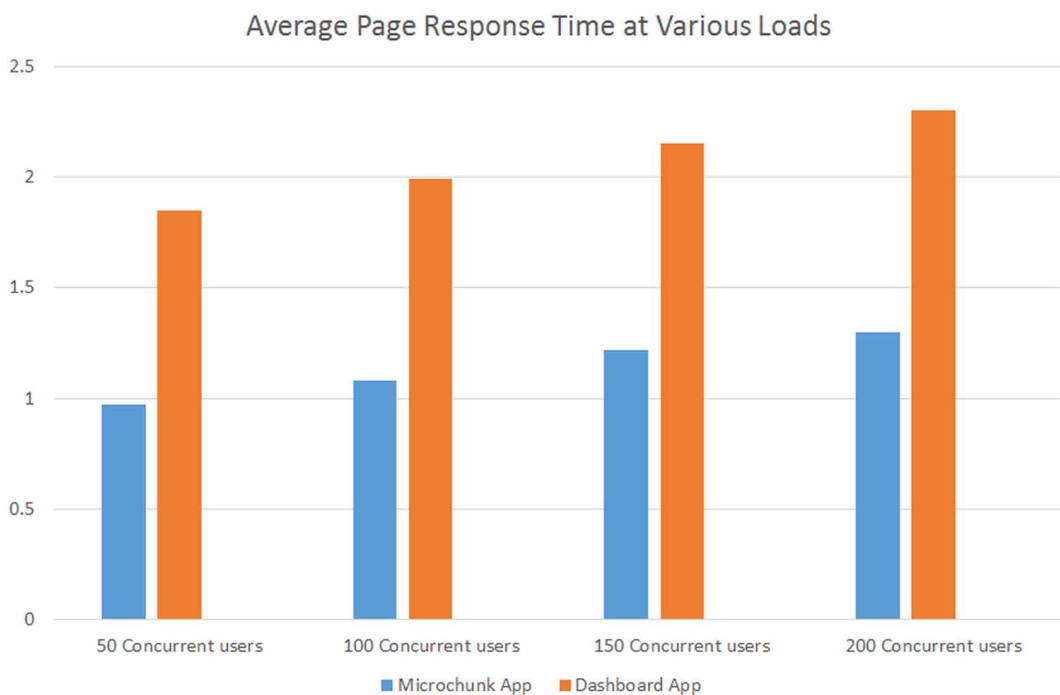
We measured the average page response time (PRT) at various loads as depicted in the Figure 7. We consistently noticed an average of 90% improvement in page response time at various loads.

We also measured the content authoring time needed for the micro chunks of micro chunk web app and content chunk of dashboard web app. Here are the main metrics:

- Total number of content templates needed:
 - Micro chunk templates in micro chunk app: 15;
 - content chunk templates in dashboard app: 10;
- Average Content authoring time:
 - Average time for authoring a Micro chunk in micro chunk app: 15 minutes;
 - Average time for authoring a content chunk in dashboard app: 10 minutes;
- Average content publishing time:
 - Micro chunks in micro chunk app: 5 minutes;
 - content chunks in dashboard app: 5 minutes.

There is 50% increase in the number of templates needed for micro chunk app as compared to regular content chunk templates needed for dashboard web app. The increase in the template number for micro chunk app can be mainly attributed to dynamic values and its placement which creates variations among templates. The authoring time for micro chunk is also higher when compared to dashboard app for the same reason.

Figure 7. Average response times



DISCUSSION

We will discuss the significance of the results and the effectiveness of micro chunk in this section.

EXPLANATION OF MAIN FINDINGS

As we can see from Table 4 and Figure 7, the micro chunk web app has a distinct performance advantage in mainly 3 areas: average page load time, average time to first byte (TTFB) and average resource requests per page. As the micro chunks are created, managed and cached by the micro chunk server, the average page response time has improved by more than 90% and the average asset load time has improved by 80%. We could also notice that average number of resource requests of micro chunk app has reduced to 25% of the dashboard web app. This is mainly due to the design of micro chunk server which acts as the facade to manage and handle all the underlying micro services. The asset load time reduction is due to asynchronous asset loading process.

We could also notice that on content management side, the number of micro chunk templates and content chunk authoring time is lagging behind that of content chunk. While the micro chunk template development is a one-time process which can be reused for future micro chunks, the increase in the micro chunk authoring effort can be attributed to the complexity involved in the authoring process. The author needs to select the right micro chunk template and use the correct dynamic values.

SIGNIFICANCE AND IMPLICATIONS OF THE RESULTS

The key finding from MCWDF is the drastic improvement of web performance for personalized mobile-first site in page load time and asset load time metrics. This is a significant improvement over traditional ways of dynamic information aggregation. This makes the micro chunk a promising design alternative for personalized mobile web applications. As more and more web applications are built with mobile-first design, the concept of micro chunk becomes more relevant.

Another aspect of MCWDF is about the handling of personalized and dynamic values. The MCWDF also handles the scenario which has partially mixed dynamic values within static content chunk. Micro chunk server provides an efficient management mechanism for creating, updating and caching micro chunks.

Impact on Content Authoring and Publishing

On the content authoring front, additional micro chunk templates need to be created and authors need to be trained to use those templates. For the micro chunk web app we created 15 new micro chunk templates to create all distinct micro chunk each with an average of 2 dynamic values.

Development of additional micro chunk templates and author training was the overhead involved in this effort. Authors were also confused about the definition and usage of dynamic values at times.

For content publishing we leverage the in-built JSON services of the CMS to expose the micro chunk content as a web service. There was no impact on content publishing side.

THREATS TO VALIDITY

We conducted the experiments for 40 pages with maximum of 200 concurrent users load. The performance improvement was observed in this setup. We would need to observe the performance improvement of MCWDF with higher number of pages and with greater user load to check the scalability and performance of the micro chunk cache.

The performance improvement is also largely dependent on the frequency of content updates in the source systems. If there is a frequent content update activity, it would trigger cache invalidation

leading to cache misses; this in turn would negatively impact the page performance. This also would impact the performance metrics of the MCWDF.

FUTURE SCOPE OF IMPROVEMENTS

Currently the micro chunk JSON is mainly developed manually in micro chunk JSON manger interface. The process of initial micro chunk JSON creation process needs to be automated. More research needs to be done in that front.

The authoring process of micro chunk needs to be improved. Currently the authors mainly depend on the training material and this could lead to confusion. A more structured and easy-to-use process needs to be designed. Efforts are needed to reduce the content authoring time and to design an efficient way to manage micro chunk templates. Automatic conversion of existing web content to micro chunk is also one of the improvements that can be done.

CONCLUSION

We have proposed a novel framework MCWDF which can be used for performance optimization of personalized mobile web pages. The MCWDF uses the concept of micro chunk to logically partition the page into modular, dynamic and reusable micro chunks. Micro chunk server would be responsible for creating, updating and caching the micro chunks for the page.

We developed a demo site which implemented the MCWDF and benchmarked the performance against a regular web application of similar worldload. As per our experiments we have noticed > 90% improvement in page performance.

REFERENCES

- Brodie, D., Gupta, A., & Shi, W. (2004). Keyword-based fragment detection for dynamic web content delivery. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters - WWW Alt. '04*. doi:10.1145/1013367.1013444
- Bruck, P.A., Motiwala, L., & Foerster, F. (2012). Mobile learning with micro-content: a framework and evaluation. In *Proceedings of the 25th Bled eConference Dependability: Reliable and Trustworthy eStructures, eProcesses, eOperations and eServices for the Future Bled eConference* (pp. 17-20).
- Candan, K. S., Agrawal, D., Li, W.-S., Po, O., & Hsiung, W.-P. (2002) View Invalidation for Dynamic Content Caching in Multi-tiered Architectures. In *Proceedings of VLDB '02*.
- Cao, P., Zhang, J., & Beach, K. (1998). Active Cache: Caching Dynamic Contents (Objects) on the Web. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, the Lake District, UK. doi:10.1007/978-1-4471-1283-9_23
- Challenger, J., Dantzig, P., Iyengar, A., & Witting, K. (2005). A fragment-based approach for efficiently creating dynamic web content. *ACM Trans. Inter. Tech. TOIT ACM Transactions on Internet Technology*, 5(2), 359–389. doi:10.1145/1064340.1064343
- Challenger, J., Iyengar, A., & Dantzig, P. (1999). A Scalable System for Consistently Caching Dynamic Web Data. *Proceedings - IEEE INFOCOM*, 99.
- Challenger, J., Iyengar, A., Witting, K., Ferstat, C., & Reed, P. (2000). Publishing System for Efficiently Creating Dynamic Web Content. *Proceedings of IEEE INFOCOM '00*.
- Challenger, J., Iyengar, A., Witting, K., Ferstat, C., & Reed, P. (2000). A publishing system for efficiently creating dynamic Web content. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. doi:10.1109/INFCOM.2000.832259
- Challenger, J., Iyengar, A., Witting, K., Ferstat, C., & Reed, P. (2000). A publishing system for efficiently creating dynamic web content. In *Proceedings of IEEE INFOCOM*.
- Chan, M. C., & Woo, T. W. C. (1999). Cache-Based Compaction: A New Technique for Optimizing Web Transfer. In *Proceedings of INFOCOM '99*.
- Christos, B., Vaggelis, K., & Ioannis, M. (2004). Web page fragmentation for personalized portal construction. In *Proceedings of the International Conference on Information Technology: Coding and Computing ITCC '04*. doi:10.1109/ITCC.2004.1286475
- Datta, K. Dutta, H. Thomas, D. VanderMeer, Suresha, and K. Ramamritham. (2002) Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In *Proceedings of SIGMOD '02*.
- Davis, P.E., Dean, S.E., Meliksetian, D.S., Milton, J., Weitzman, L., & Zhou, N. (2006). U.S. Patent No. 7,076,728. (2006). U.S. Patent and Trademark Office, Washington, DC.
- Douglis, F., Haro, A., & Rabinovich, M. 1997, December. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (pp. 83-94).
- Galletta, D. F., Henry, R., McCoy, S., & Polak, P. (2004). Web site delays: How tolerant are users? *Journal of the Association for Information Systems*, 5(1), 1.
- Griffin, W., Jones, B., & Lee, S. K. (2005). U.S. Patent No. 11/192,791. U.S. Patent and Trademark Office, Washington, DC.
- Ioannis, M., Vaggelis, K., & Christos, B. (2004). Web Page Fragmentation and Content Manipulation for Constructing Personalized Portals. In *Advanced Web Technologies and Applications, LNCS* (pp. 744-754). doi:10.1007/978-3-540-24655-8_81
- Jang, I., Suh, D., & Pack, S. (2014). Minimizing content download time in mobile collaborative community. In *Proceedings of the IEEE International Conference on Communications (ICC)*. doi:10.1109/ICC.2014.6883697

- Khaing, A., & Thein, N. L. (2005). Efficiently Creating Dynamic Web Content: A Fragment Based Approach. In *Proceedings of the 6th Asia-Pacific Symposium on Information and Telecommunication Technologies*. doi:10.1109/APSITT.2005.203648
- Maghoul, F., Yiu, P., Davis, M., Athsani, A., & Yi, J. (2008). U.S. Patent No. 12/240,323. U.S. Patent and Trademark Office, Washington, DC.
- Mohapatra, P., & Chen, H. (2001). A Framework for Managing QoS and Improving Performance of Dynamic Web Content. In *Proceedings of GLOBECOM '01*. doi:10.1109/GLOCOM.2001.966219
- Naaman, M., Garcia-Molina, H., & Paepcke, A. (2003). Evaluation of ESI and Class-Based Delta Encoding. In *Proceedings of WCW '20*.
- Nagler, W., Ebner, M., & Sherbakov, N. (2007). Flexible teaching with structured micro-content-How to structure content for sustainable multiple usage with recombinable character. In *Proceedings of the Conference ICL2007*, September 26-28. Kassel University Press.
- Ramaswamy, L., Iyengar, A., Liu, L., & Douglass, F. (2004). Automatic detection of fragments in dynamically generated web pages. In *Proceedings of the 13th Conference on World Wide Web WWW '04*. doi:10.1145/988672.988732
- Schmiedl, G., Seidl, M., & Temper, K. (2009, September). Mobile phone web browsing: a study on usage and usability of the mobile web. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services* (p. 70). ACM
- Shivakumar, S. K. (2014). *Architecting High Performing, Scalable and Available Enterprise Web Applications*. Morgan Kaufmann.
- Souders, S. (2009). *Even Faster Web Sites: Performance Best Practices for Web Developers*. O'Reilly Media.
- Zhang, X., Wang, N., Vassilakis, V. G., & Howarth, M. P. (2015). A distributed in-network caching scheme for P2P-like content chunk delivery. *Computer Networks*.
- Zhu, M., Li, D., Wang, F., Li, A., Ramakrishnan, K. K., Liu, Y., & Liu, X. (2016). CCDN: Content-Centric Data Center Networks. *IEEE/ACM Transactions on Networking*.

Shailesh Shivakumar received his BE in Computer Science & Engineering from the Bangalore University in 2001. He is currently a research scholar in IGNOU, India. His research interests include web engineering, performance engineering, portal technologies and web performance optimization.