



**EXPLOITING TEMPORAL QUERY BEHAVIOR FOR IMPROVING
RESULT CACHE ACCURACY IN WEB SEARCH ENGINES**

SAFAA JUMAAH WAJJI

DECEMBER 2016

**EXPLOITING TEMPORAL QUERY BEHAVIOR FOR IMPROVING RESULT CACHE
ACCURACY IN WEB SEARCH ENGINES**

**A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY**

**BY
SAFAA JUMAAH WAJJI**

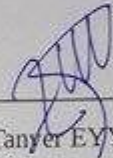
**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF
COMPUTER ENGINEERING**

DECEMBER 2016


Title of the Thesis: **EXPLOITING TEMPORAL QUERY BEHAVIOR FOR IMPROVING RESULT CACHE ACCURACY IN WEB SEARCH ENGINES.**

Submitted by **SAFAA JUMAAH WAJJI**


Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.


Prof. Dr. Halil Tanyer EYYUBOĞLU
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

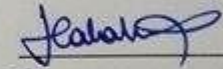
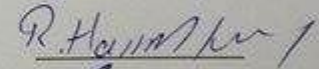

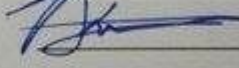

Prof. Dr. Müslim BOZYİĞİT
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.


Assist. Prof. Dr. Abdül Kadir GÖRÜR
Supervisor

Examination Date: 23.12.2016

Examining Committee Members

Assoc. Prof. Dr. H. Hakan MARAŞ	(Çankaya Univ.)	
Assist. Prof. Dr. Reza HASSANPOUR	(Çankaya Univ.)	
Asst. Prof. Dr. Gönenç ERCAN	(Hacettepe Univ.)	
Assist. Prof. Dr. Tayfun Küçükylmaz	(TED Univ.)	

STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : SAFAA WAJJI

Signature



Date

: 23.12.2016

ABSTRACT

EXPLOITING TEMPORAL QUERY BEHAVIOR FOR IMPROVING RESULT CACHE ACCURACY IN WEB SEARCH ENGINES.

WAJJI, Safaa

M.Sc., Department of Computer Engineering

Supervisor: Asst. Dr. Abdül Kadir GÖRÜR

Co-Supervisor: Asst. Dr. Tayfun Kucukyilmaz

Dec 2016, 43 pages

In Web Search Engines responding to the user queries in a timely fashion is an important requirement. One of the integral techniques to improve the response time of a search engine is caching. By storing different types of information in a fast access memory storage, caching achieves a higher availability and better response times for the search engine.

Due to anonymous and global access pattern of the queries, search engines are often considered timeless frameworks. That is, search engine sites constantly respond to queries that are submitted all around the world at an almost constant pace throughout the day. During our studies, we evaluate this phenomenon and come to the conclusion that each of the data centers, which in cooperation form the general infrastructure of a general purpose search engine in fact realizes high levels of query temporality. In this work, we aim to apply and exploit the temporal behavior of the submitted queries to improve the cache accuracy by proposing a new caching architecture. To this end, we improved the state-of-the-art result caching framework Static-Dynamic Cache (SDC) and modified it in order to incorporate query temporality. Our experiments show that the proposed caching framework improves the hit rate of a result cache up to 3%, which is roughly 25% of the possible room for improvement.

Keywords: Result Caching, Search Engines, Caches.

ÖZ

ARAMA MOTORLARINDA SORGU ZAMANSALLIGINI KULLANARAK CEVAP ONBELLEGI ENIYILEME

WAJJI, SAFAA

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi: Yrd. Doç. Dr. Abdül Kadir GÖRÜR

Aralık 2016, 41 sayfa

Arama motorları için kullanıcı sorgularına hızlı ve zamanında cevap verebilmek önemli bir gereksinimdir. Cevap zamanını iyilemek için yaygın olarak kullanılan temel tekniklerden biri cevap sayfalarının önbelleklenmesidir. Birçok farklı bilgiyi hafıza gibi hızlı erişim yeteneğine sahip bir yapıda saklayarak önbellekler arama motorlarına daha kabul edilebilir bir cevap zamanı elde etme olanağı sağlarlar.

Kullanıcı sorgularının anonim ve küresel yapısından dolayı arama motorları yaygın olarak zamana bağlı olmayan yapılar olarak algılanmışlardır. Yani, ara motoru siteleri sürekli olarak kendilerine gönderilmiş olan dünyanın her yerinden gelen sorgulara gün boyu cevap vermektedirler. Araştırmalarımız sırasında, bu fenomeni inceledik ve veri merkezlerinin, yani bir arama motorunun altyapı taşlarının, yüksek seviyede sorgu zamansallığının etkisi altında kaldığını farkettilik. Bu çalışmamızda amacımız sorgu zamansallığını kullanan yeni bir önbellek yapısı önermek ve bu şekilde önbellek başarısını artırmaktır. Bu amaçla en gelişkin teknoloji olarak kabul edilen Statik-Dinamik Önbellek (SDC) inceleyerek üzerine sorgu zamansallığını kullanan değişiklikler önerdik. Yapılan deneyler önerdiğimiz önbellek yapısının önbellek cevaplama oranını %3 kadar artırdığını göstermektedir ki bu yapılabilecek maksimum eniyilemenin neredeyse %25'ine denk gelmektedir.

Anahtar Kelimeler: Sonuç Önbelleği, Arama motorları, Önbellekler.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisors Dr. ABDUL ALKADIR GORUR of the Computer Engineering Department at Cankaya University and and Dr. Tayfun Küçükyilmaz of the Computer Engineering Department at TED University, without their helpful advices, valuable comments and guidance this thesis could not be completed. Their doors was always open for me whenever I need help from them. I want to thank my family for their support. Finally, I would like to thanks my friends, teachers for every thing.

Table of Contents

STATEMENT OF NON PLAGIARISM.....	iii
ABSTRACT.....	iv
ÖZ.....	v
ACKNOWLEDGEMENTS.....	vi
TABLE OF CONTENTS.....	vii
LIST OF FIGURES.....	ix
LIST OF TABLES.....	xi

CHAPTERS:

1. Introduction.....	1
2. Related Work.....	5
2.1 Result Caching	5
2.2 Static Cache and Dynamic Cache.....	7
2.3 Static Dynamic Caching (SDC).....	8
3. The Motivation.....	11
3.1 Search Engine.....	11
3.2 The Caching Process	12
3.3 Query Compatibility.....	16
4. Techniques and Models	20
4.1 Static-Dynamic Result Caching	20
4.1.1 Static Cache.....	20
4.1.2 Dynamic Cache	21
4.2 Semi-Static Cache.....	21
4.3 Modified Query Frequency.....	24
5. Setup	26
5.1 Static Cache Setup	26

5.2 Dynamic Cache Setup	26
5.3 Semi-Static Setup.....	26
5.4 Final Setup.....	27
6. Experimental Results.....	29
6.1 Static-Dynamic Cache Results.....	29
6.2 SSDC Results.....	31
7. Conclusions.....	43
REFERENCES.....	R1

LIST OF FIGURES

FIGURES

Figure (1.1)	Search Engine Components.....	2
Figure (1.2)	The Inverted List.....	3
Figure (2.1)	LRU Cache Mechanism.....	9
Figure (3.1)	Geographically Distributed Data Centers.....	11
Figure (3.2)	Data Centers Components.....	12
Figure (3.3)	Search Engine Components.....	13
Figure (3.4)	Static Cache Query Composition (10% of the total cache capacity is reserved as a static cache).....	17
Figure (3.5)	Static Cache Query Composition (20% of the total cache capacity is reserved as a static cache).....	18
Figure (3.6)	Static Cache Query Composition (30% of the total cache capacity is reserved as a static cache).....	18
Figure (3.7)	Static Cache Query Composition (40% of the total cache capacity is reserved as a static cache).....	19
Figure (4.1)	Day Query Occurrences During a Day.....	22
Figure (4.2)	Night Query Occurrences During a Day.....	23
Figure (4.3)	Every Time Query Occurrences During a Day.....	24
Figure (6.1)	The Hit Rate for SDC(10% as static, 90% as dynamic) and SSDC(10% as static, 40% as semi-static and 50% as dynamic).....	32
Figure (6.2)	The Hit Rate for SDC(20% as static, 80% as dynamic) and SSDC(20% as static, 30% as semi-static and 50% as dynamic).....	33
Figure (6.3)	The Hit Rate for SDC(30% as static, 70% as dynamic) and SSDC(30% as static, 20% as semi-static and 50% as dynamic).....	34
Figure (6.4)	The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(40% as static, 10% as semi-static and 50% as dynamic).....	35
Figure (6.5)	The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(10% as static, 30% as semi-static and 60% as dynamic).....	37

Figure (6.6)	The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(20% as static, 20% as semi-static and 60% as dynamic).....	38
Figure (6.7)	The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(30% as static, 10% as semi-static and 60% as dynamic).....	39
Figure (6.8)	The Hit Rate for SDC(30% as static, 70% as dynamic) and SSDC(10% as static, 20% as semi-static and 70% as dynamic).....	40
Figure (6.9)	The Hit Rate for SDC(30% as static, 70% as dynamic) and SSDC(20% as static, 10% as semi-static and 70% as dynamic).....	41
Figure (6.10)	The Hit Rate for SDC(20% as static, 80% as dynamic) and SSDC (10% as static, 10% as semi-static and 80% as dynamic).....	42
Figure (6.11)	The Hit Rate For Each Cache With Optimal Values For Each cache..	42

LIST OF TABLES

Tables

Table (3.1)	Main Statistics Of AOL Data Set.....	14
Table (5.1)	Static, Semi-Static and Dynamic Cache Sizes.....	27
Table (6.1)	Table Of Terms.....	30
Table (6.2)	Static-Dynamic Caching (SDC) Results For Different capacities.....	31
Table (6.3)	SSDC Result (10% Static-40% Semi-Static-50% Dynamic).....	32
Table (6.4)	SSDC Result (20% Static-30% Semi-Static-50% Dynamic).....	33
Table (6.5)	SSDC Result (30% Static-20% Semi-Static-50% Dynamic).....	34
Table (6.6)	SSDC Result (40% Static-10% Semi-Static-50% Dynamic).....	35
Table (6.7)	SSDC Result (10% Static-30% Semi-Static-60% Dynamic).....	36
Table (6.8)	SSDC Result (20% Static-20% Semi-Static-60% Dynamic).....	37
Table (6.9)	SSDC Result (30% Static-10% Semi-Static-60% Dynamic).....	38
Table (6.10)	SSDC Result (10% Static-20% Semi-Static-70% Dynamic).....	39
Table (6.11)	SSDC Result (20% Static-10% Semi-Static-70% Dynamic).....	40
Table (6.12)	SSDC Result (10% Static-10% Semi-Static-80% Dynamic).....	41

CHAPTER 1

INTRODUCTION

The importance of computers in our daily lives is increasing every day, especially with the expansion of data digitization, social networking and computerized communication technologies, and with the advent of the Internet and data sharing technologies. Many major research fields are gaining attention. One of the most important advances in our daily lives is that of web search engines (WSE). Today, WSE is a multi-billion dollar information industry and everyone all around the globe constantly uses WSE services. With such a large industry, improving the speed and response time of a search engine is of almost equal importance to that of improving the quality of information. Many web search engines work in a distributive manner and are composed of several sites across the world that are geographically distributed. Most of time, it is beneficial that these geographically distributed sites serve their respective regions, meeting the demands of local users in order to reduce the response time of a query. When a query is requested by a local user, the closest site will reply to the requested query. Sometimes in large search engines in order to improve throughput and response time several queries can be forwarded to a far away data center, this operation is commonly known as query forwarding.. In case of query forwarding, the response time of a query can become longer than the quality of service requirements of the web search engines. , which can lead to users disappointment and dissatisfaction due to high response time.

A typical search engine can be conceptually divided into several parts; namely Web Crawler, Indexer, cache and Query Processor (Fig. 1.1). Web Crawler (or Spider) is software or executable script for searching for and fetching web pages that have been visited by users. It parses those pages in a data structure of extracted links of the pages depending on such factors of quality including the number of requests of a page that are gathered by a web crawler and the importance and popularity of a page in store as well as the download rate.

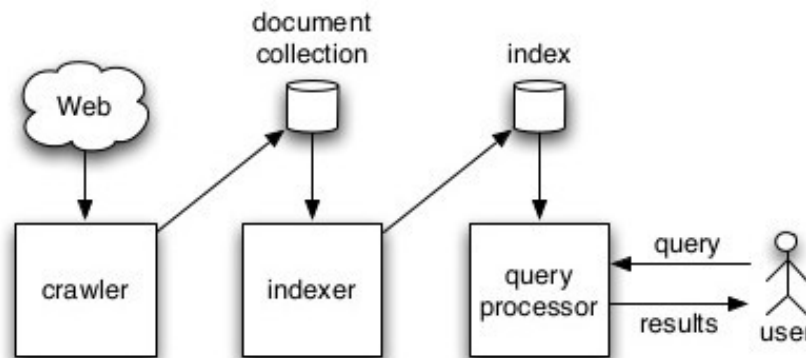


Fig. (1.1): Search Engine Components

the second part; Indexer is responsible for producing suitable data structure for searching. This data structure is commonly known as an Inverted index. The inverted index is an indexed data structure and contain a mapping of the content and it also known as posting list . The Data (in the Inverted index) is represented with a dictionary of terms of a web page and an inverted list for each term in that page, the lists containing the information about the frequency and the positions of those terms (Fig 1.2). This then is the process which generates the best results matching a query requested by the users in the least time for responding. This is known as Query Processing, where sorting pages and selecting the most appropriate (i.e. high ranking) pages query processing can take several properties into account:

- The analysis of the links for a web page
- The probability of the existence of spam
- The number of clicks for a web page (number of requests by users)
- The analysis of a search activity
- The relation between a request and society
- The statistics of the terms
- The availability of a term within local sites

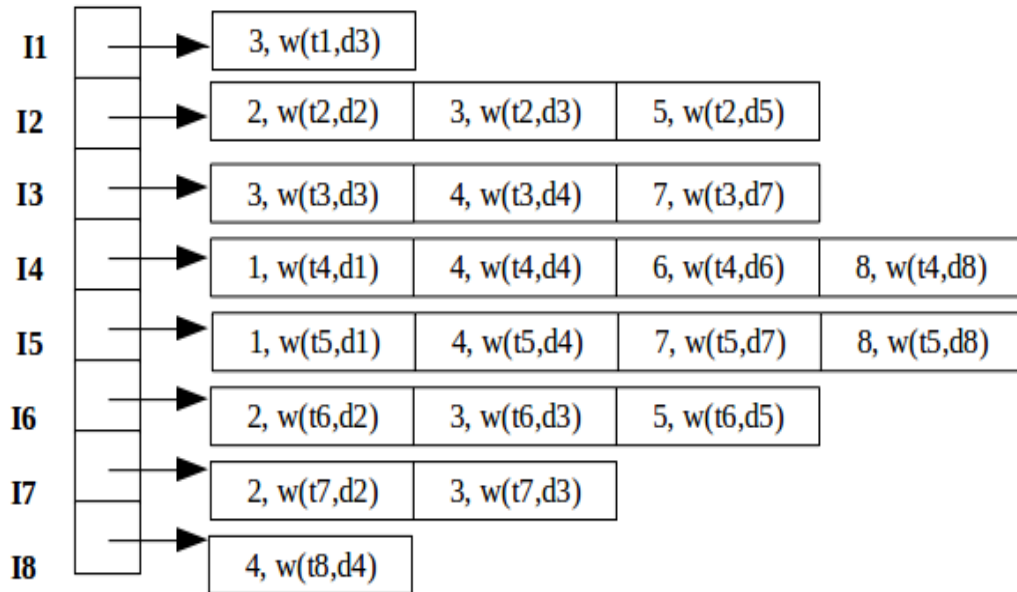


Fig (1.2): The Inverted List

The importance of this stage of the WSE process is the integral speed and efficiency of the search engine in order to improve search engine performance. A very commonly used technique is caching, which is the process of storing important information in a fast memory. Since queries or query results stored in the fast access memory can be used faster than the data residing on the disk; caching has many advantages, first it improves the response time vastly. Second it produces a high throughput on the web search engine and finally it enables search engine optimization through consumption of least computational resources. The cache stores the most requested and recently appended queries inside the storage after recomputing the data with specific algorithms dedicated to such purposes in a small amount of that storage. The efficiency of the caches is the amount of time that a request demands a reply from the user by responding to a search request in the shortest possible time. The most important metric of a query to be cached or stored in the cache is the instance number of requests for a specific query and the arrival time of a query to the cache. The synchronization between the hit rate and the arrival time is the underlying measure of the performance of the cache. The aim of caching is to reduce work load and response reaction time on the data centres of local or

global servers with numerous methods such as admission, pre-fetching, eviction and refreshing. The main major operation for the cache is the procedure of determining the query entry in the cache known as (Admission). When the cache admits a set of queries and becomes full, a number of queries will be evicted from the cache. This operation underlies the term (Eviction). For the future probabilities that appear for some queries depending on different measurements, the number of the queries is cached before their being requested, which is known as (Prefetching). For a number of reasons, such as the capacity of the cache or the validity of the queries, there is a need for a (Refreshing) technique to make a decision in those situations.



CHAPTER 2

Related Work

2.1 Result Caching

One way to improve web search engine performance is through result caching (Markatos 2001). In results caching, we process the results of many searched queries that have been requested from many users and we store them in storage with special properties including the ability to retrieve them in the least amount of time. Any future occurrence of a query that are already cached in the storage is one of the best solutions in this field. Hence, both the server and query processing performance will be high and on the server side, the load will decrease greatly due to the requests that will be served from the cache rather than the server. Moreover, the query processing will be less congested with the occurrence of any future queries that have already been cached, which will decrease the time needed to reply to a user request. Result caching is a method such that a number of sites is processed by taking the highest frequency entries, also known as the hit ratio. Whenever the hit ratio is high, performance improves; this depends on many factors, including the replacement policy in the cache as there are many replacements and refreshing techniques such as time-to-leave prediction (Alici et al., 2012). The steps of the caching starts whenever a new query enters the cache and as a result the size and number of items inside it will increase and it is here that we will need a method for replacement to avoid the overload on the cache, therefore requiring a real-time process. Moreover, the items that need to be removed from the cache are the queries with the least popularity in order to ensure that the cache will only serve the queries with the highest popularity. The researches focused on analysing the statistics of the entries. In 1998, Hoelscher [Hoelscher 1998] reached a conclusion that approximately 59% of users searched for requests within the first page of results. Moreover, in the Lampel and Moran experiment [Lampel and Moran 2003], it was found that 63.5% of users searched for results in the first page only. Similarly, 85% of users preferred the first results page in Silverstein's experiment [Silverstein et al. 1999]. The commonality between all

those experiments is the popularity of the pages within a range; in practice, the frequencies of the pages and hit rate of the cache lies under the accuracy and effectiveness of the methods and techniques that are working in the cache. There are many architectures of result caching with different results for each one of them, and each architecture aims to improve the hit rate of the cache. As a result of those attempts, there were many designs and each one of those designs dealt with the queries differently, such as the static cache and the dynamic cache. Furthermore, there are a number of hybrid designs that synchronize more than one type of cache, and all those researchers aimed at one object, offering the best cache that would serve the user in less time. In the static cache architecture, in storage with preloaded entries before setting the cache online, the entries consist of a set of queries with a high frequency from a previous log. Unlike the static cache, the dynamic cache is designed with policies to make it more flexible by offering free space by removing queries and replacing them with others according to the policy work. The first to apply time reduction for a response in search engines was Markatos [Markatos 2003], by designing a fully static cache results and comparing them with fully dynamic ones. In the first design, the queries with the highest popularity received a slight difference in many tests. On the other side, the dynamic design showed that some queries received a number of requests for only a certain amount of time in the interval appearance, and there were suddenly queries with a high number of requests, such as web page broadcasts or streamings of special events. Another result in caching design was presented by Lempel and Moran [Lempel and Moran, 2003], which provided the cache system with a new caching policy called PDC (Probabilistic Driven Caching). PDC focuses on the statistics of the queries that were submitted earlier in the search engine and any other query with no occurrence previously being considered in this policy. This policy segments the queries into two groups: the first group, that contains queries with high ranking, will receive a high probability; the second group is the queries with low probability and these queries will be evicted from the cache. Saravia designed a hybrid system from two-level caching [Saravia et al. 2001], in which the first cache is storage for the queries with an LRU (Least Recently Used) policy. The other is storage with a postings list that contains the queries inside it, wherein each query entering the first level is distributed into a posting list at the second level. From all the designs above, we found that Makrato's design is missing a pre-fetching method in order to reply to

new requests. In Saravi's cache, the second level provides good performance, which helps the memory with replying to requests, a good overall throughput with no pre-fetching strategy. The Lempel and Moran cache serves only the subsequent pages that relate to the first one. For those that are subsequent, they used an SLRU (Segmented Least Recently Used) Policy. Thus, the PDC grants a pre-fetching for the user expectation request; however, the problem of this method is that when no request is received within a period of time, the session ends and anything newly subsequent received after that will receive less priority, which might not put those subsequent items into the first results page. Another cache design (Baeza-Yates and Saint-Jean, 2003) includes a two-level cache with a static setting, Moreover, (Altingovde et al., 2011) proposed a result cache with a document ID cache, and storing the document IDs without the snippets in order to reduce the query traffic at the end of the search system. In a cache with three levels, there is a result caching and a postings list caching and the intersection of the postings list caching and the result caching (Long and Suel, 2005).

Tolosa et al. (2014) present a single static cache by making a couple of posting lists and term intersection lists by generating terms that are mined from the query logs with the top N-frequent method from posting lists cache, result cache and the term intersection cache. All these architectures can be summarized into two main designs: static and dynamic caches. In addition to those two caches, there are the hybrid designs which may contain one or both of with different policies from one design to another, such as Saravia's cache and the state-of-the-art caching of the SDC (Static Dynamic Cache) (Fagni et al. 2006). Moreover, there are the three-level architectures cache (Li et al. 2007), which proposes a cache with results, posting lists, and document caches. Additionally, there is the five-level cache that was proposed by (Marin et al. 2010) and (Özcan et al. 2012)

2.2 Static Cache And Dynamic Cache

The static is a type of cache which usually stores the most frequent subset of queries in the query logs, because of this reason the static caches in the literature are often deployed in an offline manner. In the literature features and techniques which are different than query frequency are also proposed. The entries are identified by such quality measurement such that they increase the hit ratio of the cache. On such basis, the hit rate will show the impact of the cache policies. Practically,

whenever the hit rate increases, better performance of the cache is achieved. Unlike the static design, the dynamic cache has the capability of redeveloping itself by evicting the queries inside it. By removing the queries and depending upon a policy that is suitable to it, for example, the LRU (Least Recently Used) (O'Neil 1999) (Fig 2.1) is one of those policies which depend on the recency of the queries. Most of those policies aim to offer a free space inside the memory by evicting any unimportant queries from the cache. However, even with those policies, the performance was not very accurate and occasionally there was a loss of some important data due to the behaviours of the policies. Therefore, there was a hybrid system and one of best of these systems is Fagni's cache. Most of the result caching aims to enhance and improve the performance of the search engine, Cambazoğlu proposed the last techniques to compute the unseen queries from previous logs that were already cached in order to enhance the availability of the search engine (Cambazoğlu et al. 2012). Skobeltsyn showed in his paper (Skobeltsyn et al. 2008) the impact of result caching on a dynamic cache with index pruning. Frances (Frances et al. 2014) created a multi-site web search setting by combining result caching with replication and query forwarding. Puppin et al. (2010) incremented the caching policies in order to reduce the workload of a distributed search architecture that is collection selection based. The refreshing method is an additional factor to the result caching performance (Cambazoğlu et al. 2010; Jonassen et al. 2012; Jacobs and Longo 2015). Using the information at the back end of system, the result cache receives notifications for updating the index and informs it about the expiration using a statistic from that information (Sazoğlu et al. 2015; Alıcı et al. 2015; Prokhorenkova et al. 2015; Blanco et al. 2015; Bai and Junqueira 2012; Brotnikov et al. 2015).

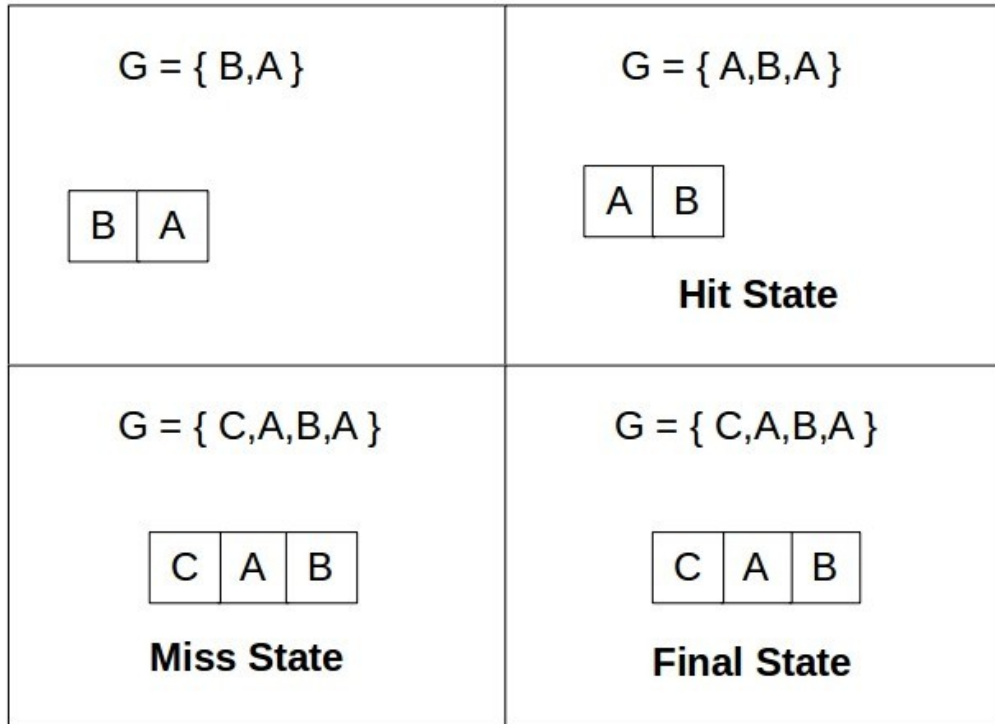


Fig. (2.1): LRU Cache Mechanism

2.3 Static Dynamic Caching (SDC)

SDC is a hybrid design of two segments of caches: static and dynamic. The static part is a block filled with a set of queries that receive a high number of requests from previous logs. The cache is filled with those queries offline and this cache will be a read-only purpose cache. In the second part of cache, there is a fully dynamic cache. The mechanism in this part is different from the static part such that all the queries yield to a replacement method and each query needs to gain more popularity in order to avoid the replacement policy and stay in the cache. In this design, Fagni used the (LRU) policy To understand dynamic caching better, we represent Q as a query and S and D for the static and dynamic parts respectively. Initially, each Q from a user logs in to the cache and the cache will search for that Q in S. If Q is found, then it is a cache hit and it will return 1 with and associated query. If Q cannot be found in S, then Q will be searched for in D. If Q appears in D, then it is a cache hit and Q's popularity will increase due to that request. Otherwise, if Q does not appear in D, then it is a cache miss and Q will be added to D as a new query, which will give Q the opportunity to compete with other queries

inside the D cache at the end. Whenever Q acquires more future occurrences, its popularity will increase. In cases when the cache is full, the LRU will delete a Q with the least popularity in order to provide space for a new Q. Every hit and miss of the cache is important because those two factors will provide the hit ratio of the cache with this equation (1):

$$\text{HitRate} = \frac{\text{No.ofhits}}{\text{No.ofhits} + \text{No.ofmisses}}$$

(1)

CHAPTER 3

The Motivation

3.1 Search Engine

WSE is software that works together to make the final virtual form of a search engine. The main problem is that search engines are geographically distributed and due to this fact, search engines serve their own regions with data centres (Fig 4.1).

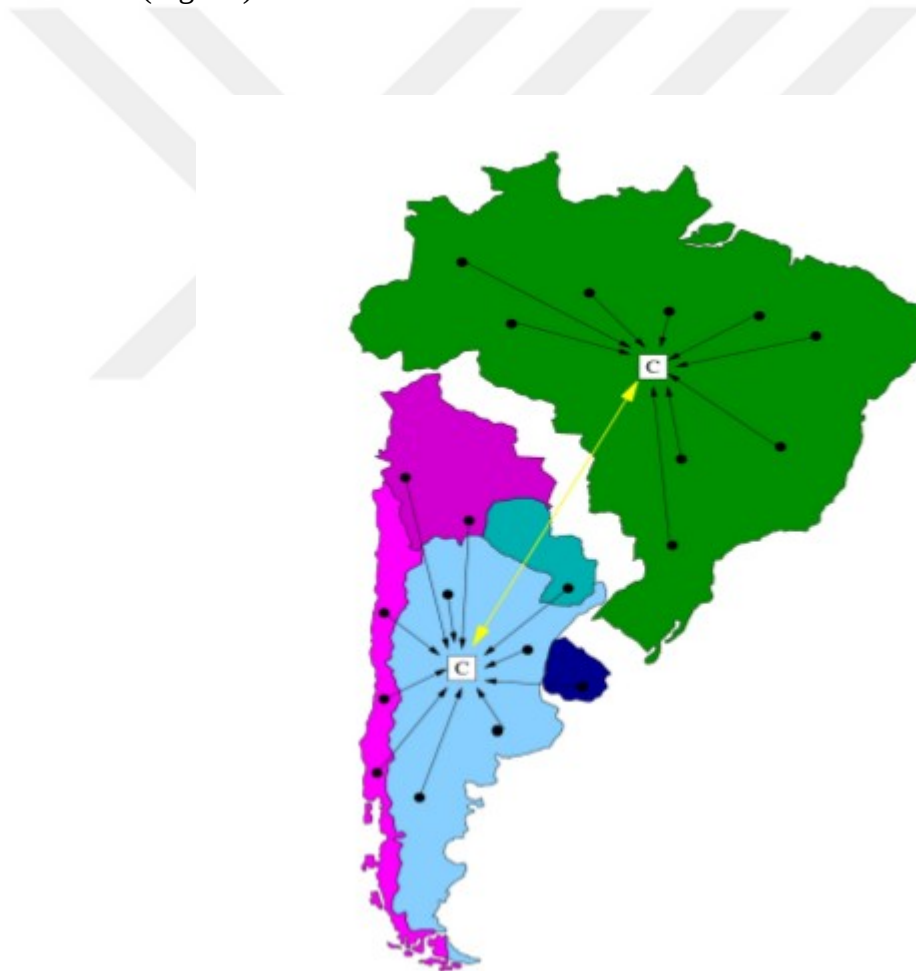


Fig (3.1): Geographically distributed Data Centres

The data centres are high powered facilities and serve users' requests. They are available at all times and the processes consume much energy (nearly 2% of the world's energy) (Fig. 3.2). Therefore, any effective method that helps to serve users perfectly with better results will help to decrease energy consumption. One of the important methods to make this possible is caching.

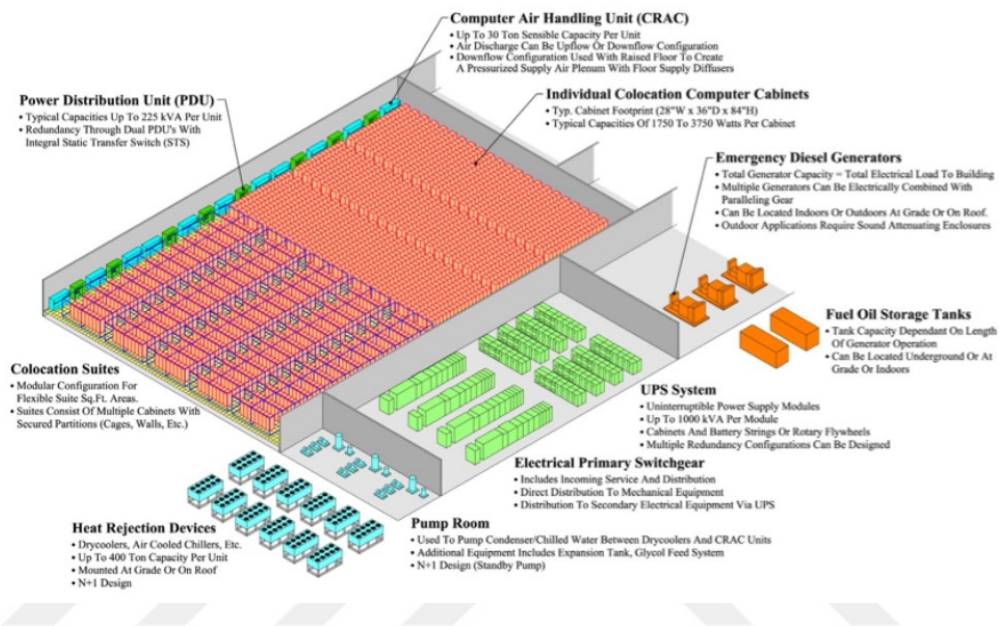


Fig (3.2): Data Centres components

3.2 The Caching Process

The concept behind caching is ambiguous for many people and some have different ideas about the sequence of the query process within the search engine. A query goes into two main options within the cache level. Every search engine is geographically distributed and when a user requests a web page, the request will reach the local server (the nearest main server to the user). One of these techniques is Duplication, the method which puts many copies of a web page into many data centre servers. Every time a user asks for that page, the page will be served from the nearest data centre to that user. Another method is query forwarding, which can be considered to be a complementary phase for the duplication method whenever a server cannot serve a user request due to overloading, the forwarding operation sends requests to another server to serve it.

Because of this, the search engine starts working on the parts to increase the speed of the search and retrieve the queries that feed users requests that depend on the caching (Fig. 3.3), especially the result caching. With this cache, researchers aim to design and improve their own cache systems.

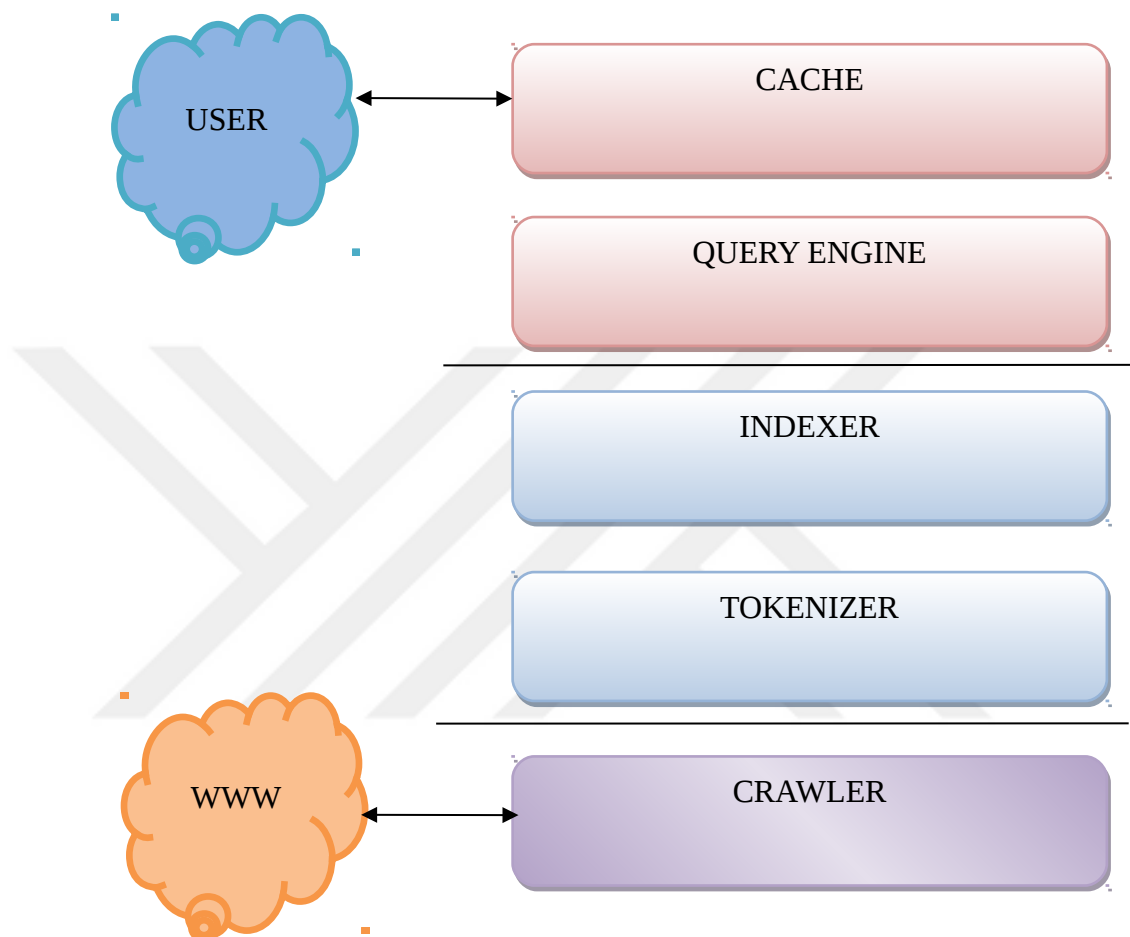


Fig (3.3) : search engine components

In this paper, we focused on a new term in the cache system, which we called the **Query Compatibility** (Kucukyilmaz et al. 2016). With this idea, we give the cache more flexibility to admit more queries inside it with more information to occur in the cache in the most compatible place that gives more replies to the user requesting it. We start our test with a dataset called AOL and this dataset belongs to America Online Inc. and it was suitable to conduct research and tests on it since it has good properties such as the huge number of website requests and the dates and

times for those requests. We modified the AOL dataset by removing a number of duplicated queries due to many simultaneous requests from many users. After this process, we had on each line in the dataset (query, date and time), where query corresponds to the website requested by a user, date and time showed the time at which the query was requested. We removed the useless queries that had been ciphered because of their content and privacy issues. After all the normalization processing, we obtained the information as seen in (Table 3.1).

<i>Dataset</i>	<i>No. of queries</i>	<i>No. of distinct</i>	<i>Date</i>
AOL	2747923	1216371	August 4 2006

Table 3.1: Main statistics of AOL data set

We found that most of the cache systems served the user with two evaluation criteria, namely frequency and recency; but with our test results, we observed that many queries produced a high number of occurrences in a period of time and decreased in another. Initially, we started a test to show the baselines for all the distinct queries in the AOL dataset and we checked the number of occurrences within one day. We found that the baseline started increasing slightly at some points in the first quarter of the baseline and in the second quarter the increments more greatly than the first quarter at some points. However, the range of the occurrences for most of the queries within the second and the third quarter were higher than the others until the fourth quarter. Then the baseline continued with same scenario as the first quarter with some increments higher than the first quarter. After that, we modified our dataset and added many statistics to our queries (total_freq, day_freq, night_freq), where total_freq corresponds the total number of hits for a query either in the day or night, where day_freq represents the number of hits for the day and night_freq determines the total number of hits for a query within the night hours. From that, we produced a result wherein most of the modern cache systems focused on the frequency with a high number of requests and ignoring queries with smaller numbers of occurrences (first and the final quarter of the baseline). This led us to conclude that those queries can admit the cache at some level and might not enter another. However, the problem here is that a number of requests for a specific query

occurs at certain times, which is the underline of the Query Compatibility term. This query might be important for a certain time and it can be requested by a number of users and not be ignored as it may be important and enhance the hit ratio of the cache. As an example of the modern state-of-art result caching system, we have Fagni's SDC. Fagni focused on the queries with the highest frequencies from previous logs. He put them inside the static part and the remaining queries competed with each other in order to be admitted into the dynamic part. The main factor in the static part is the frequency and the dynamic is the recency of the queries. In order to explain the amount of processing required to deal with this, we analysed the query and put the total frequencies into each percentage of occurrences for the day.

After checking the number of occurrences in our data set to check the number of processes that our design will deal with, we found that the queries with a low number of occurrences have acceptable frequencies and the number of frequencies can provide a good number of hits that enhance the hit ratio of the cache. With all the information that we gained from all the tests above we reached a result, that the focusing on the highest frequency queries provide a high number of hits. especially that the static cache work depending on this mechanism and the first step in the static cache is that the cache will be filled with the highest frequencies queries from a previous log offline before putting the cache in the online mode.

The benefits of using the static cache are many, one of the main benefits is that the cache is already filled with queries and that will help to avoid the compulsory misses for the queries. Also providing a high number of hits to the cache system and avoiding the amount of processing comparing to the other caching systems. Moreover, putting those queries in the static cache will provide a high speed for retrieving them to the user since they receive a high number of requests, which makes them popular for a large number of people, especially social media sites as they almost always appear due to people's behaviour when requesting them, which depending on many factors that depend on their relationships and concerns. In the final analysis, the static cache will provide a great number of hits to any cache containing it. However, the static cache alone is insufficient because, as mentioned previously, the static part is a read-only part and it will not provide any opportunity to refresh it. In order to repair this, there is the dynamic part which focuses on the queries with high frequencies for a period of time, especially the LRU methods on which SDC depends in the dynamic part. All the queries will admit the dynamic

cache regardless of whether it starts empty. However, that any new query admits the dynamic cache will receive a compulsory miss, and any query admitted to the cache needs to receive more occurrences to avoid the eviction method. The good point in the dynamic is that the queries with a high number of frequencies will not stay in the cache unless it gains more requests, thereby avoiding the cache **Hang Over** Problem, which means that any query receives a high number of frequencies for a short time, similarly to the attack on 9/11 which received a huge number of frequencies. However, by the time it started to decrease, the number of frequencies; no query inside the cache will receive such a high number of requests such as the 9/11 attack; therefore because that we use the dynamic cache system. No query will receive a high number of requests. Such a query has a high probability of not being evicted from the cache if it becomes full and a new query comes to the cache. With those two caching systems working together, we can provide a good hit ratio and cover many user requests in addition to giving the query more flexibility to admitting the cache with low frequencies.

3.3 Query Compatibility

In most of the cache systems, the researchers used a real data set such that they used a data set from a real search engine with real logs and full statistics. Since we could not provide a real data set, we used the AOL data set, which is sufficient for testing the cache systems; however, it is missing a number of other statistics that make the research and tests more effective and accurate. Nevertheless, we made many modifications to the data set in order to make it suitable for our work. We observed that the query contained a temporality, which means that the query is able to acquire a good number of occurrences for a short time. We concluded that the query can be categorised into three types query that receives most of its occurrences at the day hours called (Day Query) and the query that has the highest number of occurrences in the night hours, which is called a (Night Query) and the query with a high number of occurrences most of time, either in the day or night hours, is called the (EveryTime Query). Moreover, we found that the cache systems, specifically the static cache, focused on the every time queries and stored them, and the dynamic part gave an opportunity for them to be admitted inside it. However, there were many queries that received satisfying frequencies for a specific time to be admitted. However, after that, its was evicted when that period of time ended and on the

second day, the cache executed the same procedure with the query. As a result, the number of misses increased, which meant that this query showed a temporality that can give the cache a better hit ratio when we used a good way to utilize it. We also analysed our data set in order to understand the behaviour of the query with different cache sizes. We tested it with only fully static caches, the first being at 10% size of the static cache from the AOL data set (Fig 3.4).

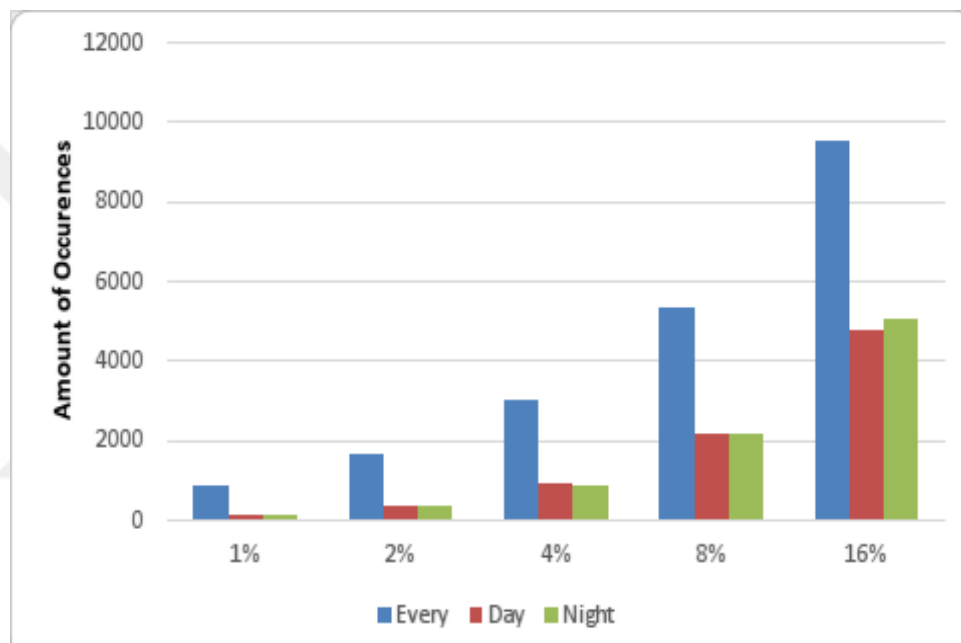


Fig (3.4): Static Cache Query composition
(10% of the total cache capacity is reserved as a static cache)

With 10% of the static cache, we can see in the cache with different capacities we found that with 1% of the capacity, the number of every time queries is more than 70% of the 10% of the static cache, and when we continue our test with higher capacities, 2%, 4%, 8%, 16% of the value of every time queries decrease and the values of the day and night queries increase, thus proving our theory about the query compatibility even with more than 10% of the cache.

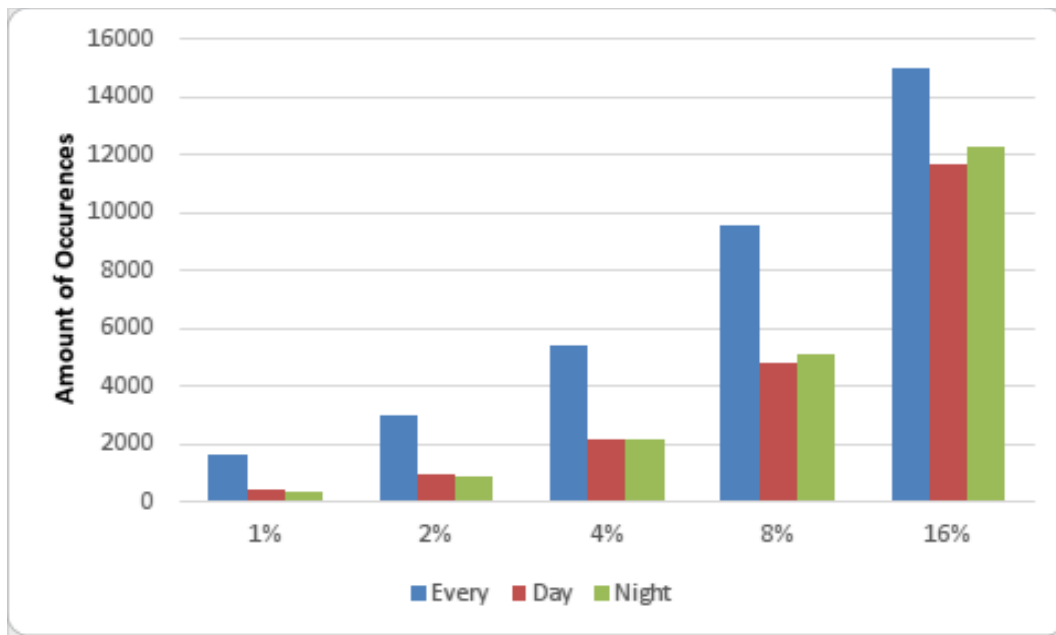


Fig (3.5): Static Cache Query composition
(20% of the total cache capacity is reserved as a static cache)

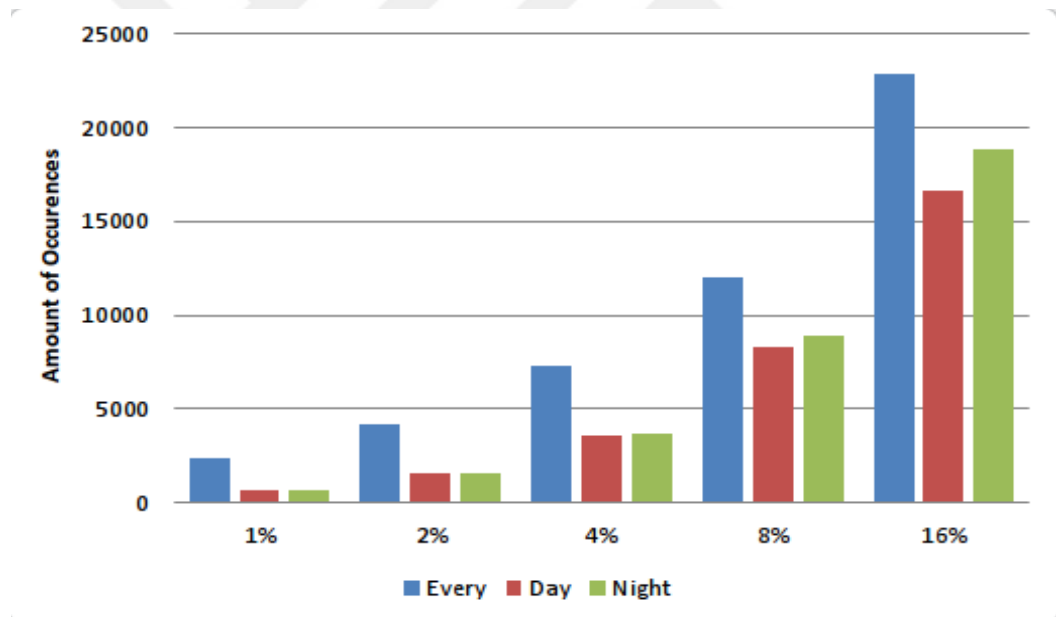


Fig (3.6): Static Cache Query composition
(30% of the total cache capacity is reserved as a static cache)

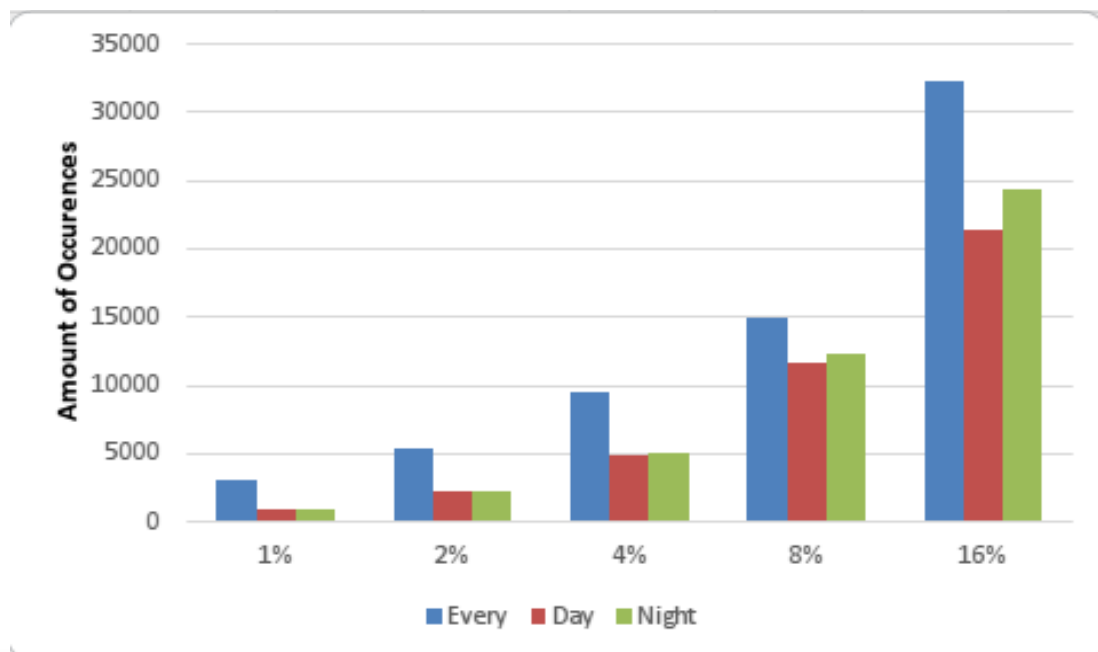


Fig (3.7): Static Cache Query composition
 (40% of the total cache capacity is reserved as a static cache)

For the 20%, 30% and 40% of static caches, the increments of day and night queries increased every time we increase the size of the cache and the capacity (Figs. 3.6 and 3.7).

We checked the test for only those values because after the 40% of static the within the SDC, the performance will decrease and the cache will start to be more static than an SDC. Moreover, any size greater than that will be unreasonable and perform badly for the cache. Additionally, there are many queries with identical properties for the day and night hours. in order to provide a good cache system in the present time, it is necessary to focus on the powerful points of modern cache systems and the SDC is considered the best of them. We already explained what it can and cannot do and we conduct a search on the query characteristics in a manner to affect any cache system enhancement. We will work in a space within the essence of the SDC; however, we will also concentrate on the weak points of the SDC in a manner so as to provide a better hit ratio by serving out the cache with query compatibility without losing any query that can make our cache more effective and a complementary part of the SDC.

CHAPTER 4

Techniques and Models

In this part, we introduce the solution by presenting the work progress with the AOL data set and the designs and techniques that we used within our research. Moreover, we implemented the static-dynamic caching with the same method that Fagni used with his own SDC. We present each method in one part with the same file as a data set and make every process on it depend on the cache sequence. The SDC progress measures the accuracy and performance of our cache by comparing the SDC result with ours, followed by checking the final result of each cache with the same data set.

4.1 Static-Dynamic Result Caching

Static-dynamic caching (Fagni et al., 2006) is a state-of-the-art caching method in the current time. We establish it by creating two separate parts and synchronizing them to obtain the baseline. Moreover, we provide many states for SDC to check the worst case and the optimal case by measuring the highest hit rate and the lowest hit rate. For this purpose, we run the SDC at different sizes by giving different percentages for the static and dynamic segments in each case.

4.1.1 Static Cache

We started our work by filtering the AOL data set by removing the unnecessary queries that are ciphered for a number of reasons from the source with the sign (-). Moreover, we removed the queries that appear with same date and time, which occurs when a number of users make simultaneous queries. Additionally, we removed any queries that were missing dates or times. After that, we calculated the number of distinct queries by adding all duplications by the query title into a field we call the FREQUENCY, which contains the number of occurrences for a query within the data set. Then, we sorted them according to the FREQUENCY field. Until this step, we did not use any of the previously explained methods since those are the

only steps that were used in the SDC. After sorting the data set, we took the queries with the highest frequencies and put them in the static part as the static cache content and calculated the total frequencies of those queries (inside the static part) as the total number of hits for this part of the SDC.

4.1.2 Dynamic Cache

In this caching part, we removed all the queries that appear in the static cache in order to conduct an accurate simulation for the SDC. Because of that, we made separate files as dynamic data sets. Then we started our test by using the LRU method as the main mechanism of the dynamic cache.

4.2 Semi-Static Cache

The semi-static is our new method that we use in the result caching which depends on a new feature for the caching. This feature was mentioned in the previous chapter as “Query Compatibility” and we focused on this feature because we found that it had never been mentioned or used previously. When we start our tests on this feature, we give it much attention and when we measured the query behaviour each time, we increased the cache sizes and capacities.

Initially, we create a data set with distinct queries, and we create compatibility between each query. Then, we provided a factor for this process and in order to provide more space to the query, we increased the time-line for the day and night queries by separating the time into two parts, the first part of the time-line being between 07:00 and 18:00 as the day period and the night time period will be between 18:00 and 07:00.

The queries that come with any time-line needs to receive 60% or more of the total occurrences within a period in order to be accepted as a query related to that time-line. Anything less than that, the query will be accepted as an every time query. We also ignored the queries that appeared only once in the data set, which we considered as unsatisfactory. To provide a clearer view about the query compatibility, we took random values that satisfied this feature. In (Fig. 4.1), we took the query (map quest) as an example of a day query.

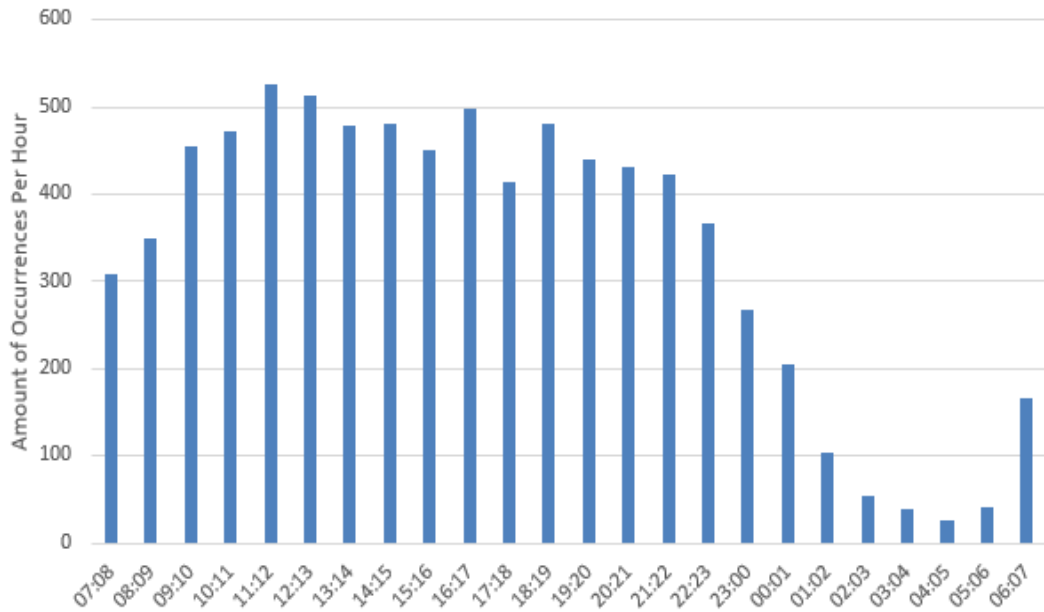


Fig (4.1): Day Query Occurrences during a day

We can easily observe that the number of occurrences starts high from the beginning of the first day hour (day period). Between 07:00 and 08:00, the query increases to more than 300 occurrences within the first hour and the occurrences continued to increase after that. From 11:00, the query reached the highest number of occurrences and continued with varying differences from 12:00. After 18:00, the occurrences started decreasing until the end of the night period. As an example for the night query, we have the query (my space) as an example in Fig. 4.2.

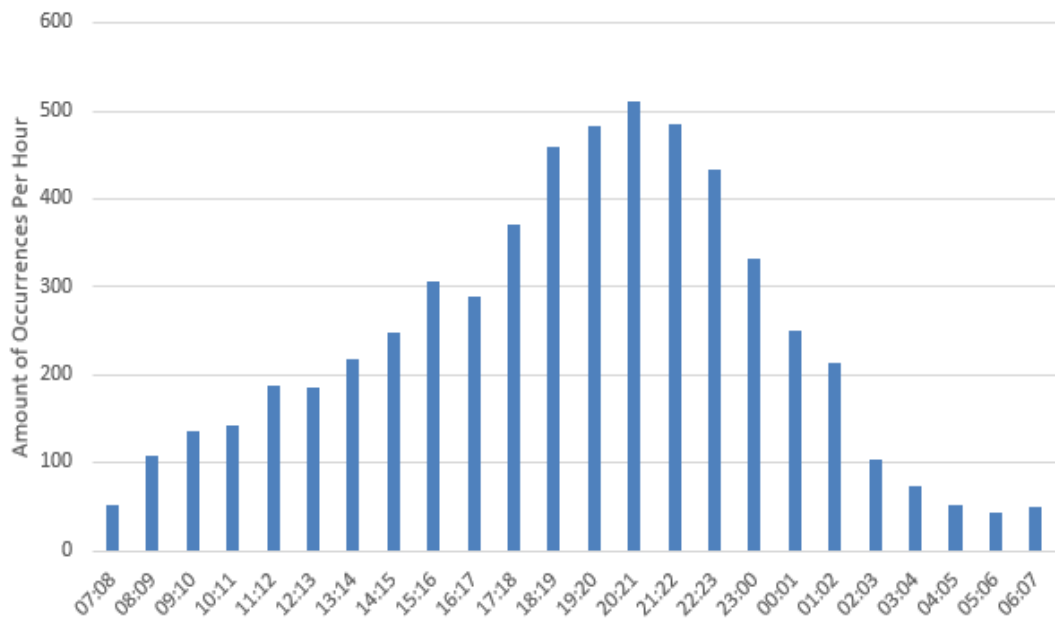


Fig (4.2): Night query occurrences during a day

With this query, we can easily classify it as a night query for the period of high activity at the beginning of the first hour in the night period starting 17:00 and lasting until the first two hours of the next day. When we compare it with the previous example, we found the number of activity hours for the previous query numbered more than the night query example. However, we need to take into consideration that the focus here is on the total number of occurrences rather than the hours of activity for the query. After those two examples, we see that the number of occurrences is higher with the every time queries and that the queries do not have a specific activity period. Moreover, have a high number of occurrences most of the time. For this example, we took one of the most frequently requested queries (Google) as an example of an every time query (Fig 4.3).

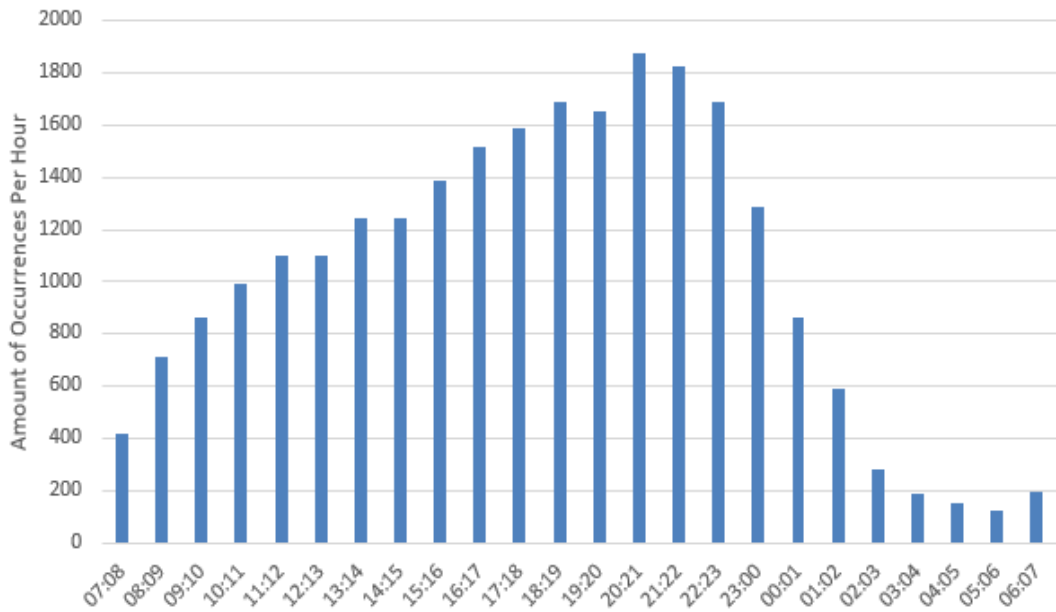


Fig (4.3): Every time query occurrences during one day

The (Google) query yields the highest query requested in our data set (AOL) in all the tests and it is more suitable to be in the static part rather than the other parts. We found a high number of occurrences from the beginning in comparison with the first hour for either the day query example (Fig 4.1) or the night query example (Fig 4.2) and almost the same or slightly below the highest occurrences for the previous examples. Most of the time, it yields a large number of occurrences, and even with the lowest occurrences, between 2:00 AM and 7:00 AM, the number of occurrences each hour of that period is very high compared to previous examples. To calculate the query compatibility, we used the following formula:

$$Q.Comp = \frac{\text{No.DayOccurrences}(n)}{\text{TotalOccurrences}(n)} \quad (1)$$

where n represent the query and the formula for the query compatibility equals the number of the occurrences for a query in the day period divided by the total number of occurrences for a query within the day and the night periods.

4.3 Modified Query Frequency

In this part, we used the modified query frequency to explain the method that we used within our semi-static cache strategy to put each query inside the correct

part. We aim to estimate future query occurrences. For example, when a query comes during the day period, there is a high probability of this query appearing in the night period. However, with the night query, the situation is different and we need to decrease the expectation for future occurrences during the day. Immediately after calculating the query compatibility, we created two files of the remaining data set after the static part queries were removed and each one of them was created with the same queries but with a different equation than the other. With the first file, we executed formula number (3), the modified query for the day queries:

$$ModifiedFreq = TotalOccurrences(n) * (Q.Comp(n) * 2) \quad (3)$$

This equation gives us the total number of occurrences for the day with a number of additional future expected occurrences for the query. Furthermore, for the night, and every time queries are made, we keep the modified frequencies equal to the total number of occurrences.

In the second process, we calculate the modified frequencies with same files; however, this time, we only served the night queries with equation (4) and we kept the other queries with the modified frequencies equal to their total occurrences.

$$ModifiedFreq = TotalOccurrences(n) * (1 - Q.Comp(n) * 2) \quad (4)$$

After finishing the work with the modified frequency for each query, we produce two files each one of which served one kind of query. Then, we sorted those files according to the modified frequency field. With those procedures, we produced all the data sets that we will use in our final synchronized simulation.

CHAPTER 5

Setup

We conducted our experiments on a PC with a 1.6 GHZ Intel Core i5 processor, 4GB of RAM and the Linux Ubuntu 15.10 operating system. We were unable to obtain a realistic data set, so we depended on the AOL data set due to it being, at that time, the best option available for the use of statistics, thus making it a good data set for experiments.

5.1 Static Cache Setup

Because of the static cache filled mechanism, we sort the AOL data set and put the n most frequent queries into the static part and keep the remaining queries in a separated file without the static cache queries. First, we start with a fully static cache, and afterwards, we decrease the cache size by small portion of 10%.

5.2 Dynamic Cache Setup

For the dynamic part, we used an LRU cache and we started the experiment with this part with a fully-dynamic cache with the remaining queries from the AOL data set after removing the static queries from it. We started with 10% and the capacities of 1%,2%,4%,8% and 16%, and we continued with 10% and increased them by another 10% for each new test until we reached a 100% fully-dynamic cache.

5.3 Semi-Static Cache Setup

This part is our novel part and the main part in our research. It contains the modified queries that have been sorted for the day and for the night. The main difference in this cache is that it has a swapping part. the first part is the part with queries that yields the highest modified frequencies depending on daytime compatibility. At the beginning of the night hours, a swap to the night part that is already filled with queries with modified frequencies depends on the night time

compatibility. The size of the semi-static cache ranges from 10% up to 40% by 10% increments for each term of tests (Table 5.1).

Dynamic Cache Size %	Semi-Static Cache %	Static Cache %
50%	10%	40%
	20%	30%
	30%	20%
	40%	10%
60%	10%	30%
	20%	20%
	30%	10%
70%	10%	20%
	20%	10%
80%	10%	10%

Table 5.1 :Static, Semi-Static and Dynamic Cache Sizes.

From the table above, for each dynamic cache size, there are many tests for more than one time with different sizes for each different size for the static and semi-static cache.

Because we could not obtain a real data set, we did the swapping and every other operation as simulations. Moreover, we conducted all those operations in order to select their optimal value and to compare our results with the SDC cache results so as to cover all the cases for the caches.

5.4 Final Setup

After filling the static cache and the swapping parts for the semi-static cache, the experiment starts by taking a query and initially searching for it in the static cache. If the query is found, then it is a hit; if the query is not found, then we search for it within the semi-static cache. For this part, the search differs such that it is less complex than the others because we have a non-realistic data set. For that, we take

the query and we also check the time of this query. The query is shown as a day query. We swap to the day part and search for it within it. If found, it is a hit, and if not found, it is a miss. Then we search for it in the dynamic cache; if the query time shows that the time is related to the night period, we swap to the night part and do the same operations as the day swapping. If a query is not found in either the static cache or the semi-static cache, we search for it in the dynamic cache. With the dynamic cache, the cache can start empty. If a query is found in the cache, then it is a hit and the node with this query will be pulled to the front of the cache because of the LRU cache mechanism. If the query is not found even in the dynamic cache, then it is a miss and we add this query to the dynamic cache with even less regard than any other information for this query. Moreover, we a put it in the front of the cache. At the end of all those searches until the end of the data set, we calculate the number of hits and misses for all the queries.

CHAPTER 6

Experimental Results

In this chapter, we present every experimental result that we obtained from each cache and compare them with one another to obtain and show the difference in performance between them. We grouped the experiments in two: Static-Dynamic caching and SSDC (Static-Semi static-Dynamic cache), in which we conducted the tests with different sizes and with capacities for the cache at 1%, 2%, 4%, 8% and 16%.

6.1 Static-Dynamic Cache Results

By taking the sum of the hits from the static cache and dynamic cache and dividing it with the total number of queries in the data set (5), we get the hit ratio for each differently sized cache.

$$HitRatio = \frac{Hits_s + Hits_D}{no.ofQueries} \quad (5)$$

Rather than making a baseline for each case for a cache, we put the results into a table for the hit rate. We start with the SDC cases and we start with a fully static cache. We then decrease the static with a partial increase of the dynamic (Table 6.2).

And to explain the terms that we will use in our setup and results we give the terms and an explanation about each one of them in the table below (Table 6.1).

Term	Explanation
50S-50D	The capacity of the static cache is 50% of the cache size (1,2,4,8,16) and 50% is the capacity of the dynamic cache for the same size.
40S-60D	The capacity of the static cache is 40% of the cache size (1,2,4,8,16) and 60% is the capacity of the dynamic cache for the same size.
30S-70D	The capacity of the static cache is 30% of the cache size (1,2,4,8,16) and 70% is the capacity of the dynamic cache for the same size.
20S-80D	The capacity of the static cache is 20% of the cache size (1,2,4,8,16) and 80% is the capacity of the dynamic cache for the same size.
10S-90D	The capacity of the static cache is 10% of the cache size (1,2,4,8,16) and 90% is the capacity of the dynamic cache for the same size.
100S-0D	The capacity of the static cache is 100% of the cache size (1,2,4,8,16) and 0% is the capacity of the dynamic cache for the same size. Which mean there is no values in the dynamic cache.
10-40-50	The capacity of the static cache is 10% of the cache size (1,2,4,8,16) and 40% for the semi-static capacity and 50% the capacity of the dynamic cache.
20-30-50	The capacity of the static cache is 20% of the cache size (1,2,4,8,16) and 30% for the semi-static capacity and 50% the capacity of the dynamic cache.
30-20-50	The capacity of the static cache is 30% of the cache size (1,2,4,8,16) and 20% for the semi-static capacity and 50% the capacity of the dynamic cache.
40-10-50	The capacity of the static cache is 40% of the cache size (1,2,4,8,16) and 10% for the semi-static capacity and 50% the capacity of the dynamic cache.
10-30-60	The capacity of the static cache is 10% of the cache size (1,2,4,8,16) and 30% for the semi-static capacity and 60% the capacity of the dynamic cache.
20-20-60	The capacity of the static cache is 20% of the cache size (1,2,4,8,16) and 20% for the semi-static capacity and 60% the capacity of the dynamic cache.
30-10-60	The capacity of the static cache is 30% of the cache size (1,2,4,8,16) and 10% for the semi-static capacity and 60% the capacity of the dynamic cache.
10-20-70	The capacity of the static cache is 10% of the cache size (1,2,4,8,16) and 20% for the semi-static capacity and 70% the capacity of the dynamic cache.

20-10-70	The capacity of the static cache is 20% of the cache size (1,2,4,8,16) and 10% for the semi-static capacity and 70% the capacity of the dynamic cache.
10-10-80	The capacity of the static cache is 10% of the cache size (1,2,4,8,16) and 10% for the semi-static capacity and 80% the capacity of the dynamic cache.

Table 6.1: table of Terms

SIZE	1%	2%	4%	8%	16%
50S-50D	0.483	0.500	0.520	0.544	0.576
40S-60D	0.479	0.495	0.515	0.537	0.566
30S-70D	0.474	0.490	0.509	0.530	0.556
20S-80D	0.468	0.483	0.501	0.522	0.545
10S-90D	0.460	0.475	0.492	0.511	0.533
100S-0D	0.448	0.463	0.481	0.500	0.520

Table 6.2: Static-Dynamic Caching (SDC) Results for Different Capacities

From the tables above, we found that the highest hit rate came in the (90S-10D). However, the SDC was not working with a static part larger than the dynamic. By decreasing the static part, the hit rate decreased and most of the SDC caches used a static part smaller than the dynamic. Most of result caching was served with 40% for the static and 60% for the dynamic, with a 30S-70D and for anything lower, the hit rate becomes lower.

6.2 SSDC Results

After all the results from the SDC, we reach our improved SDC, or what we called (SSDC). To provide further details of the results from our cache, we present the results in the tables below with the hits for each part of the cache. For the semi-dynamic part, we separated the hits into two parts: the hits for the day part and the hits for the night period.

We started our test by comparing the SDC with 10% of the cache as static and the remaining as dynamic. Then we compare the result with our cache by giving 10% as

a static and 40% as semi-static and the remaining as dynamic. We got a better amount of hits (Table 6.3) . Also we got a better hit rate that increase whenever we increase the cache capacity (Fig 6.1).

10-40-50	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	379794	133334	141398	669005	1323531	0.481
2%	461026	163094	177102	571515	1372737	0.499
4%	556444	200288	221796	455178	1433706	0.521
8%	661382	255575	281954	314786	1513697	0.550
16%	776824	330684	363465	166015	1636988	0.595

Table 6.3: SSDC Result (10% Static-40% Semi-Static-50% Dynamic)

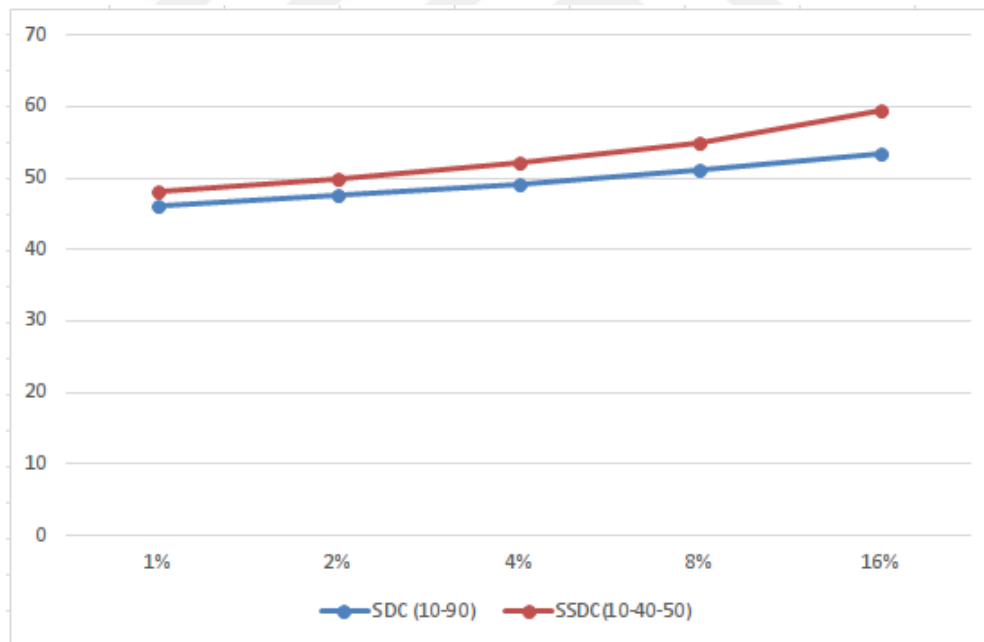


Fig 6.1: The Hit Rate for SDC(10% as static, 90% as dynamic) and SSDC(10% as static, 40% as semi-static and 50% as dynamic)

Then we increased the static part and decrease the semi-static in our cache and we compare it with SDC (20% static and 80% dynamic) see Table(6.4) and fig(6.2). We got an increasing from the beginning by 2% and with 8% capacity its increase again with 3%. with 16% we reached an high increment with 5%, which give us a good improvement comparing with any other enhancement with any other cache.

20-30-50	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	461026	92802	100699	668963	1323490	0.481
2%	556444	114819	129792	571638	1372693	0.499
4%	661382	148836	168309	455188	1433715	0.521
8%	776824	198077	219831	323618	1518350	0.552
16%	905187	265018	295227	172049	1637481	0.595

Table 6.4: SSDC Result (20% Static-30% Semi-Static-50% Dynamic)

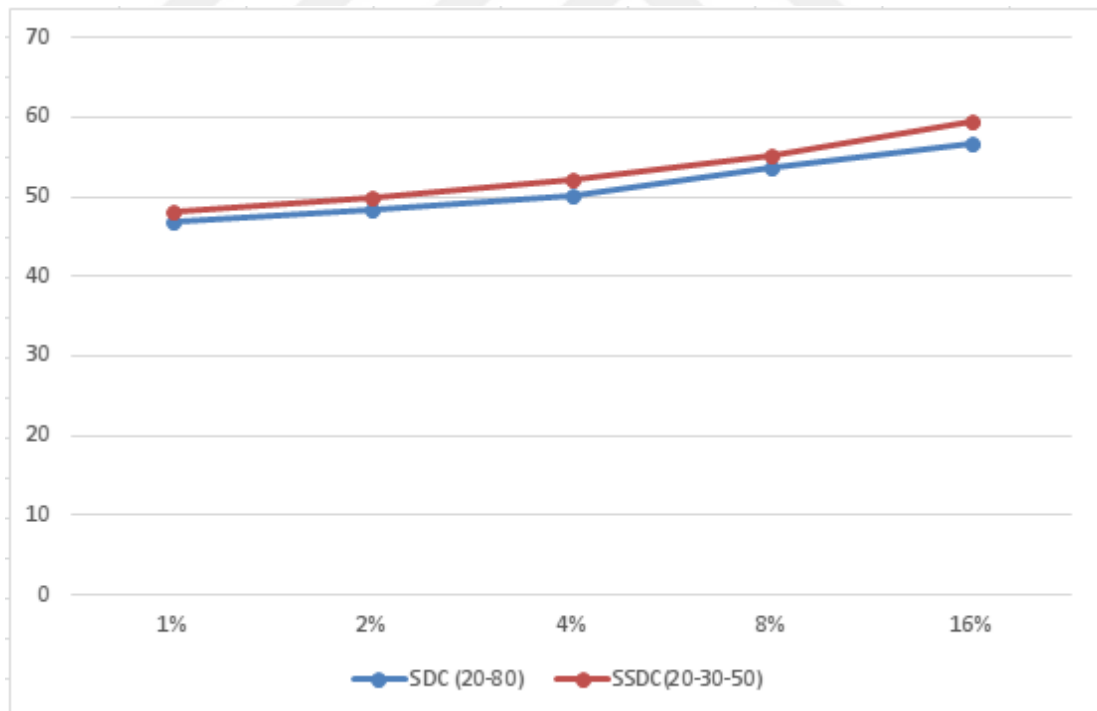


Fig 6.2: The Hit Rate for SDC(20% as static, 80% as dynamic) and SSDC(20% as static, 30% as semi-static and 50% as dynamic)

Again, we increased the static and decreased the semi-static and we kept the dynamic as it. Table (6.5) and the amount of enhanced hit rate can easily recognized with fig(6.3). Here we got an increment hit rate by 1% with 1% and 2% capacities. At the 4% and 8% the increment increased by 1% to be at total 2%. with 16% capacity we reach a high hit rate with 4%.

30-20-50	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	514789	65664	73384	671912	1325749	0.482
2%	616913	84084	96404	580287	1377688	0.501
4%	728708	111795	127215	472966	1440684	0.524
8%	848318	151299	170756	351320	1521693	0.553
16%	972697	207596	228597	223832	1632722	0.594

Table 6.5: SSDC Result (30% Static-20% Semi-Static-50% Dynamic)

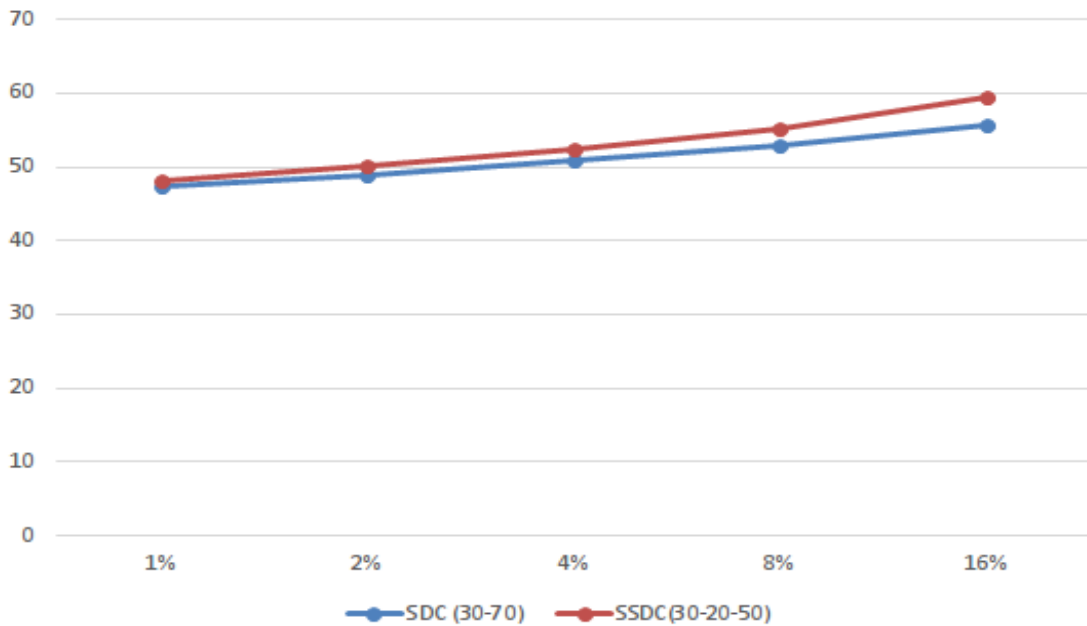


Fig 6.3: The Hit Rate for SDC(30% as static, 70% as dynamic) and SSDC(30% as static, 20% as semi-static and 50% as dynamic)

By keeping the dynamic size in 50% we gave most of the remaining size to the static and with 40% and 10% for semi-static. Table (6.6) and fig (6.4). we start with 1% increment hit rate and its continue from 1% until 8% the increment reached 2% and with 16%, we got 3% more hit rate than the SDC with 16% capacity.

40-10-50	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	556444	39945	45960	689558	1331907	0.484
2%	661613	53513	62562	605691	1383379	0.503
4%	776824	73212	84168	510538	1444742	0.525
8%	905187	98734	114882	399516	1518319	0.552
16%	1033824	139911	156025	291696	1621456	0.590

Table 6.6: SSDC Result (40% Static-10% Semi-Static-50% Dynamic)

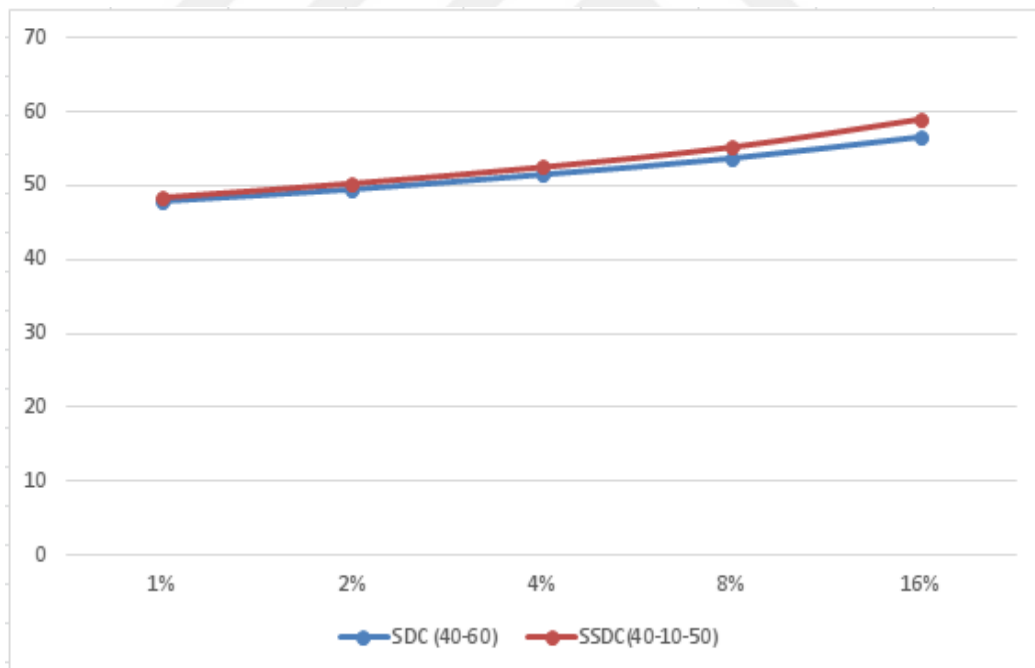


Fig 6.4: The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(40% as static, 10% as semi-static and 50% as dynamic)

After that we give the dynamic cache partial increasing with the size and start with the same previous size for each of the static and semi-static caches. Table (6.7) and fig (6.5). with this test we got a small amount of increment with 0.2% at 1% capacity. With 2% capacity the hit rate almost the same as the SDC, but with 4% we got exactly the same hit rate with 51.1. with 8% we got also a partial amount of hit rate. With 16% the increment get more than 1% enhancement by 0.5. with this test we did not get a high amount of enhancement but still better than the SDC. We make the test with many cases and we considered even the worst cases to cover all the cases with both of SDC and SSDC.

10-30-60	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	379794	113212	119744	699611	1312361	0.477
2%	461026	138962	150810	608091	1358889	0.494
4%	556444	171014	190249	498365	1416072	0.515
8%	661382	216937	243011	369632	1490962	0.542
16%	776824	288096	314383	215770	1595073	0.580

Table 6.7: SSDC Result (10% Static-30% Semi-Static-60% Dynamic)

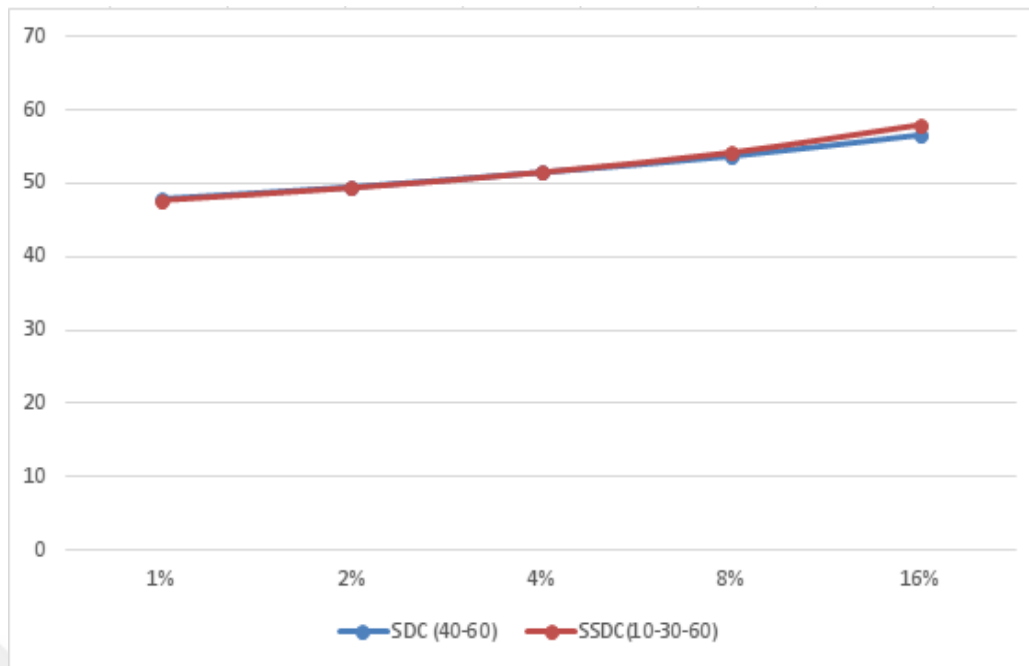


Fig 6.5: The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(10% as static, 30% as semi-static and 60% as dynamic)

In this test we give the static and semi-static the same amount of sizes, we get almost the same amount of hits when we compared it with SDC (40-60) Table(6.8). and we got a small enhancement better than the SDC fig (6.6). we compare our cache with SDC (40% static and 60% dynamic). We start the test with very small amount of decrement which decreased slowly with each new higher capacity. With 8% capacity we reach 1% increment and 2% increment with 16% capacity.

20-20-60	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	461026	72695	79063	699555	1312339	0.477
2%	556444	90786	103518	608924	1359672	0.494
4%	661382	118637	135765	503768	1419552	0.516
8%	776824	159220	177873	381050	1494967	0.544
16%	905187	210811	236841	243904	1596743	0.581

Table 6.8: SSDC Result (20% Static-20% Semi-Static-60% Dynamic)

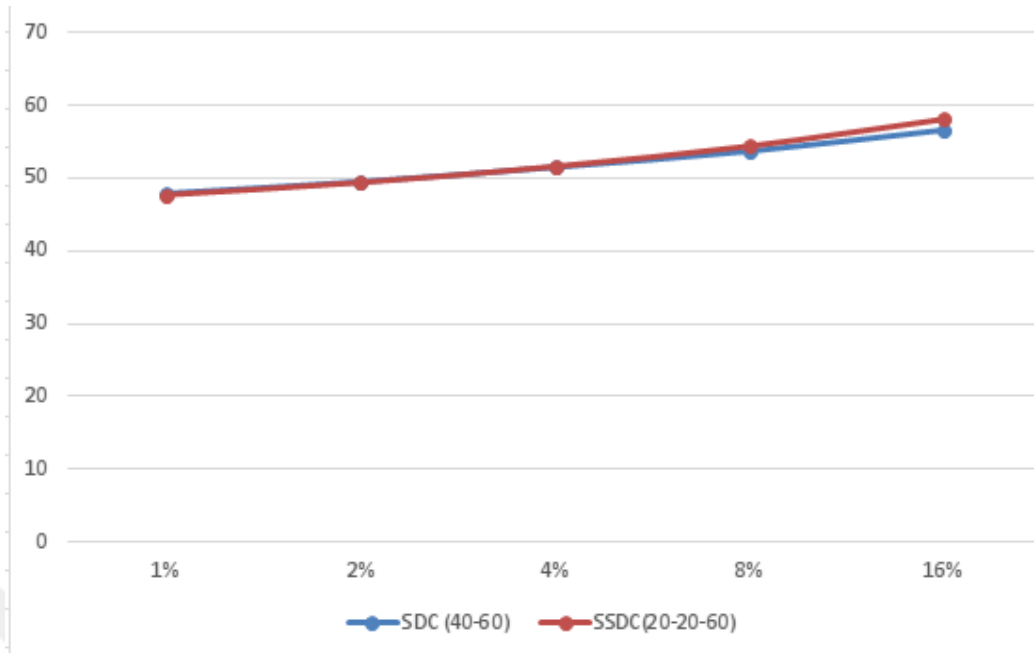


Fig 6.6: The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(20% as static, 20% as semi-static and 60% as dynamic)

also a small enhancement with SSDC (30% as Static, 10% as Semi-Static and 60% as Dynamic) and SDC (40% as static-60% as dynamic) Table (6.9) and Fig (6.7). with this test we start with the same amount of hit rate for both of the SDC and SSDC. Even with high capacities like 8% and 16% we could not get higher than 1% of increment but Technically still better than the SDC.

30-10-60	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	514789	43319	48308	712085	1318501	0.479
2%	616913	56326	64964	628875	1367078	0.497
4%	728708	75378	86986	535523	1426595	0.519
8%	848318	103264	118212	428892	1498686	0.545
16%	972697	141831	160199	314977	1589704	0.578

Table 6.9: SSDC Result (30% Static-10% Semi-Static-60% Dynamic)

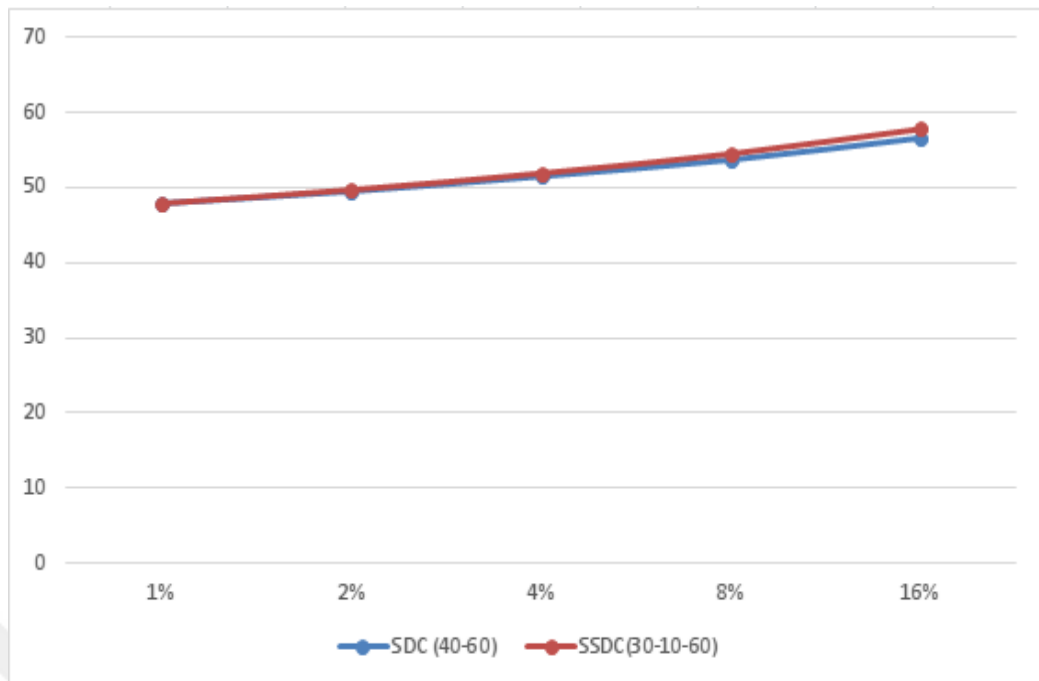


Fig 6.7: The Hit Rate for SDC(40% as static, 60% as dynamic) and SSDC(30% as static, 10% as semi-static and 60% as dynamic)

This time we increased the dynamic part with 70% size and 10% as static with 20% as a semi-static, the SDC (30% static, 70% dynamic) starts better than ours until we get better result with the 8% as a capacity. Table (6.10) and Fig (6.8). the conclusion of this test, that we got the same scenario as the previous test .

10-20-70	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	379794	88679	93553	737608	1299634	0.472
2%	461026	110019	119333	653694	1344072	0.489
4%	556444	135553	152439	552471	1396907	0.508
8%	661382	174751	197096	431288	1464517	0.532
16%	776824	229804	255968	290083	1552679	0.565

Table 6.10: SSDC Result (10% Static-20% Semi-Static-70% Dynamic)

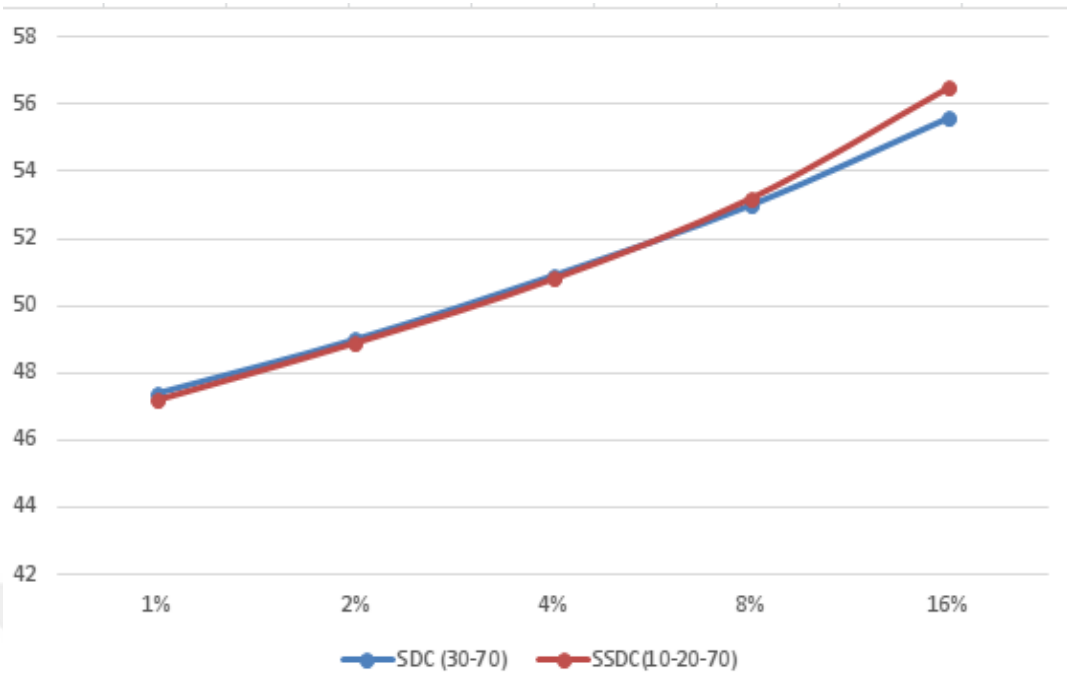


Fig 6.8: The Hit Rate for SDC(30% as static, 70% as dynamic) and SSDC(10% as static, 20% as semi-static and 70% as dynamic)

we get better result here than the previous test by increasing the static and decreasing the semi-static and Table (6.11) shows the amount of hits we got. Fig (6.9) shows the hit rate. Also, we got the same scenario from previous test. But the problem here that the SDC start better than our test with 0.1% and with 8% and 16% we only get a number near the 1% as a hit rate which make it one of worst cases to work with in SSDC, but also we got better than SDC.

20-10-70	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	461026	47923	52138	741173	1302260	0.473
2%	556444	60701	69593	661824	1348562	0.490
4%	661382	80043	92600	569984	1404009	0.510
8%	776824	108069	122852	463649	1471394	0.535
16%	905187	144456	166339	338380	1554362	0.565

Table 6.11: SSDC Result (20% Static-10% Semi-Static-70% Dynamic)

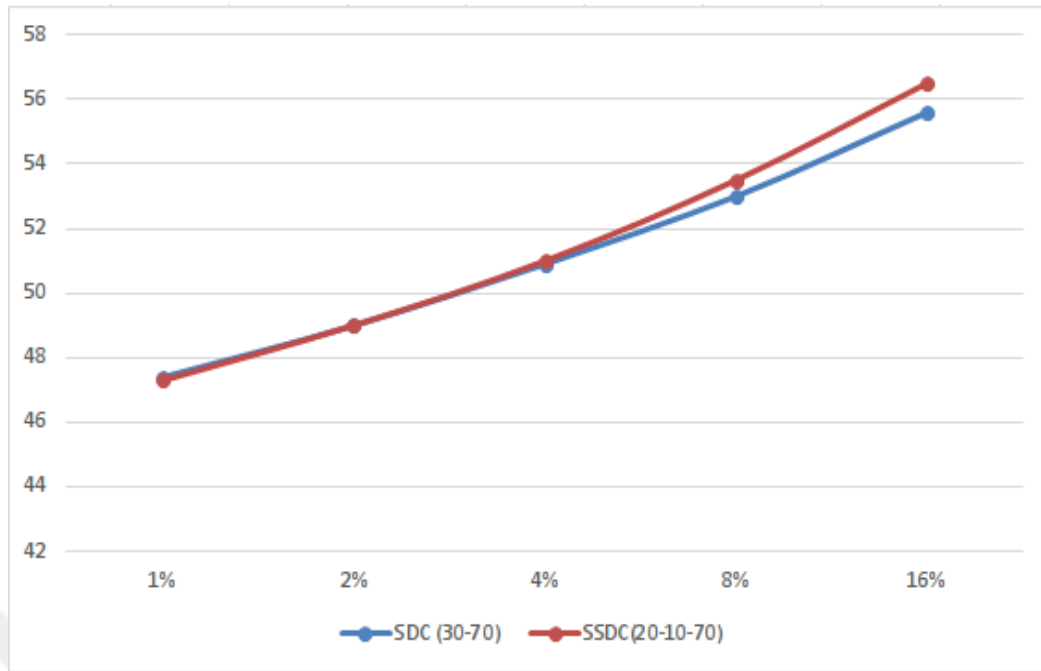


Fig 6.9: The Hit Rate for SDC(30% as static, 70% as dynamic) and SSDC(20% as static, 10% as semi-static and 70% as dynamic)

Here we give the SSDC the Highest amount of Dynamic than any previous test with 80%. we gave both static and semi-static the same amount of size. Table (6.12) and Fig (6.10) shows the result and hit rate. Here also the test was not that much better then the previous cases until the 3% increment with the 16% capacity. At the End we reached a result that whenever we increase the dynamic with high amount of sizes the hit rate drop down.

10-10-80	Static Hits	Day Hits	Night Hits	Dynamic Hit	Total Hits	Hit Rate
1%	379794	56866	60294	786846	1283800	0.467
2%	461026	72695	79063	712835	1325619	0.482
4%	556444	90786	103518	625150	1375898	0.500
8%	661382	118637	135765	521714	1437498	0.523
16%	776824	159220	177873	399229	1513146	0.550

Table 6.12: SSDC Result (10% Static-10% Semi-Static-80% Dynamic)

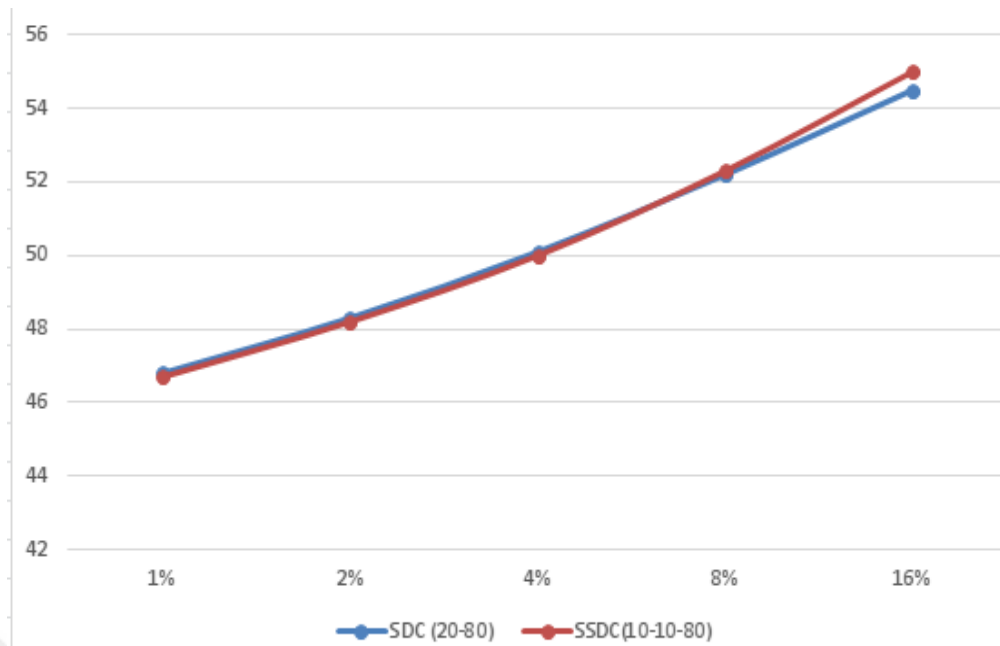


Fig 6.10: The Hit Rate for SDC(20% as static, 80% as dynamic) and SSDC(10% as static, 10% as semi-static and 80% as dynamic)

From all the tables above, we can observe that we obtain good hit ratios which are better than the SDC. Moreover, we obtain the best hit ratio, and to make a clear scene for the hit ratio, we present a baseline for our cache hit rate with the best case in SDC and the worst case in SDC in (Fig. 6.1).

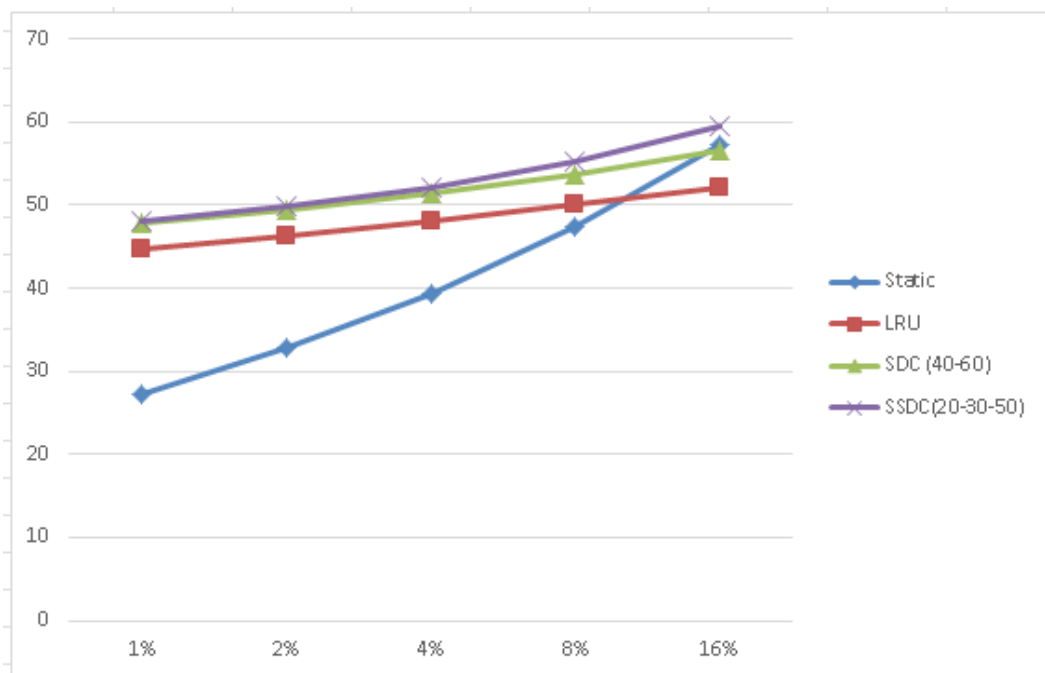


Fig 6.11: The Hit Rate For Each Cache With Optimal Values For Each Cache

CHAPTER 7

Conclusion

In this work, we presented SSDC, an improved version of the state-of-art SDC in the result caching, by enhancing the caching decisions and by exploiting the search engine user's query behaviour. We depend on new features within our research, including query compatibility and modified frequency depending on the old features from the ordinary query logs. We make a new cache system that aims to increase the hit rate for the result caching in addition to using a query log called the AOL data set that made a good choice for our work after we enhanced it by removing any unnecessary and ambiguous information.

Different models of caching were used in addition to our using the static cache by selecting the queries with the highest number of occurrences from a previous logs. In the dynamic case we served the cache depending on the recency of the queries that submitted to it. We made a simulation of the SDC cache that contained two segments, namely static and dynamic caches.

We improved the SDC and conducted many experiments with different sizes and capacities to select the optimal size among them. We aimed to enhance the hit rate in any possible way and with our new method, the compatibility of the query was to reduce the financial cost of the caching in WSE.

Our work provides a new strategy with a better hit rate than the SDC and we obtain, in the best case, an increment of 3%, which is considered to be a good hit rate in comparison to increments from the previous works from different caching models.

REFERENCES

1. **Alici, S., Altingovde, I. S., Ozcan, R., Cambazoglu, B. B., Ulusoy, O., (2011)**, “*Timestamp-based result cache invalidation on web search engines*”, ACM SIGIR 11. ACM, New York. NY, USA, USA, pp. 973-982.
2. **Alici, S., Altingovde, I. S., Ozcan, R., Cambazoglu, B. B., Ulusoy, O., (2012)**, “*Adaptive time-to-live strategies for query result caching in web search engines*”, 34th European Conference on Advances in Information Retrieval. ECIR’12. Springer-Verlag, Berlin, Heidelberg, pp. 401-412.
3. **Altingovde, I. S., Ozcan, R., Cambazoglu, B. B., Ulusoy, O., (2011)**, “*Second chance: a hybrid approach for dynamic result caching in search engines*”, 33rd European Conference on Advances in Information Retrieval. ECIR’11. Springer-Verlag, Berlin, Heidelberg, pp. 510-516.
4. **Baeza-Yates, R., Saint-Jean, F., (2003)**, “*A three level search engine index based in query log distribution*”, String Processing and Information Retrieval. Vol. 2857 of Lecture Notes in Computer Science. Springer Berlin, Heidelberg, pp. 56-65.
5. **Blanco, R., Bortnikov, E., Junqueira, F., Lempel, R., Telloli, L., Zaragova, H., (2010)**, “*Caching search engine results over incremental indices*”, 33rd International ACM SIGIR 11. ACM, New York. NY, USA, USA, pp. 82-89.
6. **Bortnikov, E., Lempel, R., Vornovitsky, K., (2011)**, “*Caching for realtime search*”, 33rd European Conference on Advances in Information Retrieval. ECIR’11. Springer-Verlag, Berlin, Heidelberg, pp. 104-116.
7. **Cambazoglu, B. B., Varol, E., Kayaaslan, E., Aykanat, C., Baeza-Yates, R., (2010)**, “*Query Forwarding in Geographically Distributed Search Engines*”, ACM SIGIR 10. ACM, Geneva., Switzerland, pp. 90-97
8. **Cambazoglu, B. B., Altingovde, I. S., Ozcan, R., Ulusoy, O., (2012)**, “*Cache-based query processing for search engines*”, ACM Transactions on the Web 6 (4), 1-24.

9. **Cambazoglu, B. B., Junqueira, F. P., Plachouras, V., Banachowski, S., Cui, B., Lim, S., Bridge, B., (2010)**, “*A refreshing perspective of search engine caching*”, 19th International Conference on World Wide Web. pp. 181-190.
10. **Fagni, T., Perego, R., Silvestri, F., Orlando, S., (2006)**, “*Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data*”, ACM Transactions on Information Systems 24 (1), pp 51-78.
11. **Frances, G., Bai, X., Cambazoglu, B. B., Baeza-Yates, R., (2014)**, “*Improving the Efficiency of Multi-site Web Search Engines*”, 7th ACM International Conference on Web Search and Data Mining. WSDM’14. ACM, New York, USA, pp. 3-12.
12. **Hoelscher, C., (1998)**, “*How internet experts search for information on the web*”, International Conference on World Wide Web, Orlando, Florida, USA.
13. **Jacobs, T., Longo, S., (2015)**, “*A Study of Caching Strategies for Web Service Discovery*”, IEEE International Conference on Web Services, ICWS., New York, USA, pp. 464-471.
14. **Junquera, F. P., Leroy, V., Morel, M., (2012)**, “*Reactive Index Replication for Distributed Search Engines*”, ACM SIGIR, SIGIR’12, August 12-16. 2012, Portland, Oregon, USA. pp. 831-840.
15. **Jonassen, S., Cambazoglu, B. B., Silvestri, F., (2012)**, “*Prefetching query results and its impact on search engines*”, 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 631-640.
16. **Kayaaslan, E., Cambazoglu, B. B., Aykanat, C., (2013)** “*Document replication strategies for geographically distributed web search engines*”, Information Processing and Management 49, pp. 51-66.
17. **Lempel, R., Moran, S., (2003)**, “*Predictive Caching and prefetching of query results in search engines*”, 12th International Conference on World Wide Web, ACM Press, pp.19-28.
18. **Li, H., Lee, W. C., Sivasubramaniam, A., Giles, C. L., (2007)**, “*A Hybrid cache and prefetch mechanism for scientific literature search engines*”, 7th International Conference on Web Engineering, ICWE’07, Springer-Verlag, Berlin, Heidelberg, pp. 121-136.
19. **Long, X., Suel, T., (2005)**, “*Three-level caching for efficient query processing in large web search engines*”, 14th International Conference on World Wide Web., ACM, New York, NY, USA, pp. 257-266.

20. **Markatos, E. P., (2000)**, “*On Caching search engine results*”, 5th International Web Caching and Content Delivery Workshop.
21. **Markatos, E. P., (2001)**, “*On Caching search engine query results*”, Computer Communications 24 (2), pp. 137-143.
22. **Marin, M., Gil-Costa, V., Gomez-Pantoja, C., (2010)**, “*New Caching Techniques For web search engines*”, 19th ACM International Symposium on High Performance Distributed Computing, HPDC’10, ACM, New York, NY, USA, pp. 215-226.
23. **O’Neil, E. J., O’Neil, P. E., Weikum, G., (1993)**, “*The lru-k page replacement algorithm for database disk buffer*”, ACM SIGMOD International Conference On Management Of Data, pp. 297-306.
24. **Ozcan, R., Altingovde, I. S., Cambazoglu, B. B., Junqueira, F. P., Ulusoy, O., (2012)**, “*A five-level static architecture for web search engines*”, information processing and Management 48 (5), 828-840.
25. **Ozcan, R., Altingovde, I. S., Cambazoglu, B. B., Ulusoy, O., (2013)**, “*Second Chance: A hybrid approach for dynamic result caching and prefetching in web search engines*”, ACM Transactions of the Web 8 (1), 3:1-3:22.
26. **Prokhorenkova, L. O., Ustinovskiy, Y., Samosvat, E., Lefortier, D., Serdyukov, P. M., (2014)**, “*Adaptive Caching of Fresh Web Search Results*”, Advances In Information Retrieval, 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 – April 2, 2015. pp. 110-122.
27. **Puppini, D., Silvestri, F., Perego, R., Baeza-Yates, R., (2010)**, “*Tuning the capacity of search engines: load-driven routing and incremental caching to reduce and balance the load*”, ACM Transactions on Information Systems 28 (2), 5:1-5:36.
28. **Silverstein, C., Henzinger, M., Marais, H., Moricz, M., (1999)**, “*Analysis of a very large web search engine query log*”, ACM SIGIR Forum, pp. 6-12.
29. **Sazoglu, F. B., Ulusoy, O., Altingovde, I. S., Ozcan, R., Cambazoglu, B. B., (2015)**, “*Propagating Expiration Decisions in a search engine Result Cache*”, 24th International Conference on World Wide Web, WWW’15, ACM, New York, NY, USA. pp. 107-108.

30. Saravia, P. C., Silva de Moura, E., Ziviani, N., Meira, W., Fonseca, R., Riberio-Neto, B., (2001), “*Rank-preserving two-level caching for scalable search engines*”, 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '01. ACM, New York, NY, USA, pp. 51-58.

31. Skobeltsyn, G., Junqueira, F., Plachouras, V., Baeza-Yates, R., (2008), “*a Combination of results caching and index pruning for high-performance web search engines*”, 31th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 131-138.

32. Tolosa, G., Becchetti, L., Feuerstein, E., Marchetti-Spaccalema, A., (2014), “*Performance Improvements for Search Systems using an Integrated Cache of Lists+Intersections*” In: String Oricessing and Information Retrieval Lecture Notes in Computer Science, Edition: Volume 8799, Springer International Publishing, pp. 227-235.

