JAVASCRIPT PERSISTENCE OBJECTS WITH ADAPTIVE SYNCHRONIZATION TIMING

Supervised by: Dr. Shehan Perera

Prepared by: T.A.M.P Fernando (138212T)

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2016

JAVASCRIPT PERSISTENCE OBJECTS WITH ADAPTIVE SYNCHRONIZATION TIMING

Supervised by: Dr. Shehan Perera

Prepared by: T.A.M.P Fernando (138212T)

Thesis submitted in partial fulfillment of the requirements for the Degree of MSc in Computer Science specializing in Software Architecture

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2016

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: ……………… Date: ……………….

Name: T.A.M.P Fernando

The above candidate has carried out research for the Masters dissertation under my supervision.

Signature of the supervisor: …………………………. Date: ……………….

Name: Dr. Shehan Perera

ABSTRACT

Traditionally, web applications were limited to server generated, dynamic HTML content. With the advancements of web technologies such as HTML5, AJAX and JavaScript Engines, web browsers evolved into application platforms. These have changed the traditional web application architectures, enabling significant part of the web application, runs inside the web browser. Most widely accepted architecture for these web applications uses JavaScript frameworks, that runs inside the web browser, which communicates using the HTTP protocol with RESTful web services, to persist the application state.

There are specialized RESTful web services offered by cloud providers, which are directly capable of persisting objects sent by the web applications. These services provide high scalability, durability and availability of the objects. However, above capabilities comes with a cost, where partial updates of these objects are not possible. To persist dynamically changing objects requires an entire object to be sent to the persistence web service for each modification. When the frequency of dynamically changing object becomes higher, it increases the object persistence time, due to network latency, congestion and persistence web service throttling. Longer response times for persisting objects may cause lower satisfaction and poor productivity among web application users.

This problem can be solved by persisting only the last modification of the object, within a time window. However, having a fix time window, also increases the persistence time for non-frequent modifications.

The aim of this research is to implement a JavaScript framework, which is capable of providing an adaptive persistence time window, to reduce unexpected object persistence times for frequently changing objects by reducing the network congestion and object persistence web service usage, enhancing overall web application user satisfaction.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my profound gratitude to my advisor, Dr. Shehan Perera, for his invaluable support throughout the research project by providing relevant knowledge, supervision and useful suggestions. His expertise and continuous guidance enabled me to complete my work successfully. Further, I would like to thank Dr. Chandana Gamage for providing valuable resources and advice and insight of the applicability of this research in its initial phases.

I am also grateful for the support and advice given by Dr. MalakaWalpola, by encouraging continuing this research till the end. Further, I would like to thank all my colleagues for their help on finding relevant research material, sharing knowledge and experience and for their encouragement.

I am deeply grateful to my parents for their love and support throughout my life. I also wish to thank my loving wife, who supported me throughout my work. Finally, I wish to express my gratitude to all my colleagues at 99XTechnology, for the support given me to manage my MSc research work.

TABLE OF CONTENTS

LIST OF FIGURES

ABBRIVATIONS

API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
BOM	Browser Object Model
CSS	Cascading Style Sheets
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
REST	Representational State Transfer
SQL	Structured Query Language
WORA	Write Once Run Anywhere
WORE	Write Once Run Everywhere
W3C	World Wide Web Consortium
GPRS	General Packet Radio Service

CHAPTER 1

INTRODUCTION

Traditionally, web applications were limited to server generated, dynamic HTML content. User interactions with these applications were primitive and allowed to navigate through hyperlinks and filling forms. Additional capabilities such as video streaming were provided by web browser capability extension technologies such as Flash, Java Applets & Silverlight. With the advancements of web technologies such as HTML5, AJAX and Powerful JavaScript Engines, web browsers became much capable of providing rich user interactions. This made web browsers become application platforms [1], that allows to provide capabilities on par with desktop applications. With the wide adoption of W3C standards, by web browser vendors, differences between web browser platforms were minimized. This has largely contributed to the tremendous growth of web applications during the recent few years.

These have changed the traditional web application architectures, enabling significant part of the web application, runs inside the web browser written using JavaScript, CSS & HTML5. With the growing complexity of rich client applications, specialized engineering skills were required, utilizing software engineering patterns and principles to keep the complexities to minimal. Also, reusing code from libraries and frameworks to aid the development process became a norm.

Most widely accepted architecture for these web applications uses web components developed with JavaScript using JavaScript Frameworks like Angular & React which runs inside the browser. These web applications, mainly communicate using the HTTP protocol with RESTful web services, to persist the application state. Apart from that, for very specific use cases such as real-time chat, web sockets are also being used. However, using web sockets has its own limits and still in its early adoption stages.

More specifically for object persistence, there are specialized RESTful web services offered by cloud providers which can directly communicate with the client part of the web applications runs inside the browser. These services provide high scalability, durability and availability of the objects [2] [3]. However, above capabilities comes with a cost, where partial updates of these objects are not possible. To persist dynamically changing objects requires an entire object to be sent to the persistence web service for each modification. When the frequency of dynamically changing object becomes higher, it increases the object persistence time, due to network latency, congestion and persistence web service throttling [4]. Longer response times for persisting objects may cause lower satisfaction and poor productivity among web application users. This problem can be solved by persisting only the last modification of the object, within a time window. This reduces the network congestion and also reduces the usage of persistence web service for higher frequency modifications, but, having a fix time window, also increases the persistence time for non-frequent modifications and increasing the staleness of modified objects. However, this creates challenges for web developers to additionally include an adaptive timing window to improve persistence response time. In addition, object size, persistence frequency and platform differences and network performance are difficult to predict.

The aim of this research is to implement a JavaScript Framework, which is capable of providing an adaptive persistence time window, to reduce unexpected object persistence times for frequently changing objects by reducing the network congestion and object persistence web service usage, enhancing overall web application user satisfaction.

The remaining parts of the document in structured as follows. Chapter 2 contains the literature review, which covers the theoretical aspects of the evolution of web applications, development, caching, data persistence mechanisms and related work done in this area. Chapter 3 contains the methodology and the architecture of the system developed. Finally, Chapter 4 contains the results and the conclusions drawn from this research.

CHAPTER 2

LITERATURE REVIEW

2.1 Evolution of web and mobile application paradigms

Over the past few years World Wide Web has grown immensely with boundless technology innovations. Initially, it all started with static web pages that only had the simply formatted content and links to other pages. With the rise of user expectations, serving more dynamic content was needed, which lead to the innovation of web server dynamically generated pages, giving birth to real web application engineering paradigm. Overtime dynamic web applications began to get popular, but had its inherited inefficiencies of limited user interactions, delayed loading web pages and delayed response to user actions. Although server performance is improved by caching dynamic data [5] [6] it partially solved the issue. This was due to the limitation of having rich user interactions, where most of the user action required to interact with the web server over the internet with inherent latencies. With the advancement of web browsers and web technologies, modern web applications started to use the web browser as an application platform where a significant part of the application logic executes inside the browser. JavaScript adds dynamism solving some of the inherent inefficiencies with traditional web applications, providing comparable user experience with desktop applications.

With the introduction of fast and efficient JavaScript engines, web browser containers began to be available on smart phone devices, which can run apps developed using web technologies like HTML5, JavaScript and CSS. This has given birth to a new paradigm called hybrid mobile application development. All together these innovations revolutionized the web application development landscape [7], to build web applications that run on a range of devices in different forms. Figure 1 shows a summary of different mobile application types.

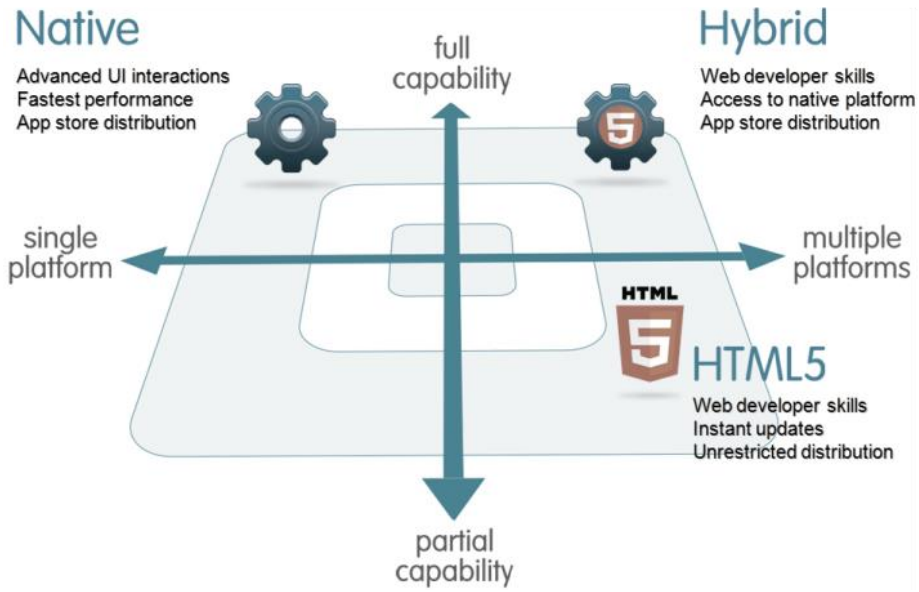

Figure 1: Native, HTML5 and Hybrid mobile application types

Source: http://svitla.com/wp-content/uploads/2013/02/Native_html5_hybrid.png

2.2 Web browser as an application platform

Over the past few years, web browsers had inherent shortcomings. These shortcomings, limited the development of rich web applications for a long period of time.

2.2.1 Web browser performance

Until recently when high performance JavaScript virtual machines were built, native functionality provided by web browsers were pretty slow especially when it comes to computationally intensive applications. The limitation of being single threaded increased the problem. Most of these challenges are now being addressed by the introduction of high performing JavaScript engines such as Google's V8 and the implementation of multithread support for JavaScript included in the HTML5 specification.

2.2.2 User interactions with web browsers

Until very recently web browsers had a very rigid user interaction model with a very primitive set of features. They lacked the desktop like rich user interactions, such as drag and drop, canvas rendering, printing, asynchronous communication & etc. Today almost all the web browsers support these features.

2.2.3 Network and security in web browsers

Most of the issues related to security [8] are originated from the document based “One size fits all” browser security model. Decisions on security were mainly originated from the origin of web server, but not by the specific need or functionality of the web application since these applications run on the sandbox environment with limited access to host machine that the web browser is running in. Now most of these challenges are getting revolutionized since the browsers provide more capabilities to use host machine resources (E.g. File access) and deploying different access control mechanisms.

Apart from these challenges, past web browsers required to reload the pages fully for each request. Now web applications not only capable of asynchronously communicating with the web server, but also communicating in real-time using web sockets.

2.2.4 Web browser compatibility and interoperability

Over the past few years, web browser compatibility was a major challenge. Although web applications are written, "Write once, run anywhere" (WORA), or sometimes write once, run everywhere (WORE) it didn’t work that way in different browsers. Most of the time, rich web applications failed to operate in certain browsers due to the differences in the Browser Object Model (BOM), Document Object Model (DOM) and rendered in unintended compositions and colors due to CSS rendering differences. This has given birth to libraries such as jQuery to be really popular to provide a uniform interface in manipulating. With the wide adoption of W3C standards, by web browser vendors, differences between web browser platforms were minimized.

2.2.5 Web application development and testing

The learning curve from statically typed languages to dynamically typed languages has been a challenge for web developers. Writing complex JavaScript was a difficult task and became challenging in applying software development methodologies due to the fundamental coding style differences like asynchronous, event driven using higher order functions. Debugging these applications also became a challenge during the past where inline code debugging was not available. Now these challenges are getting minimized with modern web browsers having embedded development, debugging and testing tools. Code and pattern reuse with libraries and frameworks also helped to rapidly develop JavaScript heavy web applications.

2.2.6 Web application deployment

One of the main strengths of Web applications is given by the inherent ability of instant deployment. This has revolutionized the software deployment landscape, allowing up to date software to be available instantly to the users with minimal distribution overhead. More recently, with the introduction of HTML5 App Cache, web application distribution has gone an extra mile in supporting the instant distribution of the updated web application to available offline in web browser clients.

2.3 Data driven web applications

With the advancement of web browsers, shift from server side to client side functionality became more apparent. This made the possibility for web applications to implement data handling in the browser, thus providing well needed features to support as an application platform. Client side data handling in web browsers involved in modifying data in the memory, mainly in the form of JavaScript Objects as well as persisting them in client side storage by serializing the data or directly storing the object depends on the storage of choice.

These web applications are capable of using the web browser storage or in memory storage, as well as distant storage to persist the data for long term storage and sharing. Figure 2 shows a basic overview of web and hybrid mobile application data communication model.

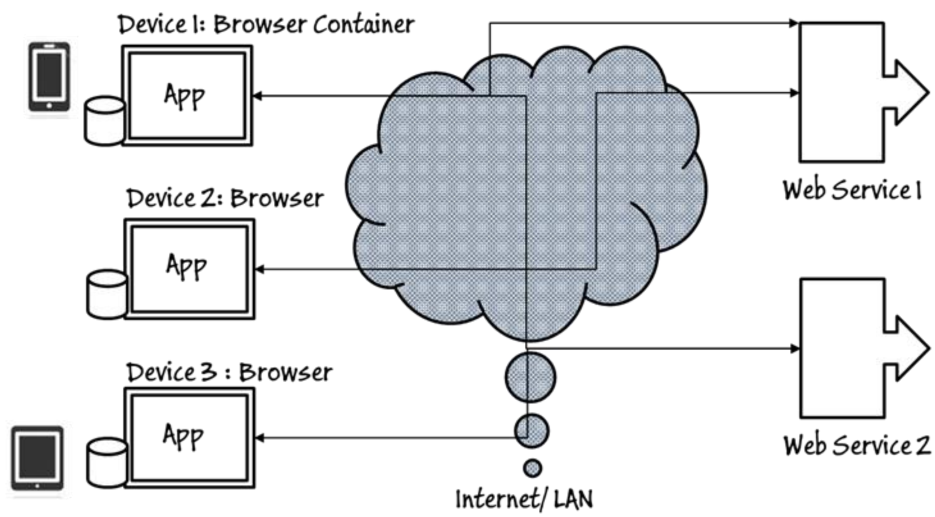

Figure 2: Web and hybrid mobile applications

2.4 Storage in web applications context

Web applications are capable of storing, data on the web servers for long term persistence and on client-side storage for temporal persistence. Conventional interaction model with server, has the following disadvantage [9].

1. Need of network connection availability for data persistence
2. Increased network congestion
3. Increase server load

This leads to the reduced performance. The conventional client-server interaction model is shown in Figure 3.

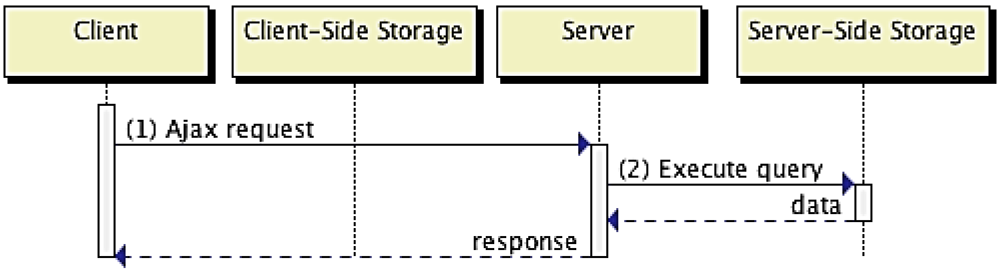

Figure 3: Client-server Interaction Model

Source: Figure 1 [9]

Usage of client-side storage or memory can minimize the above disadvantages illustrated using the following interaction model.

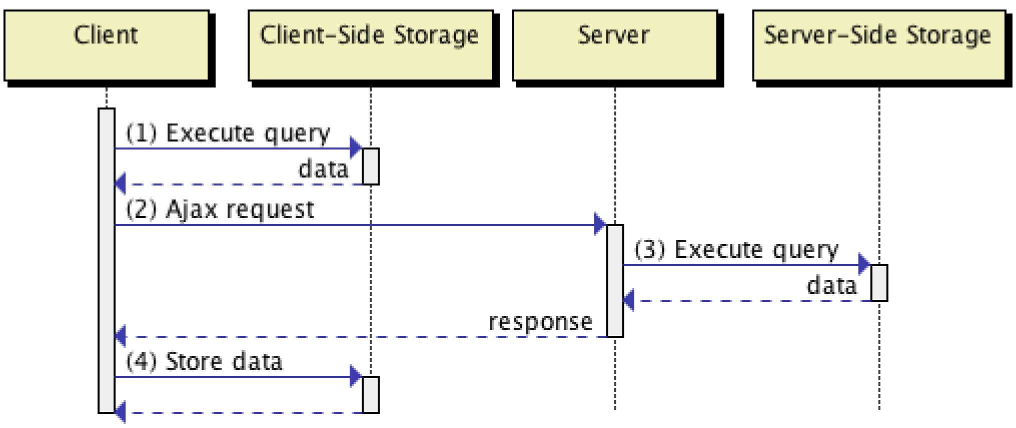

Figure 4: Client-server interaction model utilizing client-side storage or memory

Source: Figure 2 [9]

However, this introduces another challenge of data staleness, where client side is not having the most up-to-date data.

2.5 Abstracting storage in web applications

2.5.1 Separating web applications from user data storage with BSTORE

BSTORE is a framework that allows web developers to separate the application logic from its data storage providing a unified API to access data. It is designed to achieve independence between applications and storage provides allowing almost any application, avoiding the need for reserving any special functionality for the user.

Figure 5 shows an overview of the BSTORE architecture where component in the browser corresponds to a separate window whose web page contains JavaScript code that communicates with BSTORE. All requests are sent from each web page to the BSTORE file system are mediated by the FS manager.

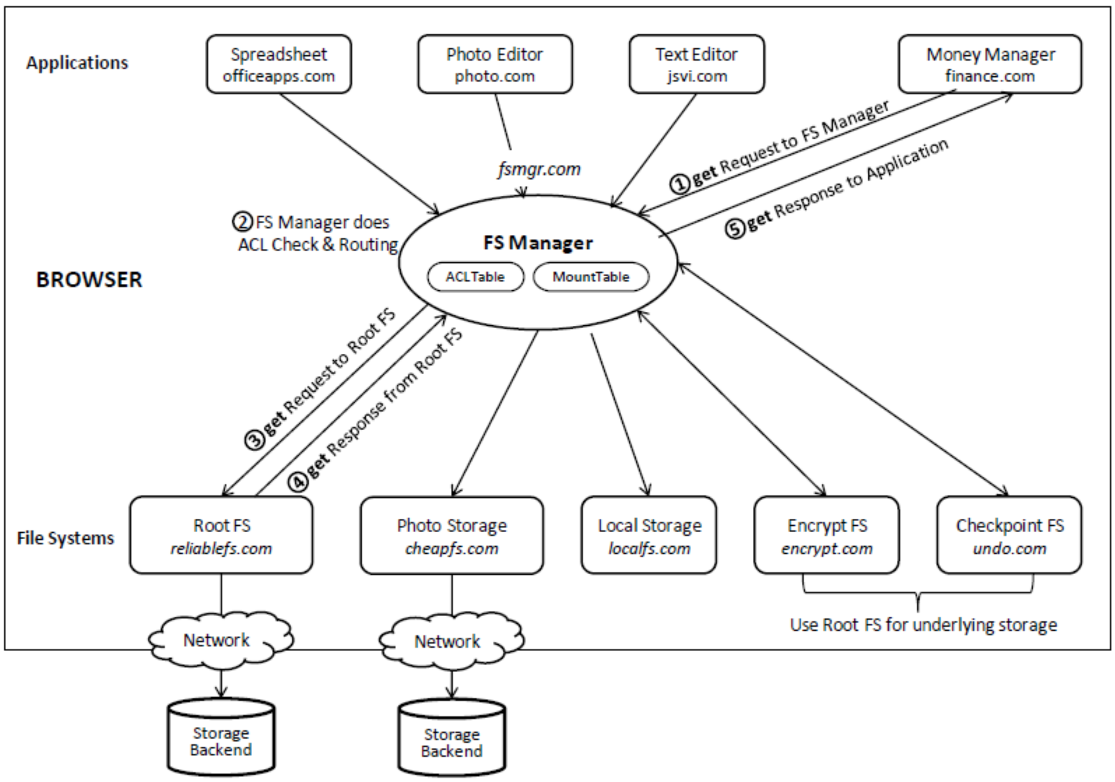

Figure 5: BSTORE architecture

Source: Figure 1 [10]

BSTORE client-side components are implemented using JavaScript targeting FireFox and Google Chrome web browsers. The specialty of BSTORE is that, each component runs on separate browser window loaded from a separate domain which also becomes a drawback when it comes to commercial applications where BSTORE to work requires multiple windows to be open at the same time. Communication between each of these browser windows are done using "postMessage" command. BSTORE communicates with the server storage system through AJAX and authenticated using, plain user credentials.

2.6 Utilizing client-side storage for web application performance

2.6.1 Sync Kit

Sync Kit [11] is a client/server toolkit for improving the performance of data intensive web applications. Sync Kit offers a mechanism to offload some of the data from a web server to (Google) Gears [12] on the client side, requiring the installation of the Gears plug-in in order to work. In future Gears could be, however, replaced with a W3C-specified client-side storage mechanism to get rid of the dependencies. On the server side, Sync Kit requires a specially designed web server that handles Sync Kit requests. In their case, Sync Kit was implemented as a collection of scripts for the Django10 web framework. The focus of Sync Kit is to use Gears to store (cache) segment of web pages called web page templates on the client side. These templates include a JavaScript library and data endpoint definitions to access dynamic contents residing on the server. Data endpoints, on the other hand, cache database objects to Gears and keep the cached data consistent with the server database. When a browser loads a web page for the first time, the template is returned as a response from the Sync Kit aware server and stored in the client-side storage. Then, new data is requested via the template’s data endpoints. Finally, the retrieved data is added to the template, cached on the client-side storage, and the result is displayed to the user as shown in Figure 6 and Figure 7.

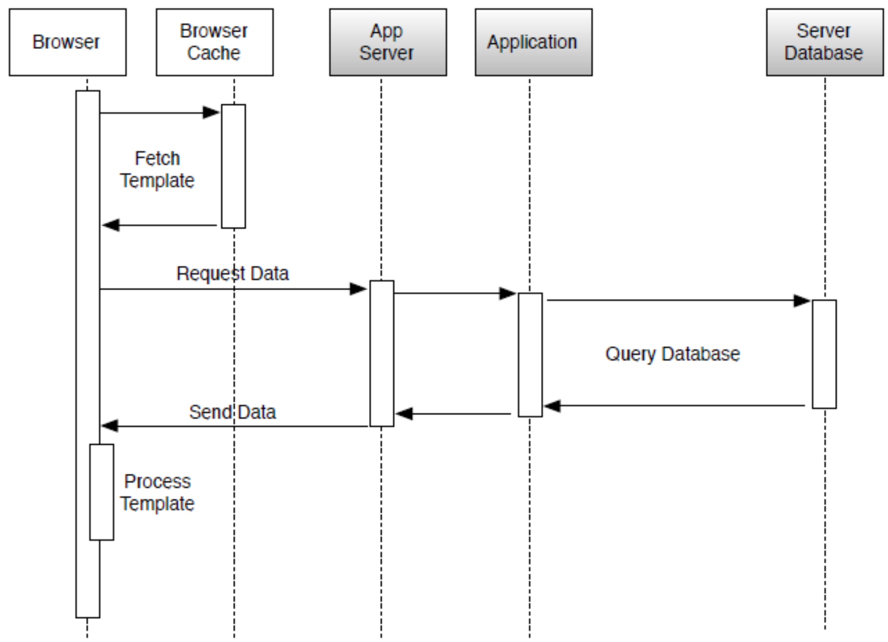

Figure 6: Template caching: Template rendering is on client

Source: (b) [11]

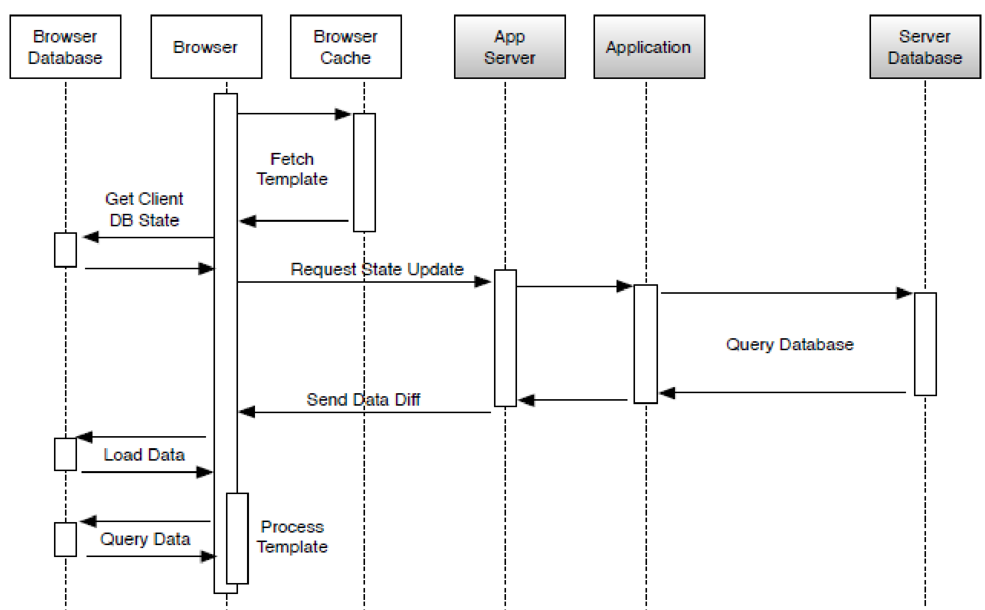

Figure 7: Sync Kit: Template rendering and database accesses

Source: (C) [11]

According to the research [11], their solution given by Sync Kit reduces server load by a factor of four and data transfer up to 5% compared to the traditional approach, when cache hit rates are high.

One of the main challenges of Sync Kit is that it requires Sync Kit aware web server implementation in server side that limits its usage in practical web application developments where more of these applications are built on top of third party APIs and services. Apart from that, it restricts the client-side implementation to be based on Sync Kit provided client-side template mechanism where better mechanisms are coming out more frequently suiting different JavaScript libraries and framework stacks.

2.6.2 Silo

Silo [18] is a framework that allows to create fast-loading web applications. It uses JavaScript and client-side storage based on current web standards, allowing the approach to be working on modern web browsers. On the server side, a Siloaware web server is required. The main purpose of Silo is to use client storage to cache website's JavaScript and CSS chunks (2 KB in size) on the client-side so that it is not needed to be requested from the server in each page load.

When a browser requests for a specific web page, the server returns a modified version of the web page containing a small JavaScript file with a list of required chunk ids and logic to fetch missing chuck IDs. Then the loaded JavaScript file logic checks the client-side storage for available chunks, and informs the server of the missing chunks using their ids where the server replies with the raw data for the missing chunks. Finally, the web page is reconstructed in its original form on the client using the JavaScript logic. Apart from that, for the latter HTTP request, the round trip can be completely eliminated by utilizing HTTP cookies in the initial HTTP request for sending the already available chunk ids. The Silo was evaluated with multiple real-world websites, such as CNN, Twitter, and Wikipedia.

Based on their experiments, it has proven that Silo can reduce web page load times by 20-80% for web pages with large amounts of JavaScript and CSS.

With the latest HTML5 standards, almost all the functionality provided by Silo can be achieved using App Cache and Client-side storages but stored separately in the browser. But the challenge addressed by providing a unified way of loading web page content (Both content and data in forms of chunks) provides several other challenges when it comes to practical web application development, making things really hard to test and debug.

2.7 Client-server data synchronization and persistence

2.7.1 Data synchronization patterns in mobile application design

Many mobile applications are data centric [13] . This is same with modern web applications. This research [13] describes common concerns related to data synchronization as a collection of patterns, grouped by the problems they address. These patterns are based on examining open source applications, inspecting platforms and frameworks of mobile systems and examining experiences developing mobile applications taking input from different experience developers.

The paper states that it is not possible to find one-size-fits-all solution for data synchronization which provides the need of configuring a data synchronization framework to suit different domains and environments. This becomes really obvious when considering indirect factors like power usage, especially when it comes to mobile devices, and efficient synchronization mechanism [14] needs to be selected based on various criteria.

As described in the paper these collections of patterns can be further classified into the following 3 collections of patterns.

1) Data synchronization mechanism patterns
2) Data storage and availability patterns

3) Data transfer patterns

2.7.1.1 Data synchronization mechanism patterns

These collections of patterns addresses the question: “when should an application synchronize data between a device and a remote system (such as a cloud server)?” [13]. These patterns can be further classified into two types based on the mechanism that the data transfer occurs.

- Asynchronous data synchronization:
  Where data synchronization happens asynchronously without blocking the user interface.
- Synchronous data synchronization:
  Manage a data synchronization event synchronously; blocking the user interface while it occurs.

2.7.1.2 Data storage and availability patterns

These collections of patterns address the questions: “how much data should be stored?” and “how much data should be available without further transfer of data?” [13]. These considerations often depend on the limitations of mobile platforms which is similarly applicable to web browser platforms. Based on the amount of data stored locally, these collections of patterns can be further classified into the following types.

- Partial storage:
  Synchronize and store data only as needed to optimize network bandwidth and storage space usage.
- Complete storage:
  Synchronize and store data before it is needed so the application has better response or loading time.

2.7.1.3 Data transfer patterns

These collections of patterns addresses address the problem of transfer quantity in set reconciliation: “how can we synchronize between sets of data such that the amount of data transmitted is minimized?” [13]. Selection of the transfer quantity needs to be carefully selected based on the application domain to optimize the network bandwidth. Further classification of data transfer patterns is shown below.

- Full transfer:

  On a synchronization event, the entire dataset is transferred between the mobile device and the remote system.

- Timestamp transfer:

  On a synchronization event, only the parts of the dataset changed since the last synchronization are transferred between the mobile device and the remote system using a last-changed timestamp.

- Mathematical transfer:

  On a synchronization event, only the parts of the dataset changed since the last synchronization are transferred between the mobile device and the remote system using a mathematical method.

2.7.2 Efficient synchronization of replicated data in distributed Systems (nSync)

This research present nsync [15], a tool for synchronizing large replicated data sets in distributed systems. Although these concerns are not directly applicable for web applications, the strategy of optimizing the synchronization plans provides a sound approach in optimally synchronizing batches of replicated data when multiple servers are present [16]. When it comes to data synchronization, nsync approach computes nearly optimal synchronization plans before actual synchronization happens to minimize the amount of data needs to be transferred using a hierarchy of gossip algorithms that take the network topology into account.

Figure 8, illustrates the capability of nsync when it comes to synchronizing multiple nodes. In native approach without nsync, each single connection would trigger an independent disk access, provoking many disk head movements and therefore resulting in a slow data transfer rate. To avoid this nsync approach uses only node to node (n2n) syncs where each node participates in at most one n2n sync at a time.

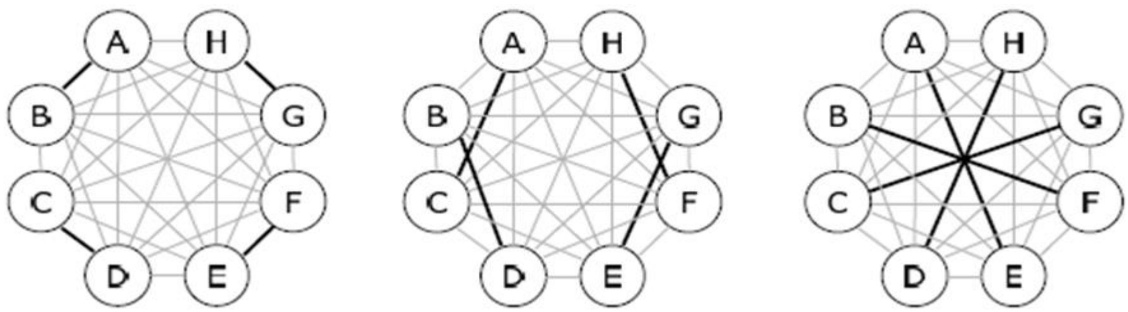

Figure 8: Three rounds of n2n syncs can synchronize eight nodes
Source: Figure. 1 [15]

2.7.3 Automated object persistence for JavaScript

Automated persistence of JavaScript Objects [17] provides a transparent mechanism on persisting structured objects. Still with automated persistence, web application developers, need to determine how and when to save data using the heap-allocated JavaScript objects to local browser storage balancing several tradeoffs. If the changes are saved quite often, it causes degradation of performance and the single threaded nature of JavaScript (At the time being research was conducted) causes to freeze the browser when saving single large batch of changes. To deal with this problem, they have investigated the use of persistence by reachability [18] for JavaScript. This approach divides the JavaScript object heap into transient and persistent sub-graphs allowing the framework to manage persistence-related tasks periodically. This way the framework manages between transient and persistent objects not immediately persisting the objects each time it gets updated.

Secondly, to synchronize data between client-storage to servers, they have investigated and developed an application framework for automated persistence at an individual object granularity, allowing the browser to send updates of the data for individual objects which maps directly to the object model created by the application developer. When there is a conflict happens, the framework triggers a conflict resolution callback where the developer needs to write a semantically relevant resolution strategy.

The automated object persistence framework uses special functions called “accessors” in JavaScript to identify whether an object attribute is updated. These “accessors” are called transparently when an object property is read or written. In this framework, when a write operation happened, triggering the “accessor”, it records that the object as mutated and stores the written value so it can be returned by the read “accessor”. Since JavaScript is a dynamic language, a scheduled process runs to identify whether new properties are added to the JavaScript object, carefully selecting the schedule intervals to avoid overhead.

As shown in the architecture diagram in Figure 9, “accessors” act as a read barrier to alert the framework when a persistent object has mutated. These Mutations including both “accessors” and the dynamic property addition, maintenance task, causes objects to be added to the live object set that is constantly updated by the application and have their state stored in local storage. This is to assure that the objects are stored safely in the browser if the browser crashed before persisting the objects. Garbage collection removes dead objects from local storage. Finally, mutations are sent to a server during remote synchronization.

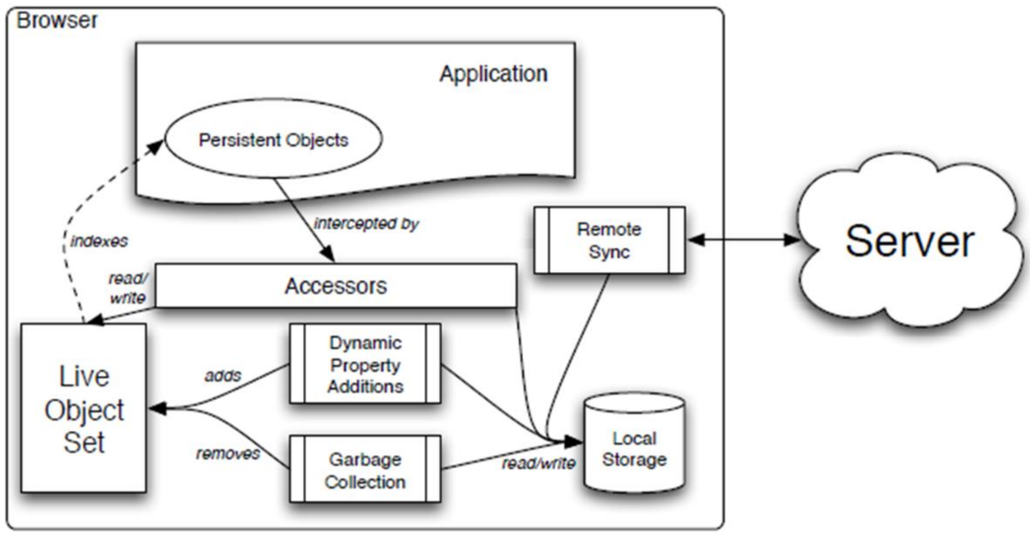

Figure 9: Architecture diagram of automated object persistence mechanism

Source: Figure 1 [17]

This paper provides a sound approach, in providing semi-transparent object persistence mechanisms using a framework. Some of the key decisions made based on the single thread nature of JavaScript are no longer valid since now JavaScript supports multiple threads with web browser technology advancements.

2.8 Response time impact for web application user experience

In modern web applications user satisfaction heavily depends on application response times. With the advancements of web browser performance, network speeds, one might not expect to have to deal with performance issues in web applications such as slower response time. But these issues remain a very real concern today.

According to the research System Response Time and User Satisfaction [19], system response time has a related effect for user satisfaction. According to the study Figure 10 shows that the user satisfaction has a correlation with response time where it decreases as response time increases. It also showed that for web applications, it appears to be a level of intolerance in the 12-second response range.

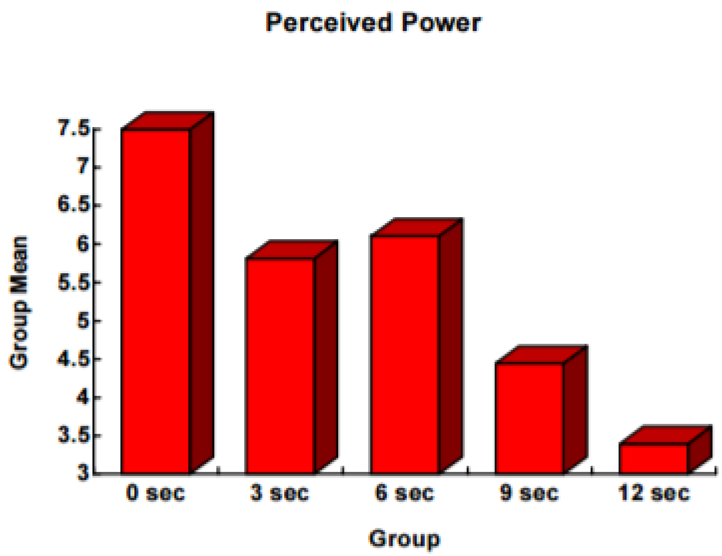

Figure 10: Perceived power construct

Source: Figure 4 [19]

CHAPTER 3

METHODOLOGY

The JavaScript object persistence framework consists of multiple components. The persistence algorithm in the framework intervenes the JavaScript object persisting operation and adjust the persistence timing for high frequency actions. This is done by adding a modification window, optimizing for reduced average persistence time. After the modification window is calculated, the metadata of the overall operation is saved for future analysis and prediction.

3.1 JavaScript persistence objects

In web applications object representation can take multiple formats. Most popular forms of JavaScript objects created with JavaScript Object Literal Notation (JSON). These objects consist of attributes and collections in the form of primitives and arrays. An example JSON object is shown in the Figure 11.

```json
{
  "todos": [
    {
      "title": "Item 1",
      "completed": false
    },
    {
      "title": "Item 2",
      "completed": false
    },
    {
      "title": "Item 3",
      "completed": false
    },
    {
      "title": "Item 4",
      "completed": false
    }
  ],
  "request_id": "8ae873a4-5634-75d6-3bf8-ed93bf4572d1"
}
```

Figure 11: JavaScript Object Notation

Apart from these JavaScript Object Notation, JavaScript Engines in web browsers also allows to upload binary objects which could be directly sent to web services.

3.2 Object persistence web Services

Object persistence web services are set of web services which allows to store objects sent over HTTP protocol with a payload of JSON or blob objects. These services internally handle the persistence complexities of the objects and also provides other capabilities such as updating, deleting objects.

3.3 Factors and their ranges

JavaScript object persistence performance is impacted by various parameters such as

- Object modification & persistence frequency
- Object size
- Network performance (Bandwidth and Latency)
- Platform (Browser/Operating System)
- Web application specific parameters
- Performance of persistence web service

If we take a particular feature of a web application that needs to persist a JavaScript object, it has a size, and user interaction frequency to persist the object within a quantifiable range which allows to model a characteristic of a feature. Optionally, other factors also help to classify the user, such as an authenticated user identity, platform identifying data (browser cookies) which could be used to further increase the predictability of user behavior with respect to persist frequency. Apart from that most of these factors impact persistence response time, in different levels and it is quite difficult to identify ranges for some of these factors, for example web application specific parameters or performance of persistence web service, where it will be different based on implementations and also changes over time. Therefor an adaptive method is needed to predict the modification window for a given feature to optimize the performance for object persistence for high persistence frequencies.

3.4 JavaScript object persistence models

3.4.1 Persistence model: direct persistence

In modern web application, the object changes are persisted in RESTful web services. If we model the modifications and persistence time of the object, it is a series of modifications that happens with time. To store the state of the object on the remote server requires it to trigger a server data persistence AJAX request that takes a round trip time (RTT) to the sever to persist the data remotely and send back the acknowledgement. In the direct persistence model, it immediately persists the object, when the object identifies a change in its attributes. Figure 12 illustrates this model and how these steps taken place.

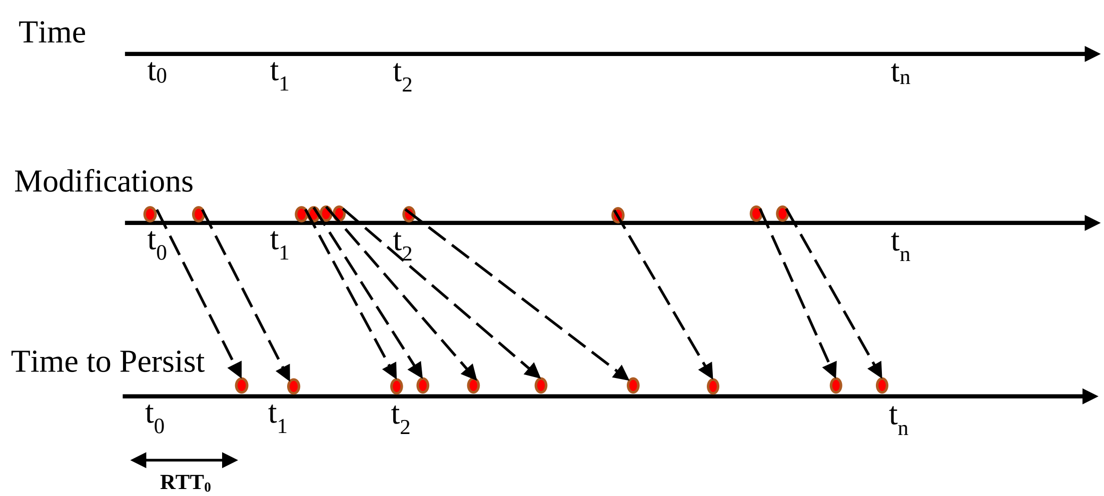

Figure 12: JavaScript persistence illustrated

RTT- Round Trip Time (Client-Server-Client)

W- Modification Window

3.4.2 Persistence model: dynamic modification window persistence

In the dynamic modification window persistence model, object changes are not directly sent to the server. A time window is calculated and within the window only the last object change is sent for persistence reducing the network congestion and service usage. Figure 13 illustrates this model and how these steps taken place.

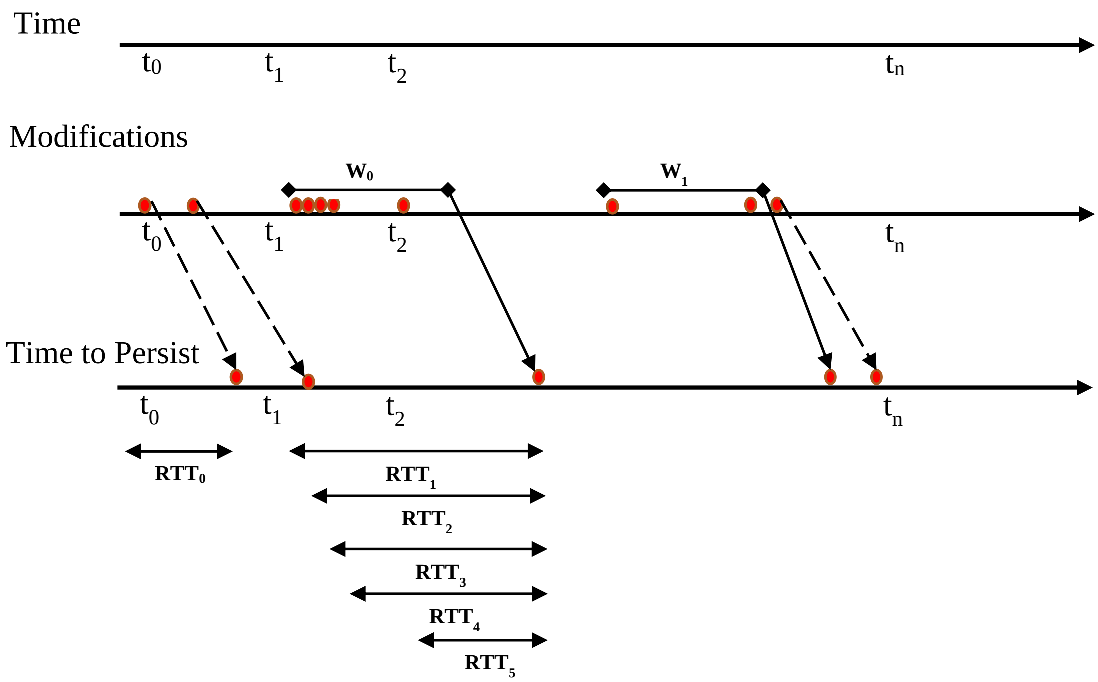

Figure 13: Dynamic modification window persistence

This forms the following equation for round trip time Average (WRTT (m)) taking modification window into account, to persist an object after **m** requests.

Window RTT Average [WRTT (m)] $= \frac{\sum_{x=1}^{x=m} RT(x) + W(x) - ST(x)}{\sum_{x=1}^{x=m} x}$

Request RTT for request **m,** [RTT (m)] $= RT(x) - ST(x)$

Where:

ST(x) = Start Request Time, RT(x) = Response Received Time, W(x) = Modification Window for a request **x**.

3.4.2.1 Predicting next request round trip time

To calculate the dynamic persistence window size for dynamic modification window persistence, polynomial regression is being used to predict the next request round trip time. To determine the order of polynomial regression required to predict next round trip time, a higher order is initially selected and equation is calculated as shown below.

$$y = 0x\text{^}10 + 0x\text{^}9 + 0x\text{^}8 + 0x\text{^}7 + 0x\text{^}6 + 0x\text{^}5 + 0x\text{^}4 + -0.12x\text{^}3 + -5.73x\text{^}2 + 765.62x + 943.16$$

It has been found that a polynomial regression with order 3 is sufficient to predict the next request round trip time. After retrieval of each response, each round trip times are logged and use to predict the next value using polynomial regression.

3.4.2.2 Adjusting dynamic modification window size

W_{size} is calculated as follows after **m** requests

If $RTT_{predicted}(m+1) - RTT\ (m)) > \Delta RTT_{threshold}$, Then

$W_{size}\ (m) = W_{size}\ (previous) + \Delta W_{increment}$

Else

$W_{size}\ (m) = MAX\ ((W_{size}\ (previous) - \Delta W_{decrement}), 0)$

End

Where:

Round trip time threshold = $\Delta RTT_{threshold}$, Predicted next request round trip time = $RTT_{predicted}(m+1)$, Modification window threshold = $\Delta W_{threshold}$, Modification window increment = $\Delta W_{increment}$, Modification window decrement = $\Delta W_{decrement}$

Other metrics calculated for future analysis include

Minimum round trip time = RTT_{min}, Maximum round trip time = RTT_{max}

3.4.3 Persistence model: dynamic modification window persistence with initial window size

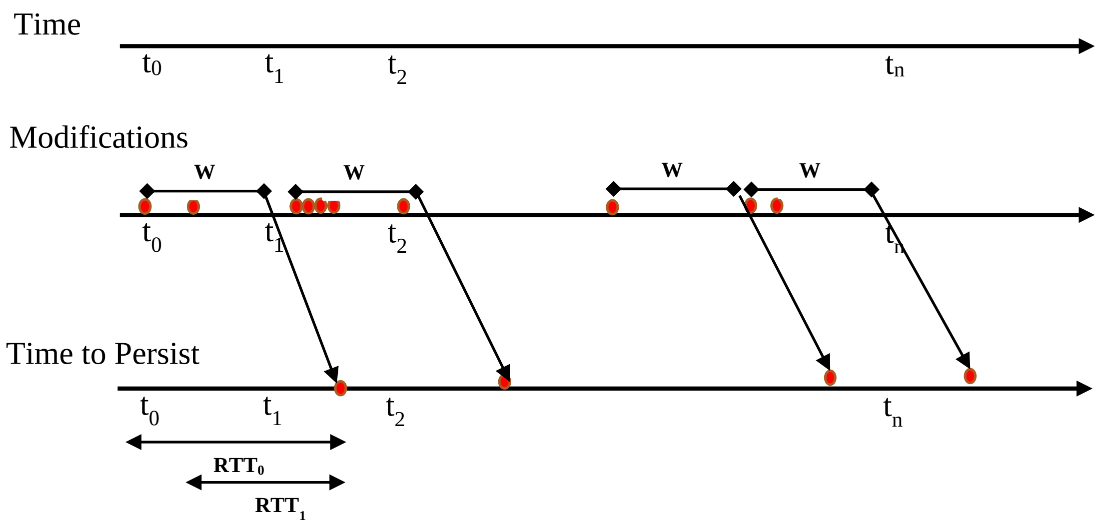

Figure 14: Dynamic modification window persistence with initial window size

In dynamic modification window persistence model, the window size is calculated dynamically to reduce network congestion. However, if high frequency object modifications happen immediately after starting the application, the network congestion and service usage can increase since the dynamic modification window calculation happens after individual responses arrives. In this persistence model, it initializes a suitable modification window at the beginning of the persistence request to avoid high frequency updates happening immediately. To predict the initial persistence window size $W_{\text{initial size}}$, further research needs to be carried out by using the data from dynamic modification window persistence model experiments.

3.5 Experiment setup

3.5.1 JavaScript object persistence window experiment

Persistence workload is created with JavaScript objects considering JavaScript object size, persistence frequency and network in to consideration. Figure 15 illustrates the experiment setup.

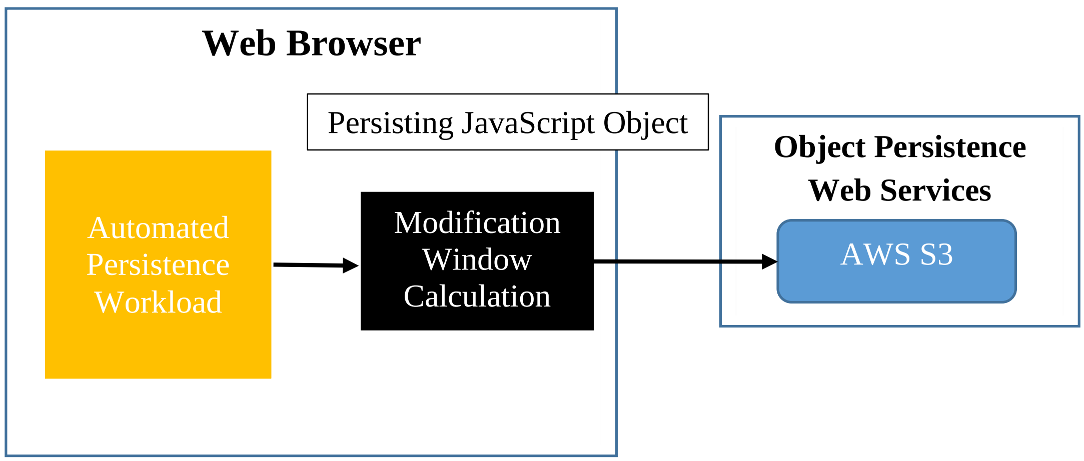

Figure 15: Experiment setup

3.5.2 Predicting next request round trip time

To predict next request round trip time, following function is used where the order of the polynomial regression algorithm is specified. Figure 16, illustrates the logic implemented to predict the next request round trip time.

```
predictNextRTT = function(req) {
    if (REQ.log.length < 3) {
        return req.requestRTT;
    };
    var result, reqLogs = utils.clone(REQ.log);
    reqLogs.push([reqLogs.length, null]);
    result = regression('polynomial', reqLogs, 3);
    return Math.abs(Math.round(_.last(_.last(result.points))));
},
```

Figure 16: Predicting next request round trip time

3.5.3 Changing persistence frequency

JavaScript object modification frequencies are simulated using a JavaScript interval function and using QUnit framework. Figure 17, illustrates the logic implemented to simulate persistence frequencies and object modifications.

```
asyncTest('Experiment with 1KB and 20 requests/second with 100 iterations', function() {
    var iterations = 100,
        uploadLogs;

    var refreshIntervalId = setInterval(function() {
        iterations--;
        reqwin.adaptiveWindow(EXPERIMENT.upload).then(function(logs) {
            uploadLogs = logs;
        })
        if (iterations === 0) {
            clearInterval(refreshIntervalId);
        }
    }, 50);

    expect(1);
    _.delay(function() {
        var logs = formatLogs(uploadLogs);
        ok(true, JSON.stringify(logs));
        start();
    }, 20000);
});
```

Figure 17: Simulating JavaScript persistence frequency

3.5.4 Changing latency, network bandwidth

Throughout the experiment, latency differences are kept to a minimum by using the same geographical distance between the client and the persistence service. Network bandwidth and additional latency are simulated using Google Chrome Developer Tools as shown in Figure 18. Also, many iterations are carried out to reduce the error factor when comparing results.

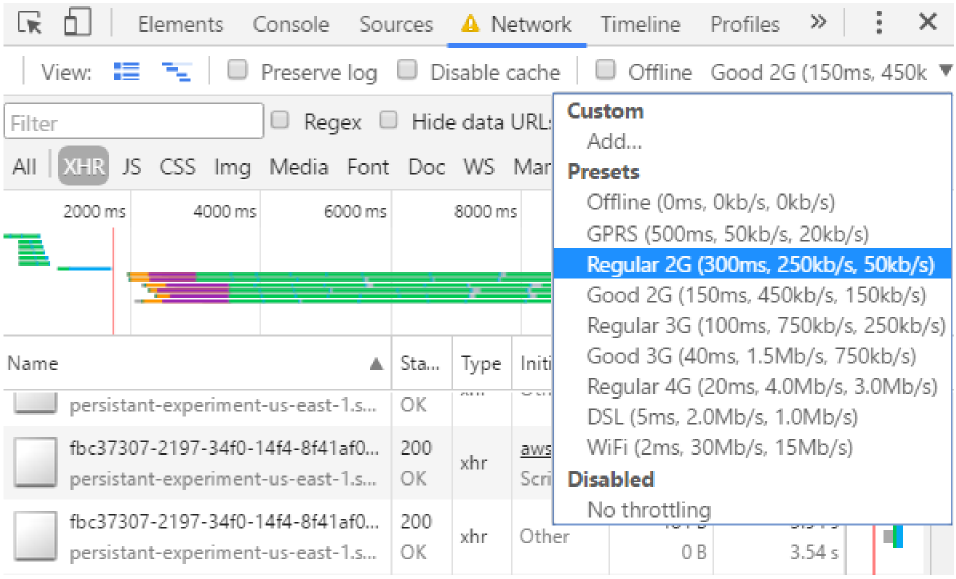

Figure 18: Simulating network bandwidth & latency

3.5.5 Adjusting persistence modification window algorithm parameters

The persistence algorithm works asynchronously adding persistence window for a provided "Save Callback" function so that, the actual persistence logic is written outside the algorithm itself. This helps to inject various persistence mechanisms reusing the algorithm implemented. For instance, saving on Amazon S3 persistence service is implemented as shown in Figure 19.

```
(function(W, $) {
    W.AP = W.AP || {};
    AWS.config.update(W.AP.AWS_CREDENTIALS);
    var recordsLog = {};

    AP.persist = {
        upload: function(obj) {
            var deferred = Q.defer(),
                s3 = new AWS.S3(),
                params = {
                    Bucket: 'persistant-experiment-us-east-1',
                    Key: obj.request_id,
                    StorageClass: 'REDUCED_REDUNDANCY',
                    Body: JSON.stringify(obj)
                };
            recordsLog[obj.request_id] = W.AP.utils.clone(obj);
            s3.upload(params, function(err, res) {
                deferred.resolve(recordsLog[res.key]);
            });
            return deferred.promise;
        }
    };
})(window, jQuery);
```

Figure 19: Persistence logic in Amazon S3

In the persistence algorithm, there are several parameters we need to adjust to optimize the algorithm for varying scenarios. During these experiments the effect of these parameters are further analyzed and predicting it for various scenarios requires further research using an analytical approach for real web applications, collecting the data of its performance results. Following are the parameters that need to be modified to optimize the algorithm for specific usage.

1. Initial window size
2. Window increment and decrement
3. Round trip time threshold

3.5.6 Implementing dynamic modification window persistence model

Persistence Window Calculation Algorithm has 3 steps. In the first step as shown in Figure 20, algorithm buffers further requests, if they are within the Window Size. If the Window

Size is 0, then all the persistence requests are being sent to the server. To send particular versions of the object change to the server, the persistence callback is called and upon its success.

```
adaptiveWindow = function(saveCallback) {
    var deferred = Q.defer(),
        currentRequestAt = utils.timestamp(),
        waitingWindow;
    records.lastRequestedAt = records.lastRequestedAt || utils.timestamp();
    waitingWindow = currentRequestAt - records.lastRequestedAt
    REQ.total++;

    records.window.push({
        request_at: currentRequestAt
    });

    var executeRequest = function() {
        var requestId = utils.guid();
        records.lastRequestedAt = utils.timestamp();
        records.saving[requestId] = utils.clone(records.window);
        records.saving[requestId].request_at = records.lastRequestedAt;
        records.window.length = 0;
        saveCallback(requestId, records.lastRequestedAt).then(function(obj) {
            W.count++;
            var requestRTT = (utils.timestamp() - records.saving[obj.request_id].request_at);
            W.time = W.time + requestRTT;
            records.saving[obj.request_id].forEach(function(record) {
                var result = {
                    requestRTT: requestRTT,
                    serveTime: utils.timestamp() - record.request_at + W.size
                };
                updateVariables(result);
                adjustRequestWindow(result);
            });
            deferred.resolve(LOGS);
        });
    };

    if (!W.size || waitingWindow >= W.size) {
        executeRequest();
    } else if (!records.pendingDelayedExecutions) {
        records.pendingDelayedExecutions = true;
        _.delay(function() {
            records.pendingDelayedExecutions = false;
            executeRequest();
        }, (W.size - waitingWindow));
    }

    return deferred.promise;
};
```

Figure 20: Persistence window buffer and persistence callback

In the second step algorithm re-calculates its statistics as shown in Figure 21 which is used to dynamically calculate the persistence window during the next step.

```
updateVariables = function(record) {
    REQ.served++;
    REQ.time = REQ.time + record.requestRTT;
    REQ.average = Math.round(REQ.time / (REQ.served || 1));
    RTT.average = Math.round(W.time / (W.count || 1));
    RTT.min = (record.requestRTT > RTT.min) && RTT.min ? RTT.min : record.requestRTT;
    RTT.max = (record.requestRTT > RTT.max) ? record.requestRTT : RTT.max;

    var log = utils.clone({
        RTT: RTT,
        W: W,
        REQ: REQ
    });
    log.REQ.windowRTT = record.windowRTT;
    log.REQ.requestRTT = record.requestRTT;
    LOGS.push(log);
},
```

Figure 21: Calculating algorithm statistics

During the third step, it adjusts its Window size accordingly using the persistence model, dynamic modification window persistence as shown in Figure 22.

```
adjustRequestWindow = function(req) {
    if (!W.disabled) {
        if (RTT.nextRTT - req.requestRTT > RTT.threshold) {
            W.size = W.size + W.increment;
        } else {
            W.size = W.size - W.decrement;
            W.size = W.size > 0 ? W.size : 0;
        }
    }
},
```

Figure 22: Adjusting persistence window

The first stage of the experiment is carried out without the persistence window in range of object modification frequencies. In the next stage, different sizes of fixed persistence windows added and the result is calculated. At the last stage of the experiment, dynamic modification window is added, in between the automated object persistence workload and Amazon S3 web service. Using the automated interval script, workload frequency is simulated and required persistence algorithm parameters are modified accordingly. After simulation completes, a delayed function shows experiment results in QUnit web interface and shown in Figure 23. In addition, the results of the experiments automatically added to an Excel Sheet for further analysis.

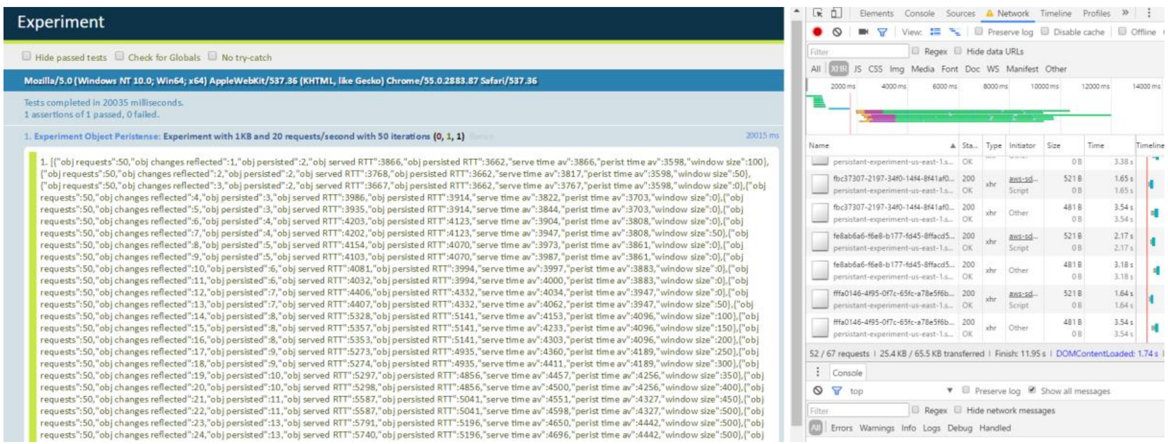

Figure 23: Displaying results using QUnit Framework and Google Chrome inspector

CHAPTER 4

Results & Conclusion

After carrying out the experiment in 3 stages, with higher object modification frequencies and modifying different parameters affecting object persistence performance

1. Without persistence window (Directly persisting)
2. With fixed persistence windows
3. With dynamic persistence windows

The results obtained from the experiments are analyzed in this chapter and discussions are carried out interpreting the results. At the end of the chapter, it comes to the conclusions of experimental findings. At last, future work is proposed to further expand the research.

4.1 Directly persisting without persistence window

In the first stage of the experiment, objects are directly persisted. The experiment is carried out by changing the network latency and speed using Google Chrome Developer Tools and object modification frequency. Figure 24 shows, change in persistence time for different object modification frequencies in a simulated GPRS mobile network with 500ms extra latency and 20-50kb/s data transfer speeds using 1KB size object.

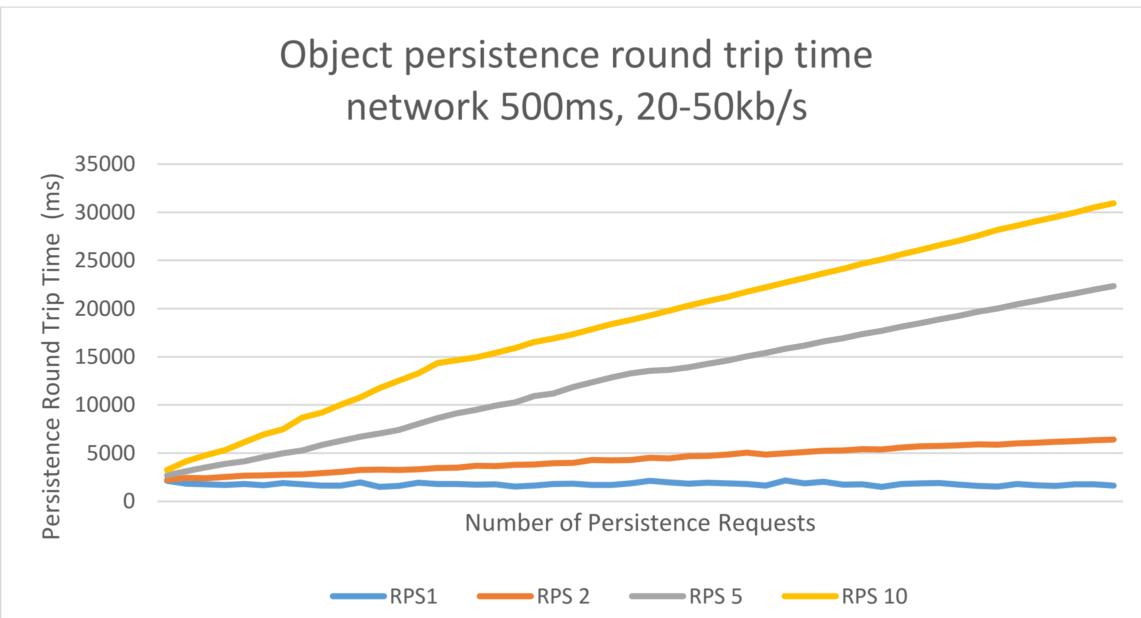

Figure 24: Round trip time vs modification frequency in GPRS network 500ms, 20-50kb/s

According to Figure 24, for higher object modification frequencies of 5 and 10 requests per second, persistence round trip time of individual object modification increases rapidly going beyond 12 seconds, which is the time limit [19] where users begin to doubt whether the web application is malfunctioning.

To understand the effect of network performance, for object persistence round trip time, next two experiments are carried increasing the network performance. Figure 25 and Figure 26, shows the results of the experiment.

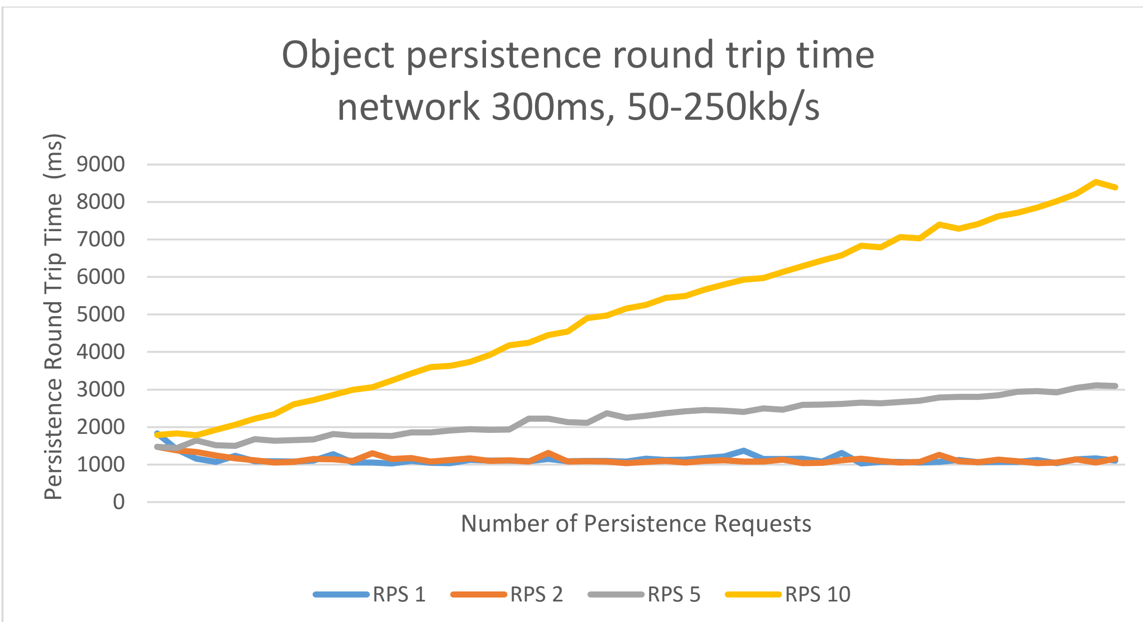

Figure 25: Round trip time vs modification frequency in regular 2G network 300ms, 50-250kb/s

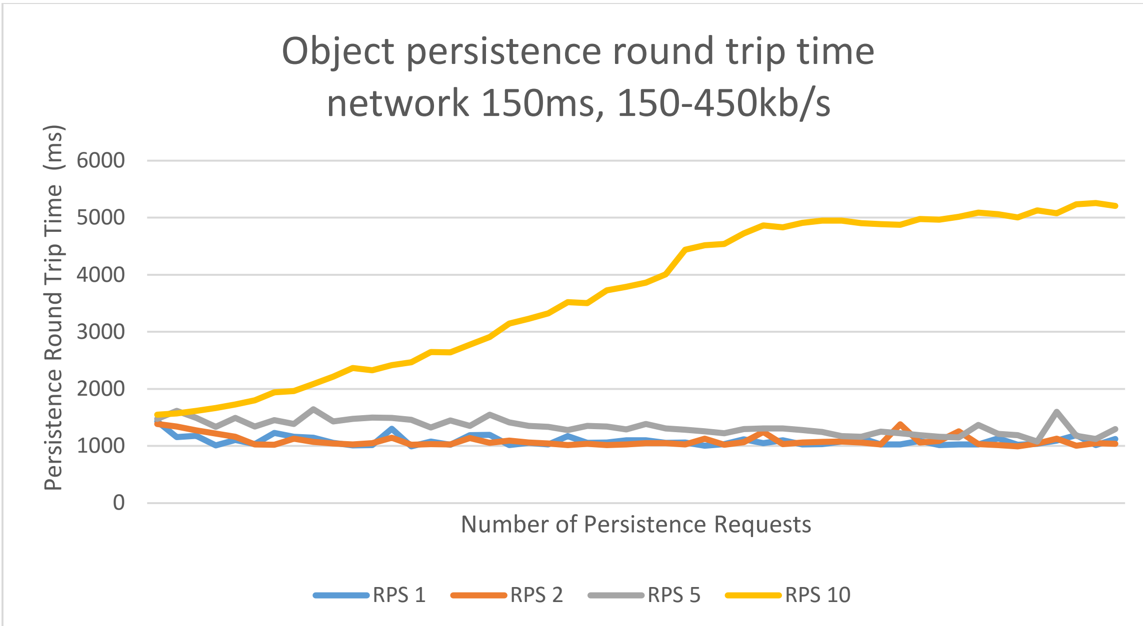

Figure 26: Round trip time vs modification frequency in good 2G network 150ms, 150-450kb/s

According to the Figure 25 & 26 shown above, increase in network performance reduces object persistence time. In these two network simulations only the highest frequency of 10 object modifications per second goes beyond the accepted response time limit. Based on the direct persistence results,

$$Object\ Persistence\ Time \propto \frac{Object\ Modification\ Frequency}{Network\ Performance}$$

Increase in frequency affects the persistence round trip time, proportionally for higher frequencies. The network speed affects the round trip time inverse proportionally.

4.2 Persisting objects with fixed persistence window

Since the most significant increase on persistence timing occurred with the GPRS network performance of 500ms, 50-250kb/s and persistence frequency of 10 object modification requests per second, further experiments are carried out adding different persistence windows. Figure 27 shows how object persistence round trip time changes with persistence window sizes of 100ms, 200ms, 500ms, 1000ms and 2000ms in comparison with not having a persistence window.

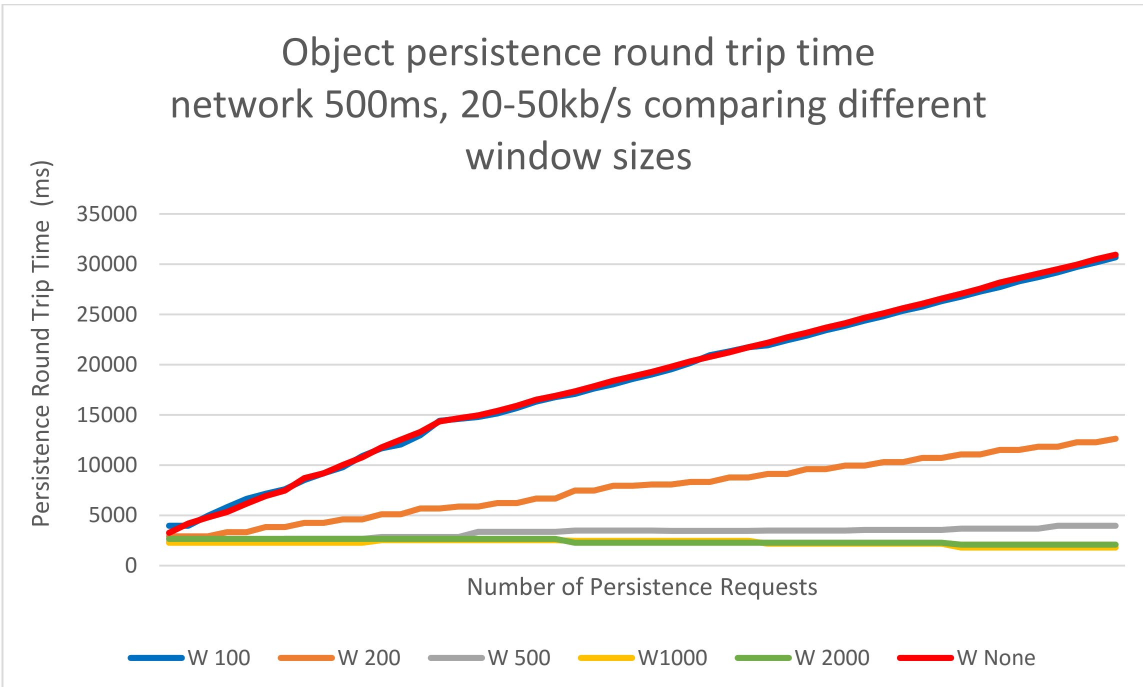

Figure 27: Round trip time vs persistence window size, 10req/s in GPRS network 500ms, 20-50kb/s

By adding a persistence window, it reduces the congestion of network by only persisting the last modified object within a time frame. This has a similar effect as reducing the object modification frequency. As shown in Figure 27, when the persistence window gets closer to the object modification frequency, the effect in addition of the persistence window is hardly noticeable. However, by increasing the persistence window above object modification frequency, it is clearly reducing the object persistence round trip times. In fact, best object persistence round trip time, for the least persistence window size happens at the persistence window size of 1000ms. Further increasing persistence window does not reduce the persistence round trip time for the given frequency.

If we define a parameter called total persistence time of an object where,

$$\begin{aligned} &Total\ persistence\ time\ of\ an\ object \\ &\qquad = waiting\ time\ in\ persistence\ window \\ &\qquad + persistence\ round\ trip\ time \end{aligned}$$

Figure 28, shows how total persistence time of an object changes with total round trip time average.

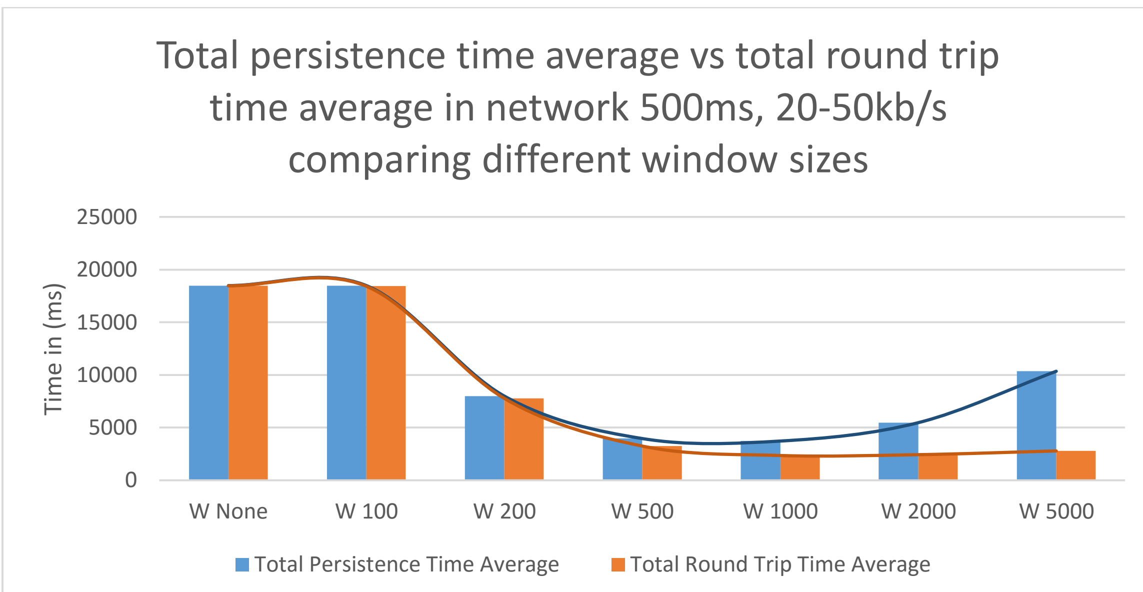

Figure 28: Total persistence time average vs total round trip time, 10req/s in GPRS network 500ms, 20-50kb/s

The downside of increasing the persistence window is the objects getting stalled in the client and beyond the optimum point, it increases the actual persistence round trip time since, the object needs to wait a time interval of the window before sent for persistence service. According to Figure 28, minimal value of total object persistence time average and the total round trip time average is achieved at the persistence window size of 1000ms.

Increasing the persistence window beyond 1000ms, total persistence time average increases. Therefore, it requires to select a suitable size for the persistence window to frequently persist the objects to the persistence service, but not too quick to congest the network or hit the throttling limit of the persistence service, which will increase the individual round trip times and total persistence time average.

4.3 Persisting objects with dynamic persistence window

In web applications, object modification frequencies are not fixed and also network performance and server throttling limits can vary drastically. Still persisting high frequency object modifications in slow networks is a challenge. Previous experiments clearly shown that adding a persistence window can reduce the object persistence round trip times. However, to improve the object persisting performance web applications, it is required to come up with a mechanism to calculate a dynamic persistence window to reduce, network congestion and reaching persistence service limits.

4.3.1 Adjusting persistence window

To adjust the dynamic persistence window, several algorithms were initially considered.

Window RTT Average [WRTT (m)] $= \frac{\sum_{x=1}^{x=m} RT(x) + W(x) - ST(x)}{\sum_{x=1}^{x=m} x}$

Request RTT for request **m,** [RTT (m)] $= RT(x) - ST(x)$

Where:

ST(x) = Start request time, RT(x) = Response received time, W(x) = Modification window for a request **x**.

In the first approach dynamic persistence window is calculated as

W_{size} is, dynamic persistence window after **m** requests

If $RTT(m) – Average (WRTT (m)) > \Delta RTT_{threshold,}$ Then

$W_{size} (m) = W_{size} (previous) + \Delta W_{increment}$

Else

$W_{size} (m) = MAX ((W_{size} (previous) – \Delta W_{decrement}), 0)$

End

Where:

Round trip time threshold = $\Delta RTT_{threshold,}$ Predicted next request round trip time = $RTT_{predicted}(m+1)$, Modification window threshold = $\Delta W_{threshold}$, Modification window increment = $\Delta W_{increment,}$ Modification window decrement = $\Delta W_{decrement}$

This algorithm worked for medium object modification frequencies. However, when the object modification frequency becomes higher, and the algorithm operates for longer period of time, Average (WRTT (m)) increases. The average round trip time further increases with the addition of persistence window. Since the persistence window is increased only according to the following condition,

$$RTT(m) – Average\ (WRTT\ (m))\ >\ \Delta RTT\ threshold$$

Over number of object persistence requests, persistence window increase becomes less.

More modifications were done to improve the persistence algorithm by rotating conditional parameters for window adjustment, using the average, maximum and minimum statistical values calculated after each persistence round trip. However, these modifications didn’t provide satisfactory results.

Then a numerical approach is used to calculate the next round trip time, using polynomial regression. As shown in the following equation, $RTT_{predicted}(m+1)$ and current RTT difference was considered and if it goes beyond a certain threshold, the persistence window is adjusted.

If RTT $_{predicted}$(m+1) - RTT (m)) > ΔRTT $_{threshold}$, Then

W $_{size}$ (m) = W $_{size}$ (previous) + ΔW $_{increment}$

Else

W $_{size}$ (m) = MAX ((W $_{size}$ (previous) – ΔW $_{decrement}$), 0)

End

RTT $_{predicted}$(m+1) prediction was fairly accurate and improved with the number of persistence requests, since the coefficients of the polynomial regression is calculated after each persistence roundtrip. Figure 29 shows the predicted value and actual value for each roundtrip time calculated for object modification frequency 10req/s in GPRS network 500ms, 20-50kb/s.

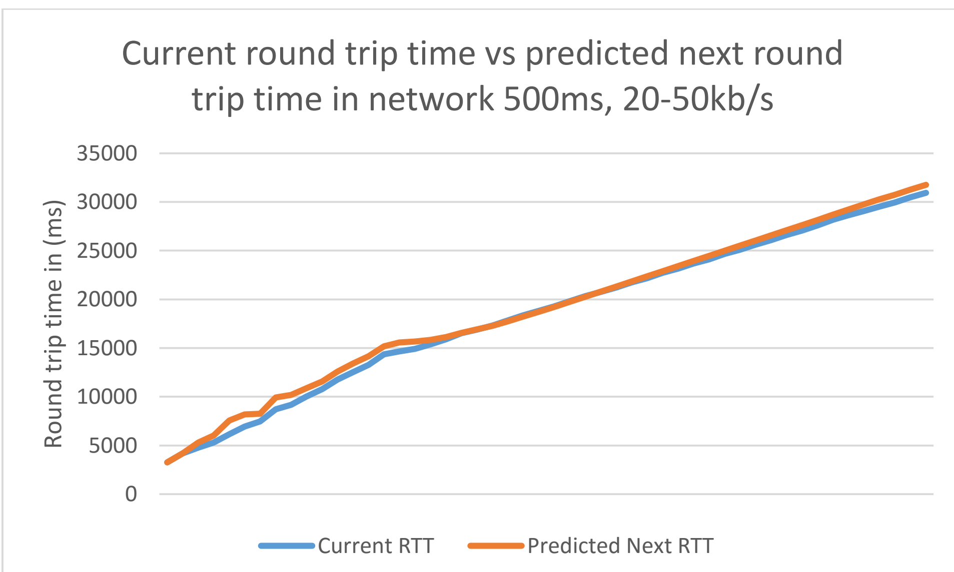

Figure 29: Current round trip time vs predicted next round trip time

4.3.2 Dynamic persistence window experiment results

Similar to fix persistence window experiments, GPRS network performance of 500ms, 50-250kb/s and persistence frequency of 10 object modification requests per second is used for dynamic persistence window experiments.

After carrying out the experiment with dynamic persistence window calculation, it was observed that, if the request frequency is high compared to the persistence round trip time, too many requests are being sent initially (Before persistence window size is calculated) which increases the persistence round trip times of all the object persistence requests, overall not getting the optimal results we found in the fixed persistence window.

Therefore, different initial persistence window values are set to compare dynamic persistence window with fixed persistence window as shown in Figure 30.

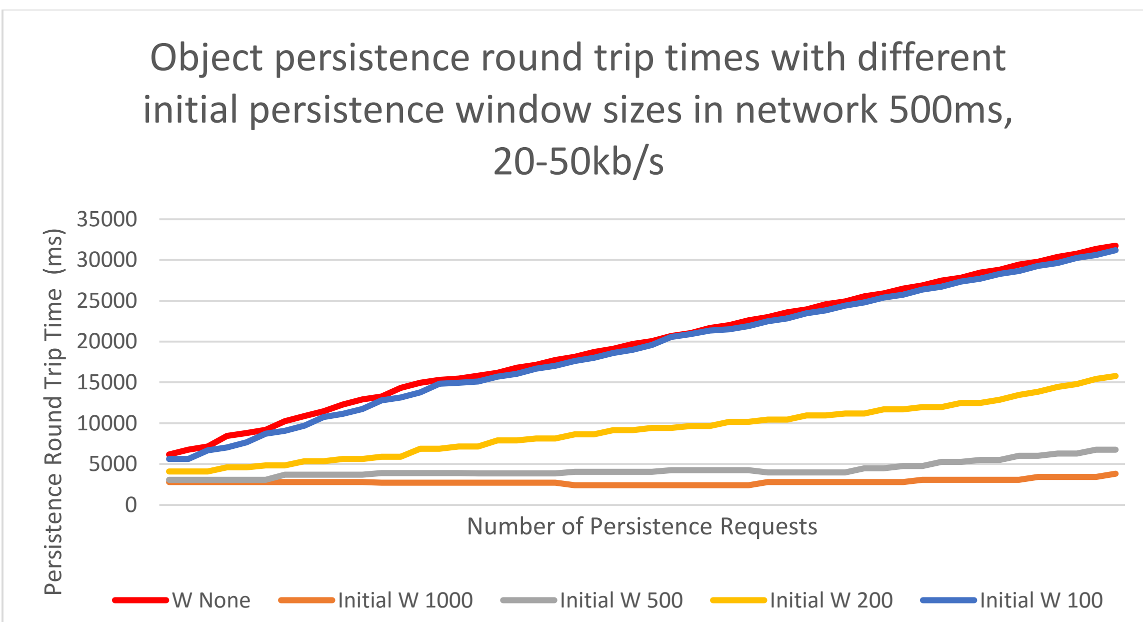

Figure 30: Object persistence round trip times with different initial persistence window sizes in dynamic persistence window

When considering fixed persistence window, with dynamic persistence window round trip times, both provide better round trip times in comparison to without having any persistence window. Also the dynamic persistence window round trip times are much closer to the respective initial persistence timing.

In this experiment the persistence workload is simulated with a fixed high frequency. According to the Figure 31, dynamic persistence window model, reduces the total persistence time of an object when persistence window size increases. This is because, with the large persistence window size in compared to the fixed persistence window, all the objects need to wait a maximum window size before the actual persistence round trip happens.

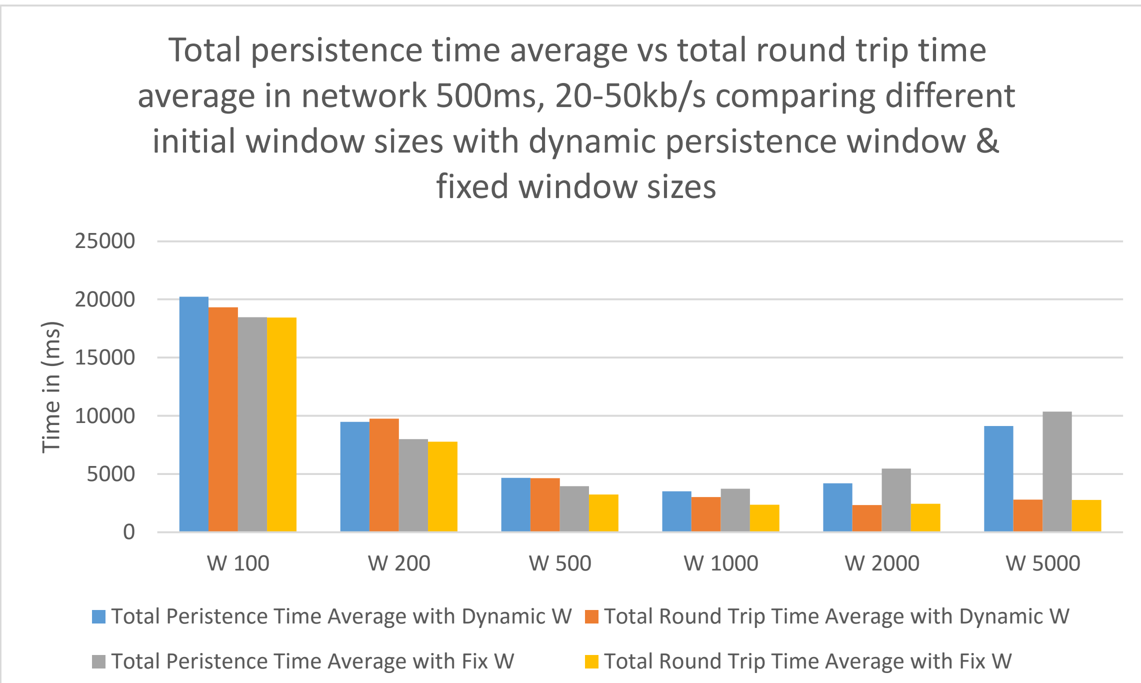

Figure 31: Total persistence time average vs total round trip time average in fixed window & dynamic window persistence models

4.4 Conclusions & future work

Adding a persistence window greatly helps to reduce object persistence round trip times, especially in slow networks and higher object modification frequencies. In comparison to fixed persistence window model, the dynamic persistence window model is more suitable for web applications since it provides less overhead for low object modification frequencies while reducing network congestion and persistence service usage for higher object modification frequencies.

However, for web applications like games, that possible to have initial higher frequencies of object persistence, it is required to select an initial persistence window. Further research needs to be carried out, using the experimental data to select a suitable initial persistence window for web applications.

To further improve the dynamic persistence window calculation algorithm, it requires to deploy the adaptive persistence window JavaScript framework developed, for functioning web applications and collect operational data similar to the experimental data collected in this research. This data can be further analyzed to adjust the algorithm parameters such as round trip time threshold, increment and decrement to further improve its performance.

Furthermore, this research can be extended to support not only object persistence, but also for other high frequency web application, request congestion control by combining all the requests arrived within a window and sending them to the web service at the end of the window time interval.

References

[1] M. Anttonen, A. Salminen, T. Mikkonen and A. Taivalsaari, "Transforming the Web into a Real Application Platform: new technologies, emerging trends and missing pieces," *ACM Symposium on Applied Computing*, pp. 800-807, 2011.

[2] S. Worlikar and F. Silva, AWS Storage Services Overview, 2015.

[3] D. Agarwal and S. K. Prasad, "AzureBench: Benchmarking the Storage Services of the Azure Cloud Platform," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, Atlanta.

[4] S. Bauer, D. Clark and W. Lehr, "Understanding broadband speed measurements," 2010.

[5] A. Lyengar and J. Challenger, "Improving Web Server Performance by Caching Dynamic Data," in *USENIX Symposium on Internet Technologies and Systems*, Monterey, California, USA, 1997.

[6] C. Garrod, A. Manjhi, A. Ailamaki, B. Maggs, T. Mowry, C. Olston and A. Tomasic, "Scalable Query Result Caching for Web Applications," in *VLDB `08*, Auckland, New Zealand, 2008.

[7] N. P. Huy and D. vanThanh, "Evaluation of mobile app paradigms," in *Mobile Computing & Multimedia*, Bali, Indonesia, 2012.

[8] W. West and S. Pulimood, "Analysis of Privacy and Security in HTML5 Web," *Journal of Computing Sciences in Colleges,* vol. 27, pp. 80-87, 2012.

[9] M. Laine, "Client-Side Storage in Web Applications," 2012.

[10] R. Chandra, P. Gupta and N. Zeldovich, "Separating Web Applications from User Data Storage with BSTORE," in *USENIX conference on Web application development*, 2010.

[11] A. M. D. K. S. M. E. Benson, "Sync Kit: A Persistent Client-Side Database Caching Toolkit for Data Intensive Websites," in *International World Wide Web Conference*, Raleigh, North Carolina, USA, 2010.

[12] Google, "Google Gears," 31 May 2007. [Online]. Available: https://code.google.com/p/gears/. [Accessed 5 March 2014].

[13] Z. Mccormick and D. Schmidt, "Data Synchronization Patterns in Mobile Application Design," in *Pattern Languages of Programs (PLoP)*, Tucson, Arizona, USA, 2012.

[14] H. Shen, M. Kumar, S. Das and Z. Wang, "Energy-Efficient Data Caching and Prefetching for Mobile Devices Based on Utility," *Mobile Networks and Applications*, pp. 475-486, 2005.

[15] T. Schütt, F.Schintke and A. Reinefeld, "Efficient Synchronization of Replicated Data in Distributed Systems," *Springer-Verlag*, pp. 274-283, 2003.

[16] S. Agarwal, D. Starobinski and A. Trachtenberg, "On the scalability of data synchronization protocols for PDAs and mobile devices," *Network, IEEE*, vol. 16, no. 4, pp. 22-28, 2002.

[17] B. Cannon and E. Wohlstadter, "Automated Object Persistence for JavaScript," in *International World Wide Web Conference*, Raleigh, North Carolina, USA, 2010.

[18] A. Hosking and J. Chen, "Mostly-copying reachability-based orthogonal persistence," *Object-oriented programming, systems, languages, and applications*, pp. 382-398, 1999.

[19] J. A. Hoxmeier and C. DiCesare, "System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications," in *Americas Conference on Information Systems*, California, 2000.

[20] C. Anderson and M. Wolff, "The Web is Dead – Long Live the Internet," *Wired Magazine,* pp. 118-127 & 164-166, 2010.

[21] S. Mavrody, Sergey's HTML5 & CSS3 Quick Reference. 2nd Edition, Belisso Corp., 2012.

[22] D. Bermbach and J. Kuhlenkamp, "2.3 Consistency in Distributed Storage Systems an Overview of Models, Metrics and Measurement Approaches," *Springer,* vol. 7853, pp. 175-189, 2013.

[23] S. Obrutsky and E. Erturk , "Multimedia Storage in the Cloud using Amazon Web Services: Implications for Online Education".