DEDICATION

To my parents Zainab and Alifor their infallible support.

To my dear brothers Gamal and Fath Elrahman.

To my friends for not giving up their friendship.

ACKNOWLEDGMENT

I am really thankful to my advisors Dr.Yahia Abdalla Mohammed and Dr. Amir Abdelfattah Ahmed Eisa, for their guidance, patience, and encouragement during this research. Who trained me to do focused, systematic and scientific research. Their dedication towards research is inspiring. I am very thankful to Dr. Mohammed Elhaiz, Dr. Eman Abuelmaale, on my thesis committee and for their advice and suggestions regarding the thesis and beyond.

I would like to thank my family and friends for their encouragement and for believing in me and my potential.

Moreover, I would like to thank my colleagues at The Sudan University of Science and Technology and International University of Africa for help, which makes my study at Rutgers enjoyable and fruitful. I am also thankful to Computer Center–Sudan University for their assistance and support.

PUPLICATION BASED ON THIS THESIS

1. Hiba. A. Nasir, Yahia. A. Mohammed, Amir. A. Eisa, "Agent-based Proxy Cache Cleanup Model using Fuzzy Logic", proceedings of International Conference of Computing Electrical and Electronic Engineering (ICCEEE), Khartoum, Sudan, August 2013.
2. Hiba. A. Nasir, Yahia. A. Mohammed, Amir. A. Eisa, "A Survey of Intelligent Web Caching", proceedings of Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE 2015), Khartoum, Sudan, September 2015.
3. Hiba. A. Nasir, Yahia. A. Mohammed, Amir. A. Eisa, "Proxy Cache Cleanup Improvement using an Agent- Based Model", Elsevier 2015 (on review)

ABSTRACT

Web proxy caching is one of the effective solutions to avoid web service bottleneck, reduce traffic over the Internet and improve scalability of the web service. The core of a caching system is the caching replacement policies. This study describes the use of intelligent agent model to improve the performance of the proxy cache. A multi-agent system has been developed to control the cache cleanup task on the hierarchy caches.

Fuzzy logic is used to combine Least Frequently Used (LFU), Least Recently Used (LRU) and Size caching replacement policies on the parent cache side. LFU and LRU policies are used on the child caches side. Cache cleaner Agents use fuzzy logic to make an intelligent decision and remove the web object proactively when it has high clean up priority. Reactive Coordination has been applied between the parent and child cleaner agents to achieve the cleanup task in efficient way, they have a common goal to increase hit ratio and byte hit ratio.

Coordination agent applied the coordination rules when the web object with medium priority is found in parent and children caches. Q-learning algorithm has been implemented by the cleaner agent to avoid difficult calculation when it reached a similar state and take a suitable action.

A reward value has been associated to each action, when Cleaner agent takes its optimal action that leads to the goal, it has an instant high reward. Other actions have low reward. States and actions had been represented on a graph each node represented a "state", agent's movement from one node to another represented the "action".

The model has been tested using five samples of workload generated using Webtraff simulator, these samples represented the users requests and used cache sizes. The standard performance metrics Hit Ratio and byte hit ratio are used to evaluate the cache performance.

Simulation results show that when the cache size increase the new approach PCCIA performs better than LRU,LFU and Size replacement polices in terms of hit rate and byte hit rate.

المستخلص

تعد ذاكرة الوكيل المخبأه واحدة من الحلول الفعالة لتجنب االاختناق فى خدمة الانترنيت و تقليل الإزدحام وتحسين كفاءة خدمة الإنترنيت. وتعتبر سياسات الاحلال هو أساس نظام الذاكرة. وصفت هذه الدراسة إستخدام إنموذج العميل الذكى لتحسين اداء الذاكرة المخبأه.تم تطوير نظام العملاء المتعددين للتحكم فى عملية تفريغ الذاكرة المخبأه للمعمارية الهرمية.

أستخدم المنطق الضبابى لدمج سياسات الاحلال Least Frequency Used, Least Recently Used and Size فى ذاكرة الوكيل الاب, وتم إستخدام سياسات Least Frequency Used and Least Recently Used فى جانب الذواكر الأبناء. بمبادره من العميل المفرغ للذاكرة تم إستخدام المنطق الضبابى لإتخاذ قرار ذكى لمسح صفحة الانترنيت التى لها اولوية تفريغ عالية. تم التنسيق بين العملاء المفرغي لذاكرة الأب والأبناء وذلك لانجاز مهمة التفريغ بطريقة كفؤة. وكل العملاء لهم هدف مشترك عام وهو زيادة نسبة عدد الطلبات ونسبة حجم الطلبات بالبايت.

قام الوكيل المنسق بتطبيق قواعد للتنسيق عندما تكون صفحة الإنترنيت الموجودة فى ذاكرة الأب والأبناء لها اولوية تفريغ متوسطة. تم تطبيق خوارزمية التعلم Q بواسطة العميل المفرغ لتجنب الحسابات عندما يصل العميل الى حالة مشابهه وذلك لإتخاذ الفعل المناسب.

تم إسناد قيمة لكل فعل. يصل العميل المفرغ الى أفضل فعل الذى يقود الى الحالة الهدفية التى لها قيمة عالية.وبقية الأفعال لها قيم منخفضة.تم تمثيل الحالات والافعال فى شكل عقد تمثل الحالات ,تم تمثيل الفعل بحركة العميل من عقدة لأخرى.

أستخدم خمس عينات مولدة بإستخدام المحاكى Webtraff لإختبار الإنموذج ,هذه العينات تمثل طلبات المستخدمين التى استخدمت فى حجم الذاكرة. أستخدمت مصفوف القياسات القياسية نسبة عدد الطلبات ونسبة حجم الطلبات بالبايت لتقييم أداء الذاكرة.

أوضحت نتائج المحاكى ان الطريقة الجديدة تنجز أفضل على سياسات الاحلال Least Frequency Used, Least Recently Used and Size فى نسبة عدد الطلبات ونسبة حجم الطلبات بالبايت.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATION

AI	Artificial Intelligence
AV	Agent/Role view
BHR	Byte Hit Ratio
BN	Bayesian network
C1 del	delete web object from child 1 cache
C1 freq	web object's frequency in child 1 cache
C1 save	save web object in child 1t cache
C1 time	web object's request time in child 1 cache
C2 del	delete web object from child 2 cache
C2 freq	web object's frequency in child 2 cache
C2 save	save web object in child 1 cache
C2 time	web object's request time in child 2 cache
CART	Classification and Regression Trees
CG	Coordination Graph
CoordRL	Coordinated Reinforcement-Learning
CP	Cleanup Priority
DV	Domain view
FL	Fuzzy Logic
GB	Gigabyte
GDS	Greedy-Dual-Size

GTV	Goal/Task view
HR	Hit Ratio
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IV	Interaction view
LFU	Least Frequently Used Algorithm
LRU	Least Recently Used Algorithm
MAL	Multi Agent Learning
MARS	Multivariate Adaptive Regression Splines
MAS	Multi-Agent Systems
MB	Megabyte
MF	Member Function
MMDP	Multi=agent Markov Decision Process
NN	Neural Networks
OV	Organization view
P del	delete web object from parent cache
P freq	web object's frequency in parent cache
P save	save web object in parent cache
P time	web object's request time in parent cache
PCCIA	Proxy Cache Cleanup Improvement using an Agent-based Model
RF	Random Forest
RL	Reinforcement Learning
RS	Rough Set
SIZE	SIZE Algorithm
ST	Spanning Tree

TN	TreeNet
UML	Unified Modeling Language
URL	Uniform Resource Locator
VE	Variable Elimination Algorithm
WWW	World Wide Web

CHAPTER 1: INTRODUCTION

1.1 Motivation

The Web has become the most important source of information for the world; it contributes greatly to our life in many fields such as education, entertainment, Internet banking, remote shopping and software downloading. This has led to rapid growth in the number of Internet users, which resulting in an explosive increase in traffic or bottleneck over the Internet[1].

Web proxy caching is one of the effective solutions for improving Web performance to avoid Web service bottleneck, reduce traffic over the Internet and improve scalability of the Web system.

In Web caching, the web objects that to be visited in the future are stored close to the user to reduce network bandwidth, user (client) perceived lag and loads on the origin servers so as improving scalability of Web system[2].

Cache replacement is the core of web caching; hence, the design of efficient cache replacement algorithms is crucial for the success of caching mechanisms.

1.2 Web Proxy Caching Policies

There are many replacement policies to consider when designing a proxy server. The most commonly known are still based on traditional caching policies. These conventional policies are suitable in traditional caching like CPU caches and virtual memory systems, but they are not efficient in Web caching. This is because they only consider one factor in caching decisions

and ignore the other factors that have impact on the efficiency of the Web proxy caching [3]. The simplest and most common cache management approaches are:

- Least-Recently-Used (LRU) algorithm which removes the least recently accessed objects until there is sufficient space for the new objects. LRU is easy to implement and proficient for uniform size objects such as the memory cache. However, it does not perform well in Web caching since it does not consider the size or the download latency of objects.
- Least-Frequently-Used (LFU) is another common Web caching that replaces the object with the least number of accesses. LFU keeps more popular Web objects and evicts rarely used ones. LFU suffers from the cache pollution in objects with the large reference accounts, which are never replaced even if they are not re-accessed again.
- SIZE policy is one of the common Web caching policies that replace the largest object(s) from cache when space is needed for a new object.
- Greedy-Dual-Size (GDS) policy it is extension of the SIZE policy. The algorithm integrates several factors and assigns a key value or priority for each Web object stored in the cache. When cache space becomes occupied and new object is required to be stored in cache, the object with the lowest key value is removed. When user requests an object g, GDS algorithm assigns key value K(g) of object g as shown in Equation 1.1:

$$K(g) = L + C(g) / S(g) \quad (1.1)$$

Where C(g) is the cost of fetching object g from the server into the cache; S(g) is the size of object g; and L is an aging factor. L starts at 0 and is updated to the key value of the last replaced object. The key value K(g) of object g is updated using the new L value since the object g is accessed again. Thus, larger key values are assigned to objects that have been visited recently. If the cost is set to 1, it becomes GDS(1), and when the cost is set to P=2+size/536, it becomes GDS(P). Cao and Irani [3] proved that the GDS algorithm achieved better performance

compared with some traditional caching algorithms. However, the GDS algorithm ignores the frequency of the access to web object.

- Greedy-Dual-Size-Frequency (GDSF)[3]enhanced GDS algorithm by integrating the frequency factor into the key value K(g) as shown in the following Equation:

$$K(g) = L + F(g) * C(g) / S(g) \qquad (1.2)$$

where F(g) is frequency of the visits of g. Initially, when g is requested by user, F(g) is initialized to 1. If g is in the cache, its frequency is increased by one.

In fact, few important features of Web objects that can influence the Web proxy caching are: recency, frequency, size, cost of fetching the object, and access latency of object. Depending on these factors, the Web proxy policies are classified into five categories: Recency-based polices, Frequency based polices, Size-based polices, Function-based polices and Randomized polices.

Many Web cache replacement policies have been proposed for improving performance of Web caching. However, it is difficult to have a policy that performs well in all environments or for all time because the combination of the factors that can influence the Web proxy cache is not an easy task[3].

Due to cache space limitations, an intelligent mechanism is required to manage the web cache contents efficiently. The core of a caching system is the cache replacement policy.

Most of the web proxy policies use one factor for making decisions about caching. However, a combination of these factors to get wise replacement decision is not a simple task, because one factor in a particular environment

may be more important in other environments. Hence, there is a need for an effective and adaptive approach, which can effectively incorporate the significant factors into web caching decisions.

1.3 Problem Statement

Since the space apportioned to the cache is limited and the most of the existing traditional proxy caching policies such as Size, Greedy Dual Size (GDS), Least Frequently Used (LFU), and Least Recently Used (LRU), are not efficient in Proxy caching, since each policy considers one factor and ignore the others that have effect on the efficiency of the proxy caching and may suffer from a cache pollution problem that means that a cache contains objects that are not frequently visited. This causes a reduction of the effective cache size and negatively affects the performance of web proxy caching.

Combination of the factors that can influence the cleanup task to get wise replacement decision is not an easy task because one factor in a particular situation or environment is more important than other environments [9]. Hence, the difficulty in determining which ideal web objects will be re-accessed is still a big challenge faced by the existing Web caching techniques. What Web objects should be cached and what Web objects should be replaced to make the best use of available cache space, improve hit rates, reduce network traffic, and alleviate loads on the original server.

Thus, there is a need to find another approach to perform and control the cache cleanup task in an efficient way.Recent studies have shown that the

intelligent web caching approaches are more efficient and adaptive to web caching environments compared to other approaches.

Proxy cache clean up task is performed on hierarchy caches individually; there are no coordination between parent and children agents when they performed the cache cleaning task to decide which objects should be stored in the caching system and which object should be removed.

The study tries to solve the cache cleanup problem focus on three key problems:

- Proxy cache cleanup process is performed with human administrator, the study proposed a multi-agent systems to automate this task.
- Most of the existing traditional caching policies are not efficient in Proxy caching, they consider just one factor and ignore other factors that influence the efficiency the web caching, the study propose the integration of the factors to improve cache efficiency.
- Current cache cleanup task is performed in each child and parent proxy caches individually, the study proposed a proactive coordination rulesto coordinate this task on parent and children caches, so as to improve the web performance in efficient way.

1.4 Research Aim

This research aims to enhance the performance of web proxy caching through intelligent web proxy caching approaches based on intelligent Agent model.

1.5 Research Objectives

In order to achieve the aim of the study, the objectives of this research are stated as follows:

1- To develop an intelligent approach based on agent model to automate proxy cache cleanup task in an efficient way,

2-To integrate caching polices on parent and child caches,

3- To develop a coordination mechanism between the parent and child agent to achieve cache cleaning task in efficient way,

4- To increase Hit Rate and Byte Hit Ratio on the hierarchy proxy caches,

5- And to improve the scalability of hierarchy cache system.

1.6 Research Scope

This work proposes an agent-based model to automate and control the cache cleanup task. The cache cleanup task is performed proactively in the hierarchical caching system based on the integration of the caching polices so as to improve the proxy cache performance.

1.7 Research Assumptions

In order to achieve the research objectives, the assumptions of the study are stated as follows:

1- Web caching is applied on hierarchy web proxy cache.
2- Hit ratio (HR) and byte hit ratio (BHR) are used to evaluate the performances of intelligent web proxy caching approaches since HR and BHR are two widely used metrics for evaluating the performance of web proxy caching policies.

3- WebTraff simulator [88] is modified and used for evaluating the proposed intelligent web proxy caching approach.
4- Jade Platform [9]has exploited for developing agent based model.

1.8 Research Questions

1. How can the performance of web proxy caching be enhanced using Intelligent Agent?
2. What are the tasks to be achieved by using agents?
3. How to build efficient intelligent agent model to improve the cache cleanup task?
4. How to use the multi-agent systems to distribute controlling functions such as monitoring, cleanup and coordinating tasks among multiple agents.
5. How to add the artificial intelligence techniques such as learning to enhancing the performance of web proxy caching?
6. How agents can coordinate the cleanup task?
7. How to evaluate the new approach compared to other works?

1.9 Research Contribution

The major contribution of this work is to add the idea of using multi-agent systems to automate the hierarchy caches cleanup task proactively and in efficient way. Agents use fuzzy logic to integrate the Size, Least Recently Used (LRU) and Lest Frequency Used (LFU) caching polices and makes an intelligent decision about clean up priority so as to improve the performance of the proxy cache.

More over this work added the coordination mechanism to coordinate object replacement because it is one of the important issues in hierarchical caching systems to decide which objects should be stored and which object should be removed. Also Q learning algorithm has been implemented to avoid difficult calculation when the cleaner agent reached a similar state and take a suitable action.

The experimental results show that the new approach outperforms conventional caching techniques on a large cache size in terms of hit ratio (HR) and byte hit ratio (BHR).

1.10 Thesis Organization

This thesis contains seven chapters and is organized as follows:

Chapter 1 provides a brief introduction of the study. It covers topics on motivations, problem statement, research aim, objectives, question, scope, assumptions and summary of research contributions.

Chapter 2 introduces a general overview of the literature review of this study, discusses the related work on intelligent web caching.

Chapter 3 describes in-depth the Agent Message methodology that used in this study. The methodology is presented as diagrams that describe briefly the different views of model analysis and design to agent's tasks, behaviors, interactions and messages.

Chapter 4 describes in details of cache cleaner agent's coordination and learning.

Chapter 5 describes the implementation of the new model.

Chapter 6 Simulation Results are discussed in this chapter, the new approach has been compared with the most common and relevant web proxy caching policies.

Finally, Chapter 7 concludes the thesis. In addition, it provides suggestions and recommendations for future study.

CHAPTER 2: RELATED WORK

RELATED WORK

2.1 Intelligent Web Caching

Many studies have investigated the enhancement of caching performance using artificial intelligence techniques. Some of these studies are real systems, while others are simulations. An approach to web proxy cache replacement, which utilizes neural networks for replacement decisions, has been proposed in[1]. Machine learning techniques are used to increase the performances of traditional Web proxy caching policies such as SIZE, and Hybrid. Naive Bayes (NB) and decision tree (C4.5) are used and integrated with traditional Web proxy caching techniques to form better caching approaches known as NB–SIZE, and C4.5–Hybrid. The proposed approaches are evaluated by trace-driven simulation and compared with traditional Web proxy caching techniques. Experimental results have revealed that the proposed NB–SIZE and C4.5–Hybrid significantly increased Pure Hit-Ratio, Byte Hit-Ratio and to reduced the latency when compared with SIZE and Hybrid.

Reference [2] Presented substantial RS analysis based on Inductive Learning methods to optimize mobile Web pre-caching performance to probe significant attributes and generate the decision rules. RS granularity in mobile Web pre-caching allows decision rules to be induced. These rules are important in optimizing storage of mobile application by executing caching strategy in specifying the most relevant condition attributes. This approach provides guidance to the administrator in mobile Web pre-caching to select the best parameters to be cached. Based on this analysis, the

administrator may reorganize the parameter of log data set in proxy caching accordingly.

Study [3]gave solution for scalability and robustness of Distributed web caching System and for load balancing and Metadata manageability. The study had refined the technique using proxy server clusters with Dynamic allocation of requests. They devised an algorithm for Distributed Web Cache concepts with clusters of proxy server based on geographical regions. It increases the scalability by maintaining metadata of neighbors collectively and balances load of proxy servers dynamically to other less congested proxy servers, so system doesn't get down unless all proxy servers are fully loaded so higher robustness of system isachieved. This algorithm also guarantees data consistency between the original server object and the proxy cache objects using semaphore.

Reference [4] discussed an alternative way to implement an autonomous SPY tool that is capable to self-direct, either to cache or not; the objects in a document based on the behavior of users' activities (number of object hits, script size of objects, and time to receive objects) in an Internet based electronic services (e-services) for enhancing Web access. In this study, an integration of Particle Swarm Optimization (PSO) and Artificial Neural Networks (ANN) in Web caching technology is found promising in alleviating the congestion of Internet access.

The usage of Rough Set (RS) theory for performance enhancement of Web caching is illustrated in [5]. The RClass System framework is used as a knowledge representation scheme for uncertainty in data for optimizing

the performance of proxy caching that is used to store the knowledge discovery of user behaviors in log format. Substantial RS analysis based on Inductive Learning methods is presented to optimize Web caching performance to probe significant attributes and generate the decision rules, these rules are important in optimizing users' storage by executing caching strategy, in specifying the most relevant condition attributes. The proposed framework is illustrated using trace-based experiments from Boston University Web trace data set.

In [6] principles and some existing web caching and pre-fetching approaches are reviewed. The conventional and intelligent web caching techniques are investigated and discussed. Moreover, Web pre-fetching techniques are summarized and classified with comparative limitations of these approaches. The paper also discusses some studies that integrate both web caching and pre-fetching together.

Reference [7]discusses an alternative way to implement log data detection tool. This tool is capable to self directed either to cache or not to cache the objects in a document based on the log data. In this study, an integration of PSO and ANN in Web caching technology is promising in alleviating the congestion of Internet access.

Paper [8] provides an improved prediction accuracy and state space complexity by using novel approaches that combine clustering, association rules and Markov models. The three techniques are integrated together to maximize their strengths. The integration model has been shownto achieve better prediction accuracy than individual and other integrated models.

Reference [9] proposes splitting client-side web cache to two caches, short-term and long-term. Initially, a web object is stored in short-term cache, and the web objects that are visited more than the pre-specified threshold value will be moved to long-term cache. Other objects are removed by Least Recently Used (LRU) algorithm as short-term cache becomes full. More significantly, when the long-term cache saturates, a neuro-fuzzy system is employed in classifying each object stored in it into either cacheable or un cacheable. The old un cacheable objects are candidates for removal from the long-term cache. By implementing this mechanism, cache pollution can be mitigated and cache space can be utilized effectively. Experimental results have revealed that the proposed approach can improve the performance up to 14.8% and 17.9% in terms of hit ratio (HR) compared to LRU and Least Frequently Used (LFU). In terms of byte hit ratio (BHR), the performance is improved up to 2.57% and 26.25%, and for latency saving ratio (LSR), the performance is improved up to 8.3% and 18.9%, compared to LRU and LFU. Although the simulation results have proven that work helps in improving the performance in terms of the hit ratio (HR), the performance in terms of the byte hit ratio (BHR) is not good enough since the cost and size of the predicted objects in the cache replacement process were not taken into consideration. Moreover, the training process requires long time and extra computational overhead.

In [10], an algorithm called Pre-IPGDSF# is developed by integrating Web caching and Web pre-fetching in Web servers. An algorithm called Intelligent Predictive Greedy Dual Size Frequency#, IPGDSF#, is used for caching. For pre-fetching a static pre-fetching method is used. Trace driven

analysis, using three different Web proxy server logs, is used to evaluate the effects of different replacement policies on the performance of a Web proxy server. Results indicate that, for larger cache sizes, Pre-IPGDSF# outperforms all other algorithms in terms of both hit rate and byte hit rate. Reference[11] proposed a hybrid technique based on combination of ANN and Particle Swarm Optimization (PSO) for classification Web object either to cache or not and generate rules from log data by using Rough Set technique on proxy server (Rough Neuro-PSO).

[12] reported an integrated caching and pre-fetching technique to reduce latency in mobile environment. The proposed model consist of bandwidth monitoring agent to find out current bandwidth usage, a prediction module to predict the number and the list of urls to be pre-fetched and a pre-fetch module to pre-fetch the web page and store them in a pre-fetch area. Simulation results show that the browser implemented in a mobile environment maintains almost constant web traffic even id pre-fetching is done and latency is reduced up to 40- 70%.

In [13] Vague improved Markov model is presented to perform the prediction. In this work, Vague rules are suggested to perform the pruning at different levels of Markov model. Once the prediction table is generated, the association mining will be implemented to identify the most effective next page. In this paper, an integrated model is suggested to improve the prediction accuracy and effectiveness.

In[14] a technique to remove the problem of cold cache pollution is proposed which is proved mathematically that it is better than the existing

LRU-Distance algorithm, the two modified LRU algorithms, LRU-Distance and SLRU proposed, to reduce cold cache pollution. These two algorithms are also simple to implement. But cold cache pollution is partially removed by these two modified form of the LRU.

An Intelligent Predictive Web caching algorithm, IPGDSF was investigated in [15],it capable of adapting its behavior based on access statistics. This algorithm is based on the GDSF algorithm; IPGDSF has compared with several cache replacement policies like LRU, GDSF, and GDSF for Web proxies, using a trace-driven simulation approach. the study show that IPGDSF outperforms all other algorithms in terms of hit rate as well as byte hit rate.

Reference [16]Presented an Energy Efficient Intelligent Agent (IA) controlled combined proxy pre-fetch-cache framework. The principle objective is to satisfy most of the clients' requests through a local proxy cache populated from various distributed proxies within the same or neighboring clusters and very few requests are sent to the remote origin server.

A replacement algorithm called ALIRS which is based on LIRS ALIRS algorithm was proposed in [17] to solve the scalability issue in the cache system. The authors were used parallel computing and Actor concurrency model to hide the expensive cost of replacement operations, which significantly improve the scalability. Experimental results show that ALIRS has both high cache hit ratio and high scalability.

Reference [18] provides the most comprehensive comparison of function based proxy cache replacement strategies. The comparisons are based on three important metrics, hit rate, byte hit rate and removal rate. The strategies were reported here are the instances of the parameters that provide the best result for the corresponding strategy. LRU and GDSF strategies are no longer "good-enough" strategies, they have demonstrated the trade-off between byte hit-rate and hit-rate when considering an object's recency, frequency and size characteristics.

A new algorithm called least Grade Replacement (LGR) is proposed in [19]by considering recency, frequency, perfect-history and size in replacing policy. The 2- and 4-way set associative caches were used to determine the optimal recency coefficients. The cache size was varied from 32k to 256k in the simulation. The results showed that the new algorithm (LGR) is better than LRU and LFU in terms of Hit Ratio (HR) and Byte Hit Ratio (BHR).

Reference [20] proposed a SEMALRU replacement policy by combining the semantic content and recency of web pages. It outperformed other policies in terms of Page Hit Ratio, Byte Hit Ratio and number of replacement as demonstrated in the text. The policy was tested in a simulated environment with the related and unrelated set of user access pattern. The parameters pertinent to cache replacement algorithms are computed and the results showing the improvement in the efficiency of the algorithm are furnished.

Reference[21] proposed and used the data envelopment analysis (DEA) as a technique that can be used to enhance the trace-driven simulation experiments that constitute the common methodology to study the object replacement strategies in web caching. The DEA model clearly showed that the cache size plays a crucial role in improving the performance of all the algorithms, for all the performance metrics under study. When the cache size increases, there is a general convergence of the efficiency scores towards the unity.

The implementation of a caching scheme for ad hoc networks was described in [22]. In this scheme the mobile nodes implement a local cache. This feature allows to intercept the forwarding requests and serve the documents requested directly using their local cache. Using the information obtained by the documents forwarded the nodes can redirect therequests to other nodes that are known to have the document requested. This caching scheme has been implemented using the network simulator NS-2. The paper describes the architecture, implementation details and customization parameters.

Reference [23] proposed a caching scheme which utilized a multi-level class information. A MLR (Multinomial Logistics Regression) based classifier is constructed using the information from web logs. Simulation results confirm that the model has good prediction capability and suggest that the proposed approach can improve the performance of the cache substantially.

A modification of the performance model of Proxy Cache Servers to a more powerful case when the inter-arrival times and the service times are generally distributed was proposed in [24]. The paper described the original proxy cache server model where the arrival process is a Poisson process and the service times are supposed to be exponentially distributed random variables. Then they calculate the basic performance parameters of the modified performance model using the well known Queueing Network Analysis (QNA) approximation method. The accuracy of the new model is validated by means of a simulation study over an extended range of test cases.

The work [25] presented a technique to classify whether a cached object is a One-Timers (OT) referenced only once or not. Statistical analysis of the workload shows that as much as 76% of objects are One-Timers (OT), Caching OT objects usually degrade the performance of all Web cache replacement algorithms .Simulation shows that classification may significantly enhance the performance of replacement algorithms with respect to the HR, the BHR and the DSR.

A game between an Internet Service (access)Provider (ISP) and content provider (CP) on a platform of end-user demand was considered in [27].A price-convex demand-response is motivated based on the delay-sensitive applications that are expected to be subjected to the assumed usage-priced priority service over best-effort service. The authors considered two-sided market with multi-class demand wherein one class (that under consideration herein) is delay-sensitive. Both the Internet and proposed Information Centric Network, encompassing Content Centric

Networking scenarios are considered. For the purposes, the first case is basically different in the polarity of the side-payment (from ISP to CP) and, more importantly here, in that content caching by the ISP is incented. A price convex demand-response model is extended to account for content caching. The corresponding Nash equilibrium are derived and studied numerically.

Reference [28]considered ASes that maintain peering agreements with each other for mutual benefit, and engage in content-level peering to leverage each others' cache contents. The authors propose a model of the interaction and the coordination between the caches managed by peering ASes. They address whether stable and efficient content-level peering can be implemented without explicit coordination between the neighboring ASes or alternatively, whether the interaction needs to rely on explicit announcements of content reach ability in order for the system to be stable. They show that content-level peering leads to stable cache configurations, both with and without coordination. If the ASes do coordinate, then coordination that avoids simultaneous updates by peering ISPs provides faster and more cost efficient convergence to a stable configuration. Furthermore, if the content popularity estimates are inaccurate, content-level peering is likely to lead to cost efficient cache allocations. They validate the analytical results using simulations on the measured peering topology of more than 600 ASes.

Research [29]tries to improve hit rates in proxy system by applying data mining technique. The data set were collected from proxy servers in the university and were investigated relationship based on several features.

The model was used to predict the future access websites. Association rule technique was applied to get the relation among Date, Time, Main Group web, Sub Group web, and Domain name for created model. The results showed that this technique can predict web content for the next day, moreover the future accesses of websites increased from 38.15% to 85.57%.

Reference [30] provides a framework to overcome the limitations of the existing Anti Spam SMTP Proxy server engine by restructuring the existing ASSP server engine. The authors dealed with spam detection as image spam, video spam that comes as attachments and a single line message that would direct to another URL with a trainable fuzzy classifier to build an automatic anti spam filter. Based on the rule, evaluations were done to predict whether the given mail is a valid or invalid and the report is updated in the database.

Reference[31]enhance traditional Web caching polices using supervised machine learning techniques such as a support vector machine, a naïve Bayesian classifier (NB), a decision tree (C4.5) and Size . It trained from Web proxy logs files to predict the object that would be revisited.HR increased by 30.15% and BHR increased by 32.43%.

Reference [32]proposed a hybrid of web proxy caching architecture by integrating forward and reverses proxy caching techniques to improve the performance of computer network.

Evaluation of web pre-fetching and caching algorithms has been studied in[33].They explain advantages and disadvantages to each algorithm and its own application area.

In the Study [34] a Response Time Gain Factor is included in this web object replacement algorithm with Size heterogeneity of a web object for performance improvement of the response speed, it makes a qualitative comparison between these policies and its performance object-hit ratio and an improvement of response speed.

Reference [35]proposes a splitting browser cache to two caches, instant cache to store the web object and durable cache to store the web objects that are visited more than the pre-specified threshold value.

Reference [36]proposed an Intelligent Predictive Web caching algorithm, IPGDSF, capable of adapting its behavior based on access statistics considers future frequency in calculating the key value.

Paper [37]concerned Web server log file analysis to discover knowledge and by applying Clustering and optimization technique to get user interest which is helpful or useful for giving suggestion about specific users's interest.

Reference [38] introduced advanced machine learning approaches for Web caching to decide either to cache or not to the cache server, which could be modeled as a classification problem. The challenges include identifying attributes ranking and significant improvements in the classification accuracy. Four methods are employed in this work; Classification and Regression Trees (CART), Multivariate Adaptive Regression Splines (MARS), Random Forest (RF) and TreeNet(TN) are used for classification on Web caching. The experimental results reveal that CART performed extremely well in classifying Web objects from the

existing log data and an excellent attribute to consider for an accomplishment of Web cache performance enhancement. Reference[39] Focus on a framework of a network of proxy caches. In a typical proxy cache network implementation, such as IRCache, each node in the network makes its own caching decisions based on the request patterns it observes. The authors develop algorithms for implementing a network of caches under both centralized and decentralized frameworks. The caching implementations are compared and contrasted using numerical computations. The results demonstrate that the performance of proxy caching networks can be improved when nodes also consider objects held by their neighbors.

Work[40]analyzed the confluence of the two effects through a tractable mathematical model that enables to establish the conditions under which pre-fetching reduces the average response time of a requested document. The model accommodates both passive client and proxy caching along with pre-fetching. The analysis is used to dynamically compute the "optimal" number of documents to pre-fetch in the subsequent client's idle (think) period. This optimal number is determined through a simple numerical procedure. Closed-form expressions for this optimal number are obtained for special yet important cases. Simulations are used to validate analysis and study the interactions among various system parameters.

In [41] machine learning method is try for a classification problem in Web caching that requires a decision to cache or not to cache Web objects in a proxy cache server. The experimental results reveal that CART performed extremely well in classifying Web objects from the existing log

data with a size of Web objects as a significant attribute for Web cache performance enhancement.

In[42]Rough Set analysis based on Inductive Learning methods to probe the significant attributes and generate the decision rules by executing caching strategy.

In [43]the data set were collected from proxy servers. The model was used to predict the future access websites. Association rule technique was applied to get the relation among Date, Time, Main Group web and Sub Group web, and Domain name for created model. The results showed that this technique can predict web content for the next day, moreover the future accesses of websites increased from38.15% to 85.57 %.

Reference [44]proposed a new object management policy that can be applied in the hybrid architecture by employing the upper level proxy cache having a reference table and employing the summary table in each proxy cache. The proposed solution numerically outperforms the previous solution from 72% to 94% in terms of response time.

In [45]Bayesian network (BN) learned from Web proxy logs file to predict the classes of objects to be re-visited or not. The trained classifiers were integrated with Web proxy caching to provide more effective proxy caching policies. From the simulation results,-BN-GDS achieve the best in HR.-BN-LRU and BN-DA achieved the best in BHR.

2.2 Old Intelligent Web Caching

Although the following studies with old date, they have been considered because they important and it is the starting point in this research:

In [46]ANN has been used for making cache replacement decision. An object is selected for replacement based on the rating returned by ANN Simulation result illustrate that -HR 86.60%to 100%-BHR93.36% to99.92%.

In [47] NNW are trained to classify cacheable objects from real world data sets using information known to be important in web proxy caching, such as frequency, recency and size. In simulation, the final NN achieve HR that are 86.60% of the optimal in the worst case and 100% of the optimal in the best case. BHR are 93.36% of the optimal in the worst case and 99.92% of the optimal in the best case.

In [48] trace driven analysis has been used to evaluate the effects of different replacement policies on the performance of a Web server. The authors propose a modification of GDSF policy, GDSF#, which allows augmenting or weakening the impact of size or frequency or both on HR and BHR. The simulation results show that our proposed replacement policy GDSF# gives close to perfect performance in both the important metrics: HR and BHR.

In [49]A new architecture of cache farming with the recommender system concept to manage users' requirements was proposed. This solution helps reducing the retrieval time and also increasing the hit rate.

Reference [50] proposed algorithm that applied set of fuzzy control rules to identify the pages to evict from the cache, the authors test the performance via trace driven simulations using traces collected on various proxy servers.

Reference [51]proposed a similarity-aware multi-cache architecture, in which the cached web documents are organized into a number of sub-caches according to their content similarity. A predictor is then developed to predict the cached documents a user might access next. Once a pre-fetching plan was formed, a set of agents are employed to work together for pre-fetching document between proxy caches and browsing clients. Preliminary experiments have shown that the predictor offers superior performance when compared with some existing prediction algorithms.

Study[52] Proposed a fuzzy algorithm in which the decision parameters are treated as fuzzy variables. A simulation is also performed and the results are compared with Optimal, LRU, and LFU replacement algorithms. Nine different workloads were examined and as it was shown the fuzzy approach has better performance over the LFU and LRU algorithms in eight of these workloads. Results say that, the fuzzy approach is suitable for looping, probabilistic and temporal pattern of reference and it even is better in mixed reference patterns.

The feasibility and performance of a locally distributed, self-organizing network proxy has been investigated in [53].The main observation made in the reference is that existing strategies for distributed web object caching do not motivate use because they generally attempt to

maximize server performance and often cannot guarantee performance for clients. This guarantee is provided in the reference by restricting the attention to the local environment. Trace-driven simulation has been used to evaluate the performance of the proposed scheme. Results suggest that cache hit rate in the local environment is highly dependent on population size and cache size. Further, the benefits of distributed caching can be realized for small populations.

Study [54] proposed a novel cooperative proxy-and-client caching system that combines the advantages of both proxy caching and peer-to-peer (P2P) client communications. The authors propose a comprehensive suite of protocols to facilitate the interactions among different network entities. They also develop an efficient cache allocation algorithm to minimize the aggregated transmission cost of the whole system. The simulation results demonstrate that the proposed approach achieves remarkably lower transmission cost. Moreover, it is much more robust than a pure P2P system in the presence of node failures.

In work [55]A theoretical model has been introduced to analyze the access cost of placing a set of object copies in the cache hierarchy, under which the object placement problem is formulated as an optimization problem. The problem is proved to be divided into sub problems, and a dynamic programming algorithm is proposed to obtain the optimal solution. Performance of different caching strategies is evaluated using simulations. It is shown that the proposed algorithm outperforms other cache placement strategies in hierarchical caching systems.

A dynamic and scalable caching algorithm of proxy server with a finite storage size for multimedia objects is proposed in [56]. Caching sequences for videos are obtained to decrease both the buffer size and the required bandwidth and saved into metafiles in advance. Caching and replacing algorithms for multimedia objects based on the metafiles have been presented. Experimental results show the superior performance of the proposed algorithm.

Reference [57] propose a new proxy-level web caching mechanism that takes into account aggregate patterns observed in user object requests. The integrated caching mechanism consists of a quasi-static portion that exploits historical request patterns, as well as a dynamic portion that handles deviations from normal usage patterns. This approach captures both the static and the dynamic dimensions of user web requests. The performance of the proposed mechanism is empirically tested against the popular LRU caching policy using an actual proxy trace dataset. The results demonstrate that the mechanism performs favorably versus LRU.

Study [58] addressed the short-term pre-fetching problem on a Web cache environment using an algorithm (clustWeb) for clustering inter-site Web pages. The proposed scheme efficiently integrates Web caching and pre-fetching. According to this scheme, each time a user requests an object, the proxy fetches all the objects which are in the same cluster with the requested object. Specifically, the proxy traces are represented by a Web navigational graph. Then, the clusters have been resulted by partitioning this graph, where the number of clusters is not determined at priori but it is dynamically estimated by the confidence and support measures. Using real

data, the authors show the robustness and efficiency of the proposed method.

Study [60] proposed a novel cooperative caching scheme called Group Caching (GC) which allows each mobile host and its 1-hop neighbors form a group. The caching status is exchanged and maintained periodically in a group. By using the proposed Group Caching, the caching space in mobile hosts can be efficiently utilized and thus the redundancy of cached data is decreased and the average access latency is reduced. The authors evaluate the performance of the Group Caching by using NS2 and compare it with the existing schemes such as Cache Data and Zone Cooperative. The experimental results show that the cache hit ratio is increased by about 3%~30% and the average latency is reduced by about 5%~25% compared with other schemes.

In [61]behavior of LRU, LFU and FIFO replacement algorithms has been analyzed, a new Replacement Algorithm named MFMR (Most Frequent with Maximum Reusability) is proposed for proxy server cache Level1 (L1) which works about 16% better than existing algorithms considered in this paper. Also a new replacement policy for storage cache of proxy server to be Level2 (L2) which named AF_LRU (Average Frequency, Least Recently Used) is proposed. Simulation results show that pair of MFMR and AF-LRU is approximately 28% better than other existing pairs of replacement algorithms considered.

Reference [62] dealt with fragment level caching of dynamically generated web content in proxies that are closer to end users. The authors

proposed and evaluated a fragment caching scheme using Bloom filters. It is the Basic Hint-Based Scheme whereby hints are sent together with each user request. The solution makes use of Bloom filter. The analytical results indicate that the solution is feasible, help reduce load from Web site servers and provide important network traffic savings.

Reference [63] presented an extensive evaluation of the request filtering in hierarchy of proxy cache. Using the proposed ADF (Aggregation, Disaggregation and Filtering) model as well as entropy as metric for web traffic characterization, the authors evaluate how locality of reference changes as the streams of requests pass through a hierarchy of caches.

In [64] An adaptive hybrid algorithm has been developed for reducing web traffic. Intelligent agents are used for monitoring the web traffic. Depending upon the bandwidth usage, user's preferences, server and browser capabilities, intelligent agents use the best techniques to achieve maximum traffic reduction. Web caching, compression, filtering, optimization of HTML tags, and traffic dispersion are incorporated into this adaptive selection. Using this new hybrid technique, latency is reduced to 20 – 60 % and cache hit ratio is increased 40 – 82 %.

2.3 Multi-Agents Coordination

The following papers present some studies in multi-agents coordination:

In [66] cache cooperation under a game theoretical framework was model. The authors show how cache cooperation policy can allow the system to converge to a Pareto optimal configuration. The work show how cooperation impacts network caching performance and how it take advantage of the structural properties of the underlying network.

Reference [67] proposed a learning mechanism that allows an artificial agent to construct and exploit a representation of its surrounding space with minimal preconceptions about its environment. This representation is based on a data structure that encodes possibilities of behaviors afforded by the current context. The behaviors are modeled in the form of sequences of interactions. Over time, the agent learns to associate sequences of interactions with the presence of certain elements of the environment in certain locations in the agent's surrounding space. The agent uses this emergent relation between objects and possibilities of interactions to construct and maintain a representation of the surrounding space based on sequences of interactions.

A statistical caching mechanism which makes use of prior knowledge (statistical data) to predict the pattern of user movement was proposed in [68]and then replicates/migrates the cache objects among different proxies. The authors proposed a statistical inference based heuristic search algorithm to accommodate dynamic mobile data access in the mobile learning environment.

A guided tour of some research on the topic of agent coordination and a historical survey about some coordination models and languages for multi-agent systems was presented in[69].The authors show how some coordination models have been adapted to different network infrastructures, distinguishing between pre-WWW and WWW based coordination architectures. They show that the advent of the new programming paradigms of Web Services and the Semantic Web is prompting the definition of a new family of coordination models and languages, useful to describe multi-agent systems suitable for these new infrastructures.

Paper [70]focused exclusively on some exceptions occurring in the sub-task level at runtime and not on exceptions concerned with handling agents or information of alternative problem solving method.

In [71]a new testbed for multi-agent coordination algorithms was described to builds upon the RobocupRescue platform. It testbed achieves the goals in a number of ways, by using the RoboCupRescue platform to generate very realistic scenarios that incorporate a high dose of dynamism and uncertainty. using centralized or decentralized approaches depending on the simulation settings. These problems range from logistics planning, to sequential decision making, and resource allocation.

Reference [72]Presented four different coordination mechanisms based on task sharing. Three of these mechanisms are communication-based: central coordination, contract Net coordination and Brown coordination, while the last one is zone defence coordination and is based on conventions.

Reference [73]Compared the current state of the art techniques at solving some of problems Schweppe identified, describes the agent coordination algorithms that are used, and suggests some future research opportunities on applying agent coordination algorithms, that have not previously been used, to microgrids.

In [74] the authors survey a general study of coordination, including, the nature of coordination, coordination mechanisms, coordination approaches, relationship among coordination mechanisms and approaches, coordination methodologies, conversationalaspects of coordination and software architectures.

In[75] the authors studied the problem of verification (with n agents) and computation (with two agents) of a strong Nash equilibrium (SNE). A number of results for Nash equilibrium (NE) are known, but that concept is inappropriate when coalitions are an issue. They showed that the instances from the ubiquitous NE benchmark testbed, GAMUT, are not suitable for testing SNE–finding algorithms because all the instances either admit pure SNEs or do not admit any SNE. Then they compared different configurations of the algorithm using a new instance generator to identify the best one. It turns out that SNE finding takes about 100 times as long as NE finding.

In [76] The authors proved that in a normal form n-player game with m actions for each player, there exists an approximate Nash equilibrium where each player randomizes uniformly among a set of O(log m + log n) pure strategies. This result induces an N log log N algorithm for computing

an approximate Nash equilibrium in games where the number of actions is polynomial in the number of player. In addition, they established an inverse connection between the entropy of Nash equilibrium in the game, and the time it takes to find such an approximate Nash equilibrium using the random sampling algorithm.

2.4 Multi-Agents Learning

The following papers present some studies in multi-agents Learning:

In [77] a hierarchical method to learn equilibrium strategy in continuous games was developed. Hierarchy has been used to break the continuous domain of strategies into discrete sets of hierarchical strategies. The algorithm is proved to converge to Nash-Equilibrium in a specific class of games with dominant strategies. Then, it is applied to some other games and the convergence in shown. This approach is common in RL algorithms that they are applied to problem where no proof of convergence exits. The results showed that the algorithm may converge when the conditions of convergence are not satisfied, or may learn to oscillate in vicinity of the equilibrium.

Reference [78] concerned with a two-player nonzero-sum differential game in the case when players are informed about the current position. The authors considered the game in control with guide strategies. The construction of universal strategies is given both for the case of continuous and discontinuous value functions. The existence of a discontinuous value function is established. The continuous value function does not exist in the general case. In addition, they show the example of smooth value function not being a solution of the system of Hamilton Jacobi equation.

Reference [79] presented an algorithm for hierarchical Nash-Cournot learning in electricity markets. Using this method the bidding agents in an electricity market were able to get to the Nash-Cournot equilibrium faster and with the maximum profit gains. The market simulation results showed

that the presented algorithm was fast and convergent to the Nash equilibrium because of its hierarchical structure. In constructing this algorithm, aspects from both game theory and reinforcement learning were used. In each step of learning, a simple bimatrix game was constructed. The agents learned the equilibrium in that game, and then by means of hierarchy, they were able to find the accurate equilibrium in their continuous infinite domain of bidding.

The simulation studies showed that the algorithm was capable of learning even during system contingencies and different demand profiles. It was shown that line congestion could cause market power for some players and consequently raise the market clearing price (MCP) over its competitive level. Also it was discussed how demand side management programs and price sensible loads could control this market power and reduce the market prices.

In [80] cache placement strategies and their performance in cooperative hierarchical caching environments have been studied. A theoretical model has introduced to analyze the access cost of placing a set of object copies in the routing path. Using this model, the object placement problem can be formulated as an optimization problem. It is proved that the problem can be divided into sub problems, thus optimal solutions can be obtained by using dynamic programming. It is further proved that if some nodes are known to be in the optimal solution, the calculation cost of the dynamic programming algorithms can be reduced. A heuristic greedy algorithm is also presented for efficient implementation. Performance of these strategies is evaluated using simulations under both synthetic

workload traces and real workload traces. It is shown that both the optimal and the heuristic strategies perform well in cooperative hierarchical caching systems

An adaptive Q-Learning-based HTTP Adaptive Streaming (HAS) client is proposed in [81]. In contrast to existing heuristics, the proposed HAS client dynamically learns the optimal behavior corresponding to the current network environment. Considering multiple aspects of video quality, a tunable reward function has been constructed, giving the opportunity to focus on different aspects of the Quality of Experience, the quality as perceived by the end-user. The proposed HAS client has been thoroughly evaluated using a network-based simulator, investigating multiple reward conjurations and Reinforcement Learning specific settings. The evaluations show that the proposed client can outperform standard HAS in the evaluated networking environments.

Paper [82] studies repeated interactions between an agent and an opponent that changes its strategy over time (it is non-stationary). The authors proposed a frame-work for fast learning changing non-stationary strategies. The agent uses decision trees to learn the most up to date opponent's strategy. Then, its learned model is continuously re-evaluated to assess strategy switches. The method detects such strategy switches by measuring tree similarities. Aside from its fast learning process, decision trees can provide an easy interpretation of the opponent model. They evaluated the proposed approach in the iterated prisoner's dilemma, outperforming state of the art algorithms in predictive accuracy when facing non-stationary strategies.

A new algorithm called Probably Optimistic Transition (POT) was introduced in paper[83], with which an agent can be greedier than with existing algorithms and perform well with a very wide range of parameter values. POT derived by letting the agent utilize not only Bayesian optimal reasoning but also the information of potentially true MDP, an agent with POT adaptively changes the degree of optimism as it learns where a true MDP potentially lies. With a larger than optimal parameter value, the existing algorithms usually maintain too much optimism and over explore. With a smaller than optimal parameter value, the existing algorithms are not optimistic enough and become stuck into a sub-optimal state. With POT, they solved this issue by letting an agent have adaptive degrees of optimism. To do so, they relaxed the requirement placed by optimism in the face of uncertainty principle.

Reference [84] introduced a new algorithm called Coco-Q, that is convergent and produces interesting solutions to challenging stochastic games when utility is transferable and binding agreements are possible. The authors show that coco values can also be defined for stochastic games and can be learned using a simple variant of Q-learning that is provably convergent. They provide a set of examples showing how the strategies learned by the Coco-Q algorithm relate to those learned by existing multi-agent Q-learning algorithms.

Reference [85]proposed new convergent Q-learning algorithms that combine elements of policy iteration and classical Q-learning/value iteration. The main difference from the standard policy iteration approach is in the policy evaluation phase: instead of solving a linear system of

equations, the algorithm solves an optimal stopping problem inexactly with a finite number of value iterations.

The following table classifies the previous studies based on study's date and topics:

Table 2.1: Classification of the Previous Studies

Topic \ Year	2000-2004	2005- 2008	2009-2012	2013-2014	Total
Intelligent Web Caching	0	6	16	6	28
Fuzzy Logic	1	1	3	1	6
Multi-agent Coordination	3	0	4	2	9
Reinforcement Learning	0	3	1	5	9
Survey of Web Caching	0	0	3	2	5
Developing Cache Replacement Algorithms	0	7	4	1	12
Developing Cache Scheme	1	3	2	3	9
Proxy Cache Performance Analysis and Evaluation	1	1	4	0	6
Others	2	0	0	0	2
Books	2	3	1	1	7
Total	10	24	38	21	

CHAPTER 3: MODEL ANALYSIS

3.1 The New Model

Caching may be performed at different levels in a computer network. This work deals with proxy-level caching. Proxy caching is widely utilized by computer network administrators to reduce user delays and to alleviate Internet congestion.

3.1.1 Model Common Properties

The proposed model consists of four common properties:

1-Distributed Manner: The knowledge required to solve some problem is not reside in a single resource or agent so that the cooperation of many individual agent to solve the problem is needed.

2-Speed: Each agent has his own local processor and memory.

3-Efficiency: not all knowledge is needed for all tasks, agent used only part of the knowledge required to solve the problem.

4-Reliability: Multi-agent System more reliable because there would be multiple agents in the setup which provide some particular functionality or service ,if an agent resource providing some functionality dies, another agent may take over.

3.1.2 Model Detailed Properties

The properties of our model revolve around three issues:

First, providing the characteristics of intelligent agent:

- Autonomous: the agents in our model can act independently. Each

agent is independent with local tasks it has the capabilities of problem-solving and decision-making. Each agent acts as a centralized management system.

- Reactivity: the agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.
- Proactiveness: agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives.
- Social ability: agents are capable of interacting with other agents in order to satisfy their design objectives.

Second, considering four mechanisms in dealing with multi agents system:

- Cooperation: It is the process of sharing responsibilities in satisfying shared goal and generating dependent roles in joint activities.
- Coordination: It is the process of management of agents' activities so that they coordinate their deeds with each other in order to share resources, meets their own interests.
- Independence: every agent in our model is able to work concurrently and relatively independently, it has its own goal to increase hit ratio and byte hit ratio but it is also capable of coordinating with other agents in order to achieve a common goal.
- Agents Communication: Agents communicate among themselves by message passing, an agent can be permanently ready to receive messages from other agents and, at the same time, it can carry out its own computational tasks.

Agent Communication Language (ACL) is a standard language for agent communications, the most popular ACL's: FIPA "foundation for intelligent physical agent" has been used.

Third, the proposed model must contain some level of intelligence so we add artificial intelligence techniques such as:

- Reasoning: The ultimate goal of fuzzy logic is to provide foundations

for approximate reasoning with imprecise propositions, using fuzzy set theory as the principal tool. When it is applied to rule-based expert systems, there are two basic issues to be concerned with: first, the use of linguistic variables in the representation of experts' knowledge or rules, and second, the deduction of conclusions from observations and rules in a knowledge base. A fuzzy logic based system model is a knowledge based system comprising of rules

- Learning: learning provides an excellent method for optimizing the agent's action in such an environment. Reinforcement learning (RL) is a generic name given to a family of techniques in which an agent tries to learn a task by directly interacting with the environment. Multi-agent reinforcement learning in which many agents are simultaneously learning by interacting with the environment and with each other.

There are two popular learning algorithms for single-agent systems, value iteration and Q-learning, they can be extended to the multi-agent case. We use to implement the second one in this work.

3.1.3 Model Description

The new multi agent model consists of the following agents:

- Monitoring Agents: The monitoring agent is capable of accessing "access log file" which is created by the proxy server and read data.
- Cache Cleaner Agents: The main task of the cache cleaner agent is to clean the main proxy server's cache proactively according to the web object size, frequency and time.
- Coordinator Agents: Its main task is to coordinate between cache cleaner agents.
- Fuzzy logic has been used to model data that read from access log file. Fuzzy base scheme considers for fuzzification three input parameters the web object size (S), The web object Frequency (F),

and the time (T).The output parameter is the cache clean up priority (CP),which helps in making decision to clean the proxy cache.

3.1.4 Model Architecture

The model architecture shown in Figure 3.1, this architecture consists of three modules:

1-The Monitoring Module: contains the monitoring agent. It is a reactive agent that monitors the proxy cache. This agent works by using a fast response behavior. It provides information that allows the other agents to take a decision.

2- The Cleanup Module: contains the parent and child cache cleaner agents. Their task is to clean up the parent and child caches simultaneously according to web object sizes, frequencies, and times.

3- Coordination Module: contains the coordinator agent. Its main task is to coordinate between cache cleaner agents.

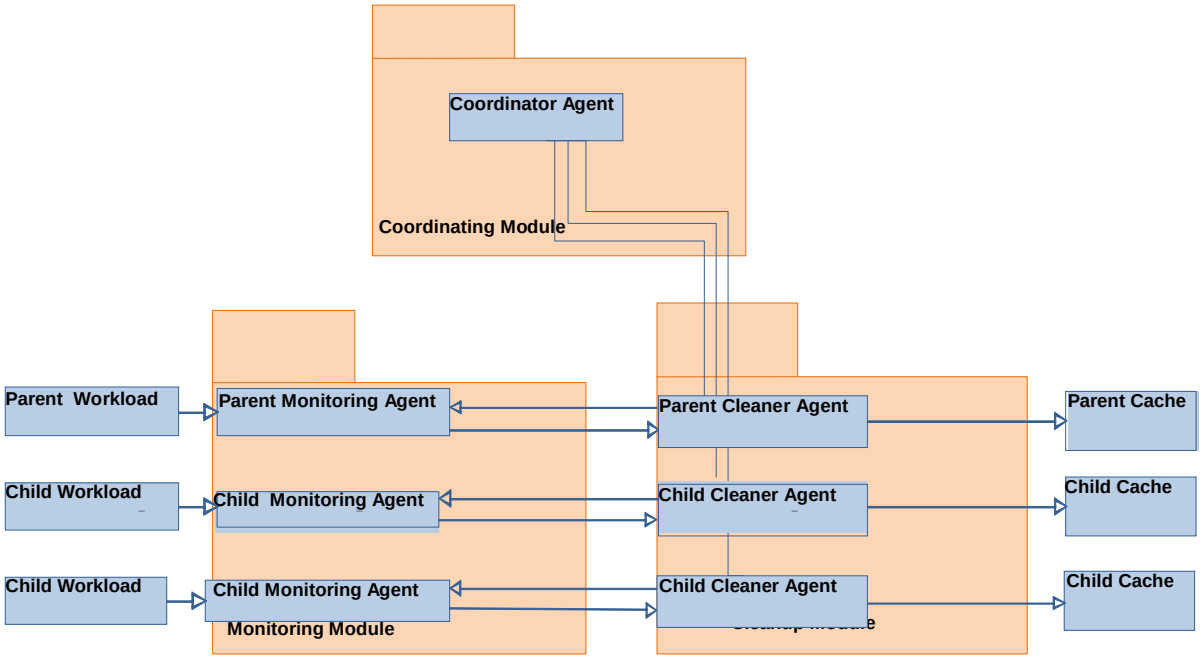


Figure 3.1: The Model Architecture

3.1.5 Agents' Tasks

• The monitoring agent reads and stores web workload which is generated by the Webtraff simulator in a local database. Figure 3.2 shows a sample of the Web workload generated by Webtraff. The monitoring agent then converts the input data to linguistic values using fuzzification. Finally, the monitoring agent communicates with the cache cleaner agent and gives it the requested data.

TIMESTAMP	DOC_ID	SIZE
0.01831	1	2885
0.17150	1	2885
1.37429	1	2885
4.24071	0	4241
5.54242	1	2885
7.12107	1	2885
7.61197	1	2885
8.13901	3	4245
8.14961	0	4241
8.38060	0	4241
9.10558	4	1624
9.50900	2	16782
10.12284	0	4241
10.31290	0	4241
11.11873	0	4241
12.12238	1	2885
12.30627	0	4241

Figure 3.2 : Sample of Web Workload Format Used in WebTraff

- The main task of the parent cache cleaner agent is to clean the main proxy cache proactively according to the web object size, frequency, and time that are requested from the parent monitoring agent. On the other hand, the child cache cleaner agent cleans the sub proxy cache depending on the Web object's frequency and recency of use.
- The main task of the coordinator agent is to receive information about the web objects from parent and child cache cleaner agent, take a suitable decision and send acknowledgments to them.

3.1.6 Agents' Behaviors

Cache cleaner agent periodically requests the three parameters values from the monitoring agent. This can be achieved by using a Ticker Behavior. That is, on each tick, add a new request. On the other hand, the monitoring agent waits for cache cleaner agents' requests. Monitoring agent executes a cyclic behavior to serve the requests. Finally, it executes a one-shot behavior updating its local database.

3.1.7 Agent's Interactions

Multi-agent system contains a number of agents which interact through communication, they are able to act in an environment. We need a model of the environment in which these agents will act.

3.1.8 Agents' Communication

Since child and parent cache agent has a partial view of the caching system they cannot observe the global state of a dynamic environment, and therefore they must communicate with each other to share the information needed for deciding which action to take. Communication decision becomes an important part of agent decision problem. An agent cannot observe directly the local state of other agent, the agent has to perform communication action just after the previous action finishes and before the next action is chosen.

- **Monitoring Agent - Cache Cleaner Agent Communication**

  - Monitoring agent continuously waits for the cache cleaner agent request's.
  - Cache cleaner agent receives the requested data.
  - Cache cleaner agent applies rules.
  - Cache cleaner agent finds the web object cleanup priority and deletes it.
  - Cache Cleaner agent sends delete notify message to the monitoring agent and terminates.
  - The Monitoring Agent begins to read a new data from the cache.
  - The Monitoring Agent updates its local database.

- **Cache Cleaner Agent – Coordinator Agent Communication**

    - Cache cleaner agent sends the information of the web object with the medium priority to coordinator agent.
    - Coordinator agent applies coordinator rules and sent acknowledgment to the cleaner agent.

3.2 Analysis using Agent Message Methodology

3.2.1 Organization view (OV)

This shows Concrete Entities (Agents, Organizations, Roles, Resources) in the system and its environment and coarse-grained relationships between them (aggregation, power, and acquaintance relationships), it gives an overall view of the system, its environment, and its global functionality. the structure and the behaviour of entities such as Organization.

Figure 3.3 describes structural relationships in the organization view, which is considered a cache cleaner organization consists of:

- Parent Monitoring Agent
- Child Monitoring Agent1
- Child Monitoring Agent2
- Parent Cleaner Agent
- Child Cleaner Agent1
- Child Cleaner Agent2
- Coordinator Agent

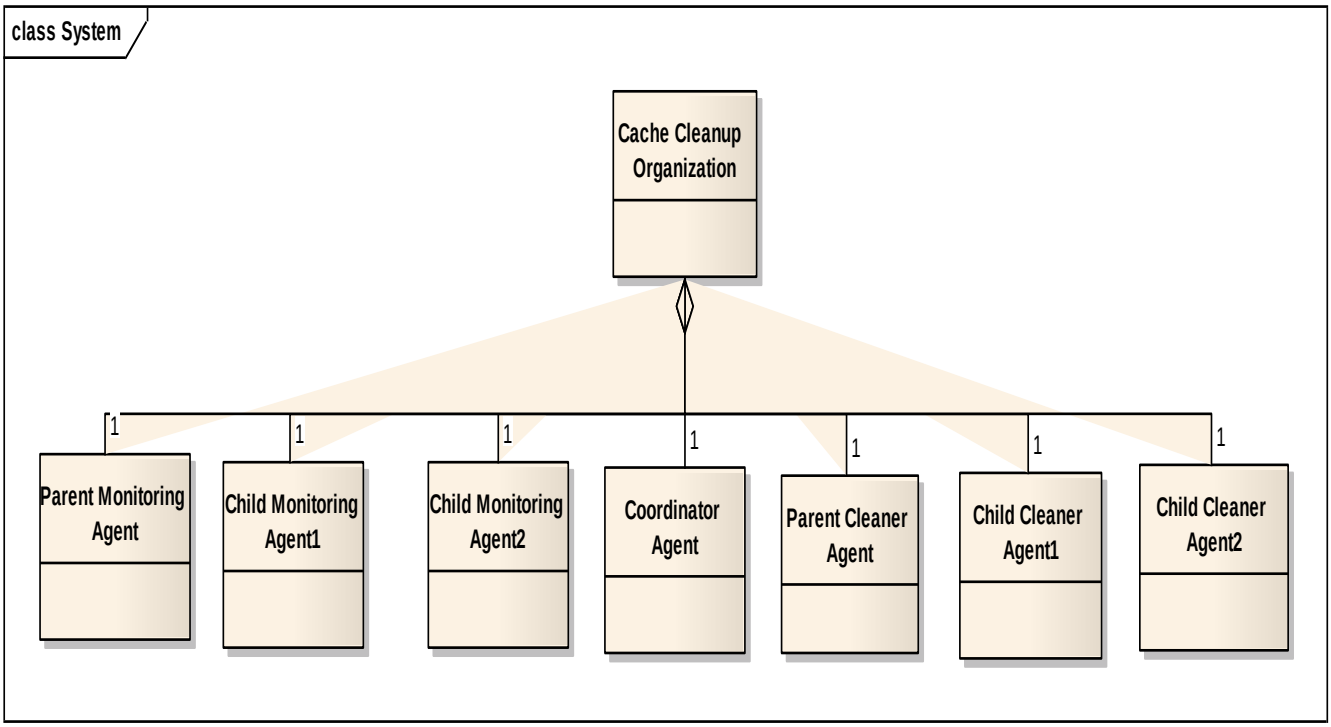


Figure 3.3. Organization Diagram (Structural relationships)

3.2.2 Goal Task View (GTV)

This shows Goals, Tasks, Situations and the dependencies among them. Goals and Tasks both have attributes of type Situation, so that they can be linked by logical dependencies to form graphs that show e.g. decomposition of high-level Goals into sub-goals, and how Tasks can be performed to achieve Goals.

Goal/task decomposition approaches are based on functional decomposition, it gives an overall view of the system roles, goals and tasks are systematically analyzed in order to determine the resolution conditions, problem-solving methods, decomposition and failure treatment.

Figurer 3.4 shows cache cleanup task decomposes to three sub tasks:

- Cache Monitoring task.
- Cache cleanup task.
- Coordination task.

Each sub task decomposes also to its sub tasks.

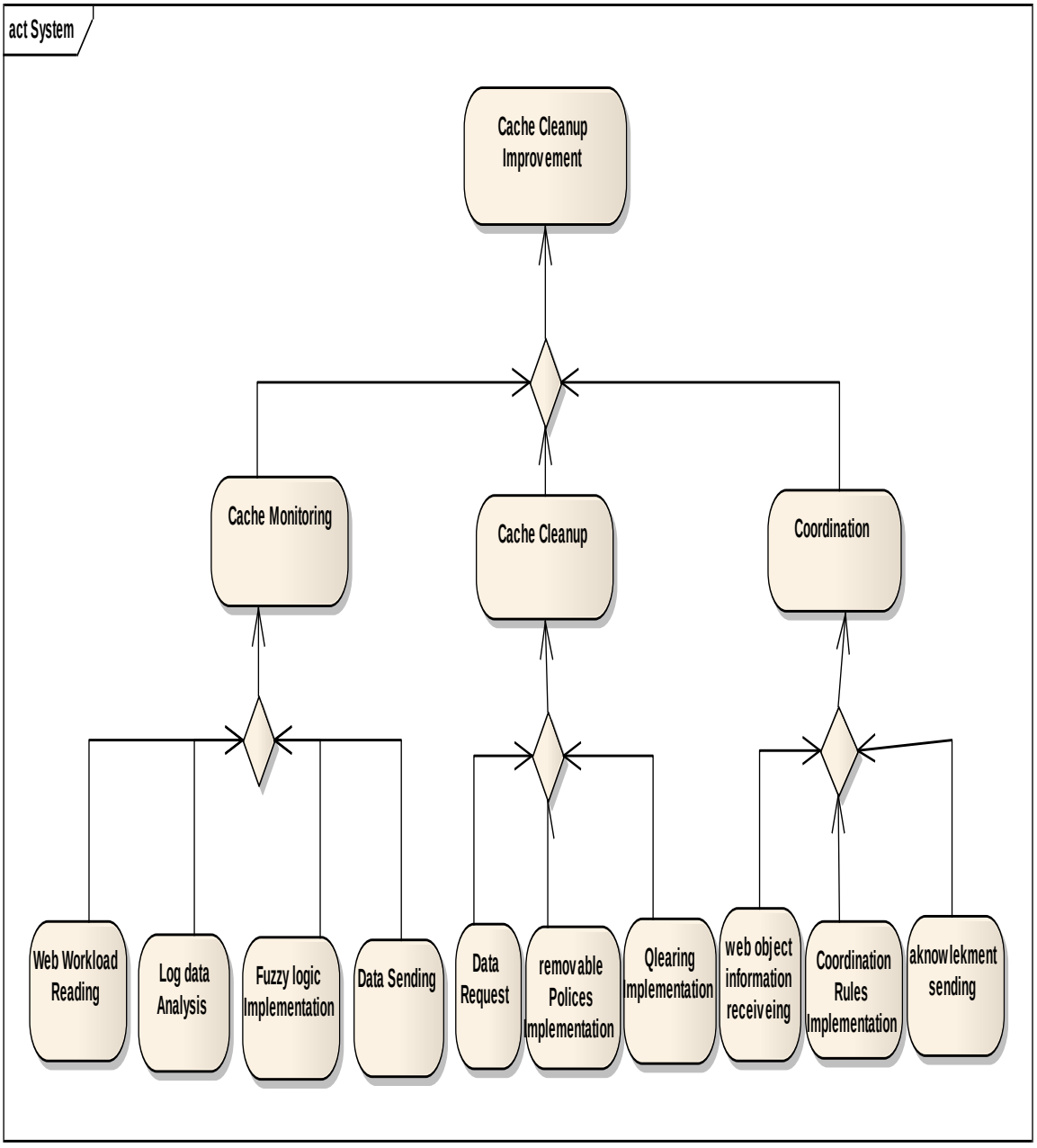


Figure 3.4: Goal/Task Implication diagram

3.2.3 Domain View (DV) in level 0

Domain view shows the domain specific concepts and relations that are relevant for the system under development. The domain view can be represented by means of typical UML class diagrams where classes represent domain specific concepts and named association represents domain specific relations

Figure 3.5 describes domain view in level 0 in three modules:

- Monitoring Module: contains the monitoring agents
- Cleanup Module: contains the cache cleaner agents
- Coordination Module: contains the coordinator agents.
  There is a aggregation relation in each module between the super class and their sub classes.

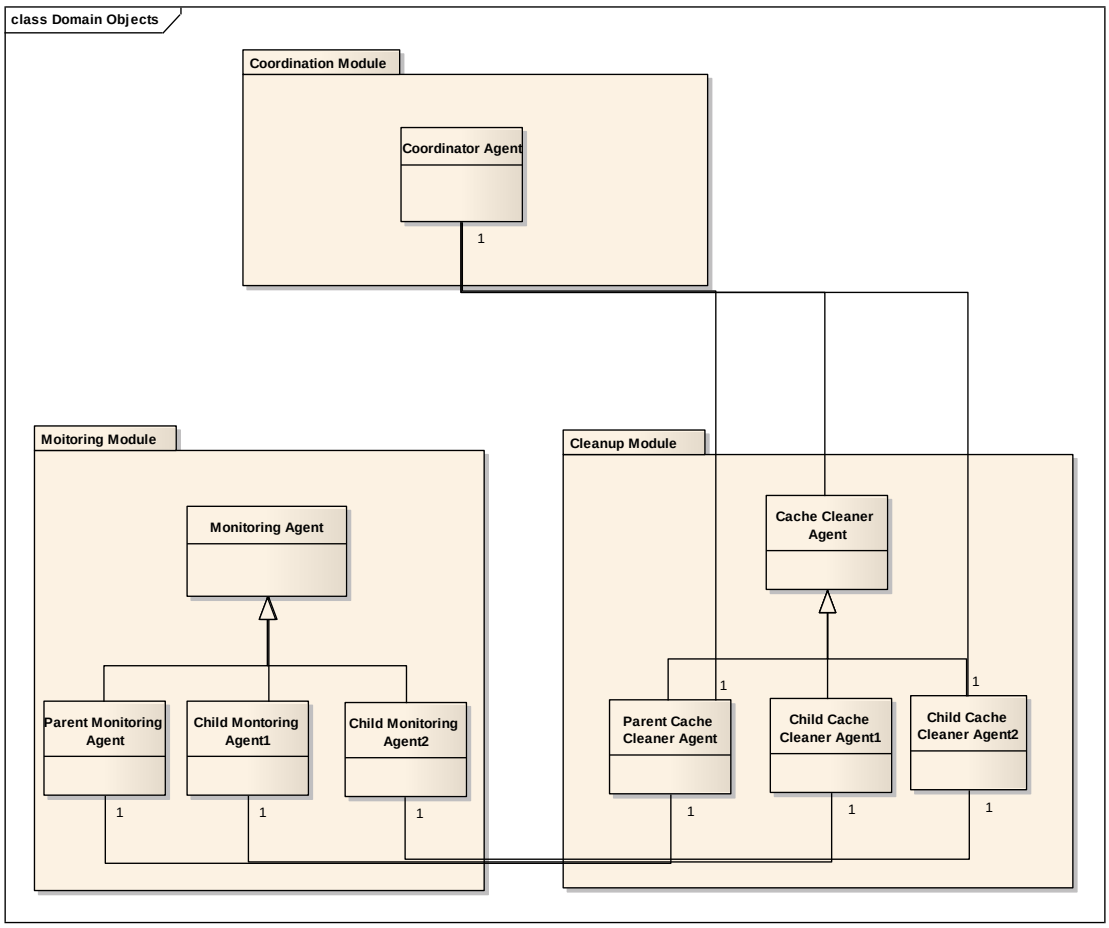


Figure 3.5: Level 0 Domain Diagram

3.2.4 Domain View (DV) in level 1

Figure 3.6 shows the three modules in details in level 1, class's attributes and actions has been added.

There are three inner classes:

- Fuzziication Class: it is inner class in the monitoring Agent class to perform the fuzzy logic phases.
- Request Performer Class: it is inner class in the cache cleaner agent class to requested data from the monitoring agent class.

Performer Class: it is inner class in the coordinator agent class to perform the coordination between parent and child cleaner agents

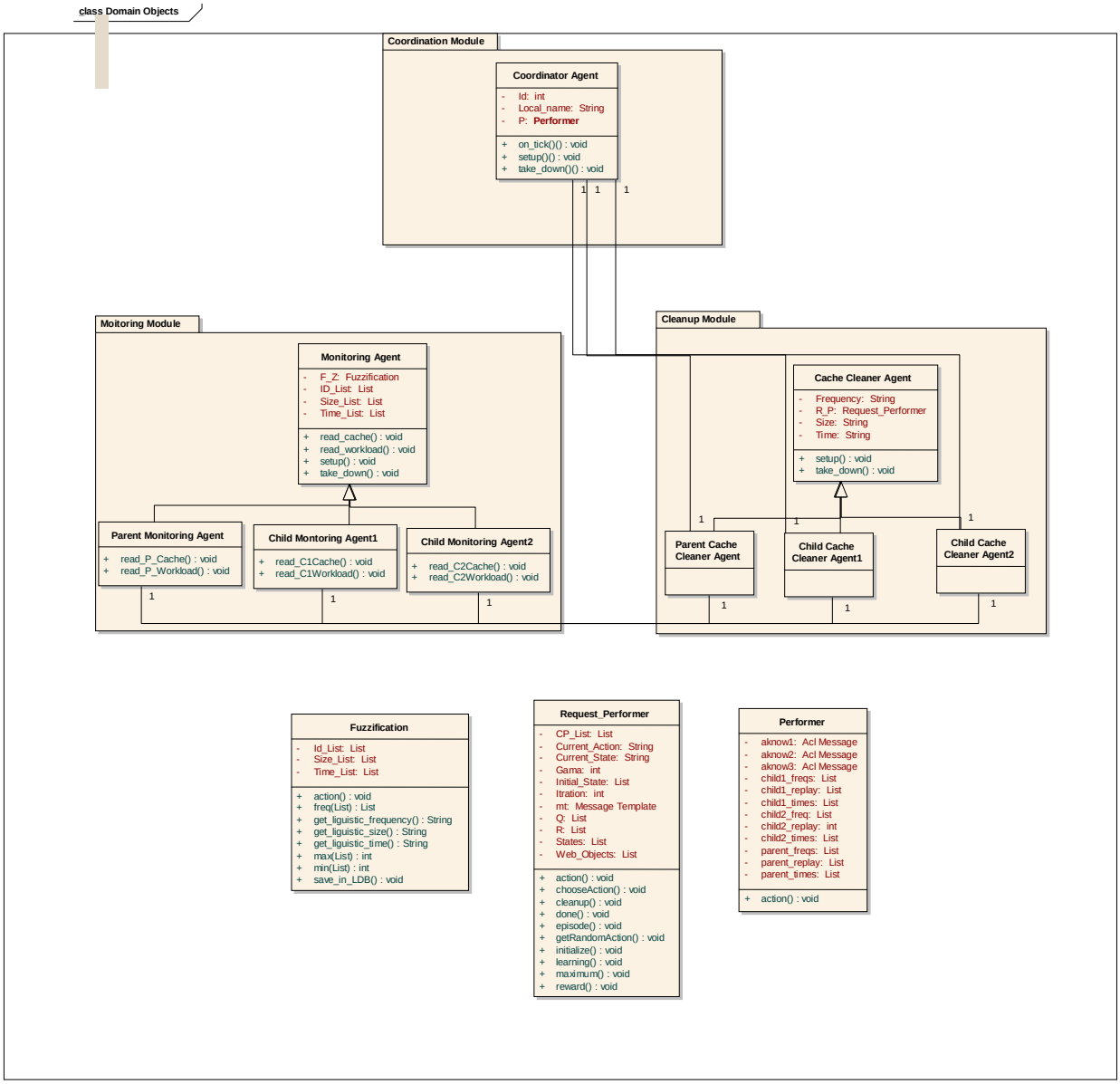


Figure 3.6: Level 1Domain Diagram

3.2.5 Interaction View (IV)

This view highlights which, why and when agents need to communicate leaving all the details about how the communication takes place to the design process.

Figure 3.7 shows interactions among agents, the information supplied/achieved by each participant, the events that trigger the interaction, other relevant effects of the interaction can also be considered.

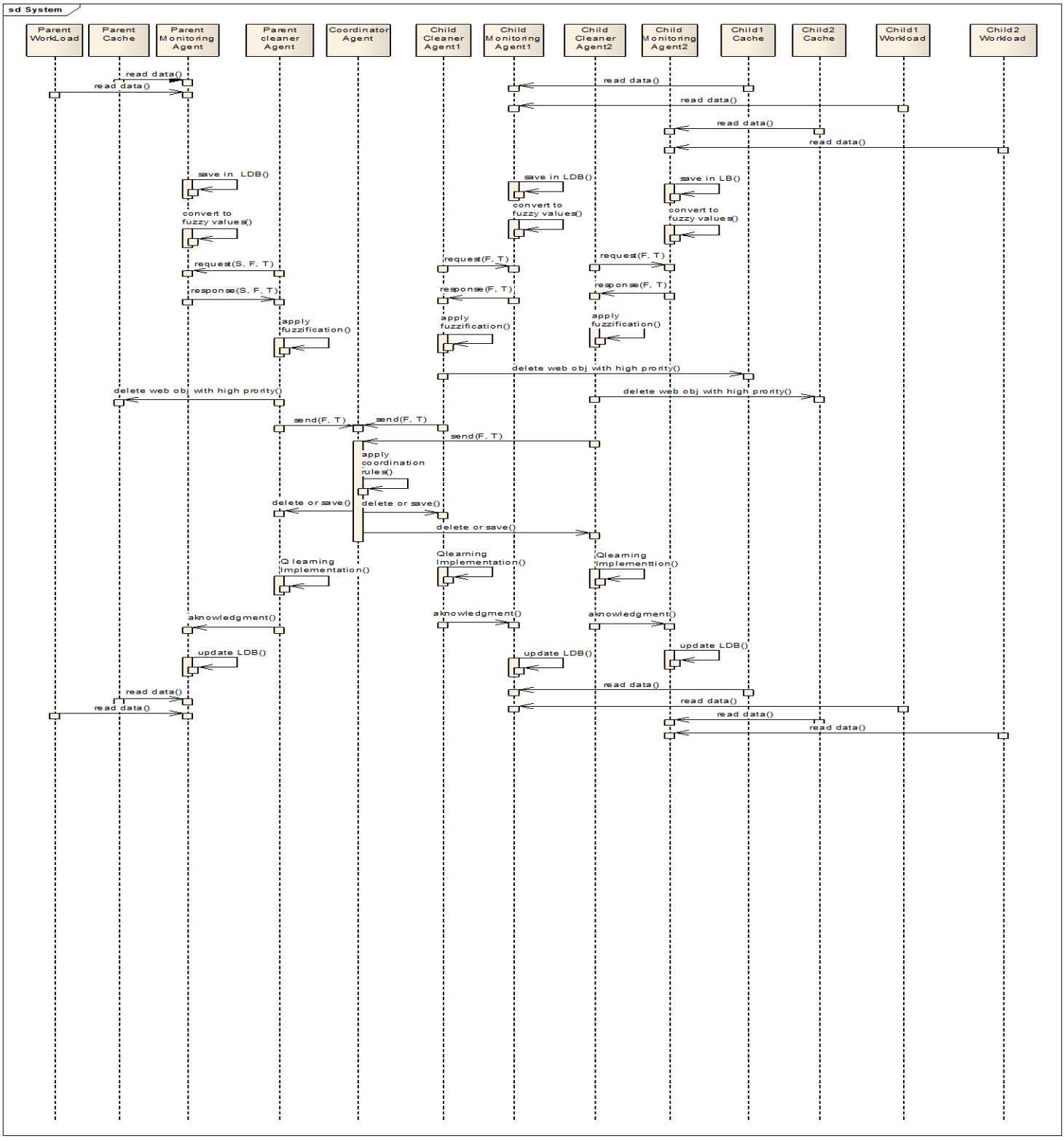


Figure 3.7: Interaction Diagram

CHAPTER 4: PARENT AND CHILD CLEANER AGENT COORDINATION AND LEARNING

4.1 Parent and Child Cleaner Agents Coordination

Coordination is needed between parent and child cleaner agents to achieve the cleanup task in efficient way.

After implementing fuzzy logic, the web object with the high priority has been removed from the cache. Central Reactive coordinator agent has been added to achieve the cleanup task in efficient way. It applies a coordination Rules.

4.1.1 Assumptions

To apply the Coordination Rules we assume that:
Parent and child agents have a common goal to increase HR,BHR.

- Web object with medium priority is found in parent and child caches.
- Web object has the same size in parent and child cache.
- Web object's frequency and request time factors have been considered.
- Parent and child agents have two possible actions can perform, D ("delete") or S ("save") Ac = {D, S}

4.1.2 Coordination Rules' Abbreviations

The following Abbreviations has been used to develop the coordination rules:

- P freq: web object's frequency in parent cache
- C1 freq: web object's frequency in child 1 cache
- C2 freq: web object's frequency in child 2 cache
- P time: web object's request time in parent cache
- C1 time: web object's request time in child 1 cache
- C2 time: web object's request time in child 2 cache
- P del: delete web object from parent cache
- C1 del: delete web object from child 1 cache
- C2 del: delete web object from child 2 cache
- P save: save web object in parent cache
- C1 save: save web object in child 1t cache
- C2 save: save web object in child 1 cache

4.1.3 Coordination Rules

In the coordination rules six input parameters has been considered:

- P freq
- C1 freq
- C2 freq
- P time
- C1 time
- C2 time

Each input parameter has two values (low or high), so we have $6_2 = 64$ rules.

And two possible actions can perform, D (delete or save), so we have six actions:

- P del
- C1 del
- C2 del
- P save
- C1 save
- C2 save

If the input parameters have the same values, the factor Time has been considered and compared, the old web object will be removed.

Rule 1: if P freq = low and C1 freq =low and C2 freq =low and P time =low C1 time= low and C2 time =low if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 2: if P freq =low and C1 freq =low and C2 freq =low and P time =low C1 time=low and C2 time =high then P del ,C1 de, C2 save

Rule 3: if P freq =low and C1 freq = low and C2 freq =low and P time = low C1 time=high and C2 time = low then P del ,C1 save, C2 del

Rule 4: if P freq =low and C1 freq =low and C2 freq =low and P time = low C1 time= high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 5: if P freq =low and C1 freq =low and C2 freq =low and P time =high C1 time= low and C2 time = low then P save ,C1 del, C2 del

Rule 6: if P freq =low and C1 freq =low and C2 freq =low and P time =high C1 time=low and C2 time = high if (P time >C2 time) then P save ,C1 del, C2 del

Rule 7: if P freq =low and C1 freq =low and C2 freq =low and P time =high C1 time=high and C2 time = low if (P time >C1 time) then P save ,C1 del, C2 del

Rule 8: if P freq =low and C1 freq =low and C2 freq =low and P time =high C1 time=high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 9: if P freq = low and C1 freq =low and C2 freq =high and P time =low C1 time= low and C2 time =low then P del ,C1 del, C2 save

Rule 10: if P freq =low and C1 freq =low and C2 freq =high and P time =low C1 time=low and C2 time =high then P del ,C1 de, C2 save

Rule 11: if P freq =low and C1 freq =low and C2 freq =high and P time = low C1 time=high and C2 time =low if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 12: if P freq =low and C1 freq =low and C2 freq =high and P time = low C1 time= high and C2 time = high then P del ,C1 del, C2 save

Rule 13: if P freq =low and C1 freq =low and C2 freq =high and P time =high C1 time= low and C2 time = low if (P time >C2 time) then P save ,C1 del, C2 del

Rule 14: if P freq =low and C1 freq =low and C2 freq =high and P time =high C1 time = low and C2 time = high then P del ,C1 del, C2 save

Rule 15: if P freq =low and C1 freq =low and C2 freq =high and P time =high C1 time=high and C2 time = low if (P time >C1 time) then P save ,C1 del, C2 del

Rule 16: if P freq =low and C1 freq =low and C2 freq =high and P time =high C1 time=high and C2 time = high then P del,C1 del, C2 save

Rule 17: if P freq = low and C1 freq = high and C2 freq =low and P time =low C1 time= low and C2 time =low then P del, C1save, C2 del

Rule 18: if P freq =low and C1 freq =high and C2 freq =low and P time =low C1 time=low and C2 time =high if (C1 time >C2 time) then P del, C1 save, C2 del

Rule 19: if P freq =low and C1 freq =high and C2 freq =low and P time = low C1 time = high and C2 time = low then P del, C1 save, C2 del

Rule 20: if P freq =low and C1 freq =high and C2 freq =low and P time = low C1 time= high and C2 time = high then P del,C1 save, C2 del

Rule 21: if P freq =low and C1 freq =high and C2 freq =low and P time =high C1 time= low and C2 time = low if (P time >C1 time) then P save, C1 del, C2 del

Rule 22: if P freq =low and C1 freq = high and C2 freq =low and P time =high C1 time=low and C2 time = high if (P time >C2 time) then P save,C1 del, C2 del

Rule 23: if P freq =low and C1 freq =high and C2 freq =low and P time =high C1 time=high and C2 time = low then P del, C1 save, C2 del

Rule 24: if P freq =low and C1 freq =high and C2 freq =low and P time =high C1 time=high and C2 time = high then P del ,C1 save, C2 del

Rule 25: if P freq =low and C1 freq =high and C2 freq =high and P time =low C1 time= low and C2 time =low if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 26: if P freq =low and C1 freq =high and C2 freq =high and P time =low C1 time=low and C2 time =high then P del,C1 de, C2 save

Rule 27: if P freq =low and C1 freq =high and C2 freq =high and P time = low C1 time=high and C2 time = low then P del ,C1 save, C2 del

Rule 28: if P freq =low and C1 freq =high and C2 freq =high and P time = low C1 time= high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del else P del ,C1 del, C2 save

Rule 29: if P freq =low and C1 freq =high and C2 freq =high and P time =high C1 time= low and C2 time = low if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 30: if P freq =low and C1 freq =high and C2 freq =high and P time =high C1 time=low and C2 time = high then P del ,C1 del, C2 save

Rule 31: if P freq =low and C1 freq =high and C2 freq =high and P time =high C1 time=high and C2 time = low then P del,C1 save, C2 del

Rule 32: if P freq =low and C1 freq =high and C2 freq =high and P time =high C1 time=high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 33: if P freq = high and C1 freq =low and C2 freq =low and P time =low C1 time= low and C2 time =low then P save ,C1 del, C2 del

Rule 34:if P freq =high and C1 freq =low and C2 freq =low and P time =low C1 time=low and C2 time =high if (P time >C2 time) then P save ,C1 del, C2 del else P del ,C1 del, C2 save

Rule 35: if P freq =high and C1 freq =low and C2 freq =low and P time = low C1 time=high and C2 time = low if (P time >C1 time) then P save ,C1 del, C2 del

Rule 36: if P freq =high and C1 freq =low and C2 freq =low and P time = low C1 time= high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 37: if P freq =high and C1 freq =low and C2 freq =low and P time =high C1 time= low and C2 time = low then P save ,C1 del, C2 del

Rule 38: if P freq =high and C1 freq =low and C2 freq =low and P time =high C1 time=low and C2 time = high then P save ,C1 del, C2 del

Rule 39: if P freq =high and C1 freq =low and C2 freq =low and P time =high C1 time=high and C2 time = low then P save ,C1 del, C2 del

Rule 40: if P freq =high and C1 freq =low and C2 freq =low and P time =high C1 time=high and C2 time = high then P save,C1 del, C2 del

Rule 41: if P freq =high and C1 freq =low and C2 freq =high and P time =low C1 time= low and C2 time =low if (P time >C2 time) then P save ,C1 del, C2 del

Rule 42: if P freq =high and C1 freq =low and C2 freq =high and P time =low C1 time=low and C2 time =high then P del ,C1 de, C2 save

Rule 43: if P freq =high and C1 freq =low and C2 freq =high and P time = low C1 time=high and C2 time = low if (P time >C2 time) then P save ,C1 del, C2 del else

Rule 44: if P freq =high and C1 freq =low and C2 freq =high and P time = low C1 time= high and C2 time = high then P del ,C1 del, C2 save

Rule 45: if P freq =high and C1 freq =low and C2 freq =high and P time =high C1 time= low and C2 time = low then P save ,C1 del, C2 del

Rule 46: if P freq =high and C1 freq =low and C2 freq =high and P time =high C1 time=low and C2 time = high if (P time >C2 time) then P save ,C1 del, C2 del

Rule 47: if P freq =high and C1 freq =low and C2 freq =high and P time =high C1 time=high and C2 time = low then P save ,C1 del, C2 del

Rule 48: if P freq =high and C1 freq =low and C2 freq =high and P time =high C1 time=high and C2 time = high if (P time >C2 time) then P save ,C1 del, C2 del

Rule 49: if P freq = high and C1 freq =high and C2 freq =low and P time =low C1 time= low and C2 time =low if (P time >C1 time) then P save ,C1 del, C2 del

Rule 50: if P freq =high and C1 freq =high and C2 freq =low and P time =low C1 time=low and C2 time =high if (P time >C1 time) then P save ,C1 del, C2 del

Rule 51: if P freq =high and C1 freq =high and C2 freq =low and P time = low C1 time=high and C2 time = low then P del ,C1 save, C2 del

Rule 52: if P freq =high and C1 freq =high and C2 freq =low and P time = low C1 time= high and C2 time = high then P del ,C1 save, C2 del

Rule 53: if P freq =high and C1 freq =high and C2 freq =low and P time =high C1 time= low and C2 time = low then P save ,C1 del, C2 del

Rule 54: if P freq =high and C1 freq =high and C2 freq =low and P time =high C1 time=low and C2 time = high then P save,C1 del, C2 del

Rule 55: if P freq =high and C1 freq =high and C2 freq =low and P time =high C1 time=high and C2 time = low if (P time >C1 time) then P save ,C1 del, C2 del

Rule 56: if P freq =high and C1 freq =high and C2 freq =low and P time =high C1 time=high and C2 time = high if (P time >C1 time) then P save ,C1 del, C2 del

Rule 57: if P freq = high and C1 freq =high and C2 freq =high and P time =low C1 time= low and C2 time =low if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 58: if P freq =high and C1 freq =high and C2 freq =high and P time =low C1 time=low and C2 time =high then P del ,C1 de, C2 save

Rule 59: if P freq =high and C1 freq =high and C2 freq =high and P time = low C1 time=high and C2 time = low then P del ,C1 save, C2 del

Rule 60: if P freq =high and C1 freq =high and C2 freq =high and P time = low C1 time= high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 61: if P freq =high and C1 freq =high and C2 freq =high and P time =high C1 time= low and C2 time = low if (C1time >C2 time) then P del ,C1 save, C2 del

Rule 62: if P freq =high and C1 freq =high and C2 freq =high and P time =high C1 time=low and C2 time = high if (P time >C2 time) then P save ,C1 del, C2 del

Rule 63: if P freq =high and C1 freq =high and C2 freq =high and P time =high C1 time=high and C2 time = low if (P time >C1 time) then P save ,C1 del, C2 del

Rule 64: if P freq =high and C1 freq =high and C2 freq =high and P time =high C1 time=high and C2 time = high if (C1time >C2 time) then P del ,C1 save, C2 del

4.2 Q-Learning Analysis

Q-Learning is a Reinforcement Learning method for solving sequential decision problems, where the utility of actions depends on a sequence of decisions and there exists uncertainty about the dynamics of the environment.

When the cache cleaner agent pass throw the similar state, it can directly takes its optimal action that lead immediately to the goal. The goal of the training is to find the sequential order of actions which maximizes the sum of the future reinforcements, thus leading to the shortest path from start to finish.

4.2.1 Q-Learning Terminology

The terminology in Q-Learning includes the terms "state" and "action". We'll call each node a "state", and the agent's movement from one node to another will be an "action", a "state" is depicted as a node, while "action" is represented by the arrows. We'll associate a reward value to each node (i.e. link between nodes). The nodes that lead immediately to the goal which have high caching priority have an instant reward of 100.Others have zero reward. Each arrow contains an instant reward value.

In the proposed model a dynamic cache environment has been consider when observed the behavior of parent and child caches. Parameters and its values were represented in a graph, each rule (state) as a node.

Actions in this setting correspond to the decision of whether to delete or save the web object.

Figure 4.2.2 explains parent cache cleaner agent's states, three parameters have been considered:

-Web object's size (Size).

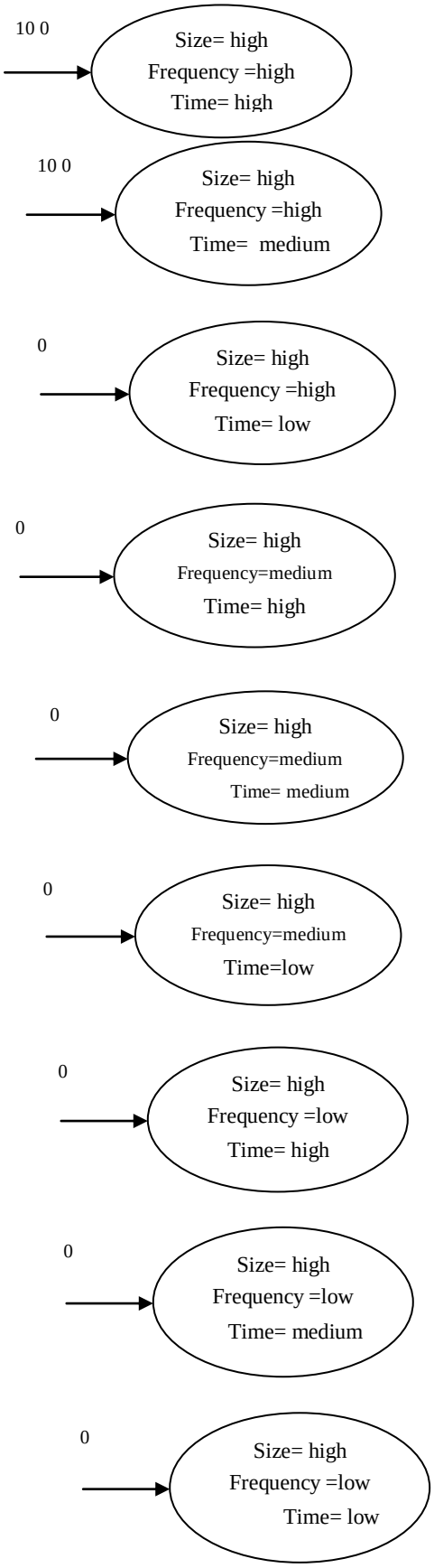
-Request time (Time).

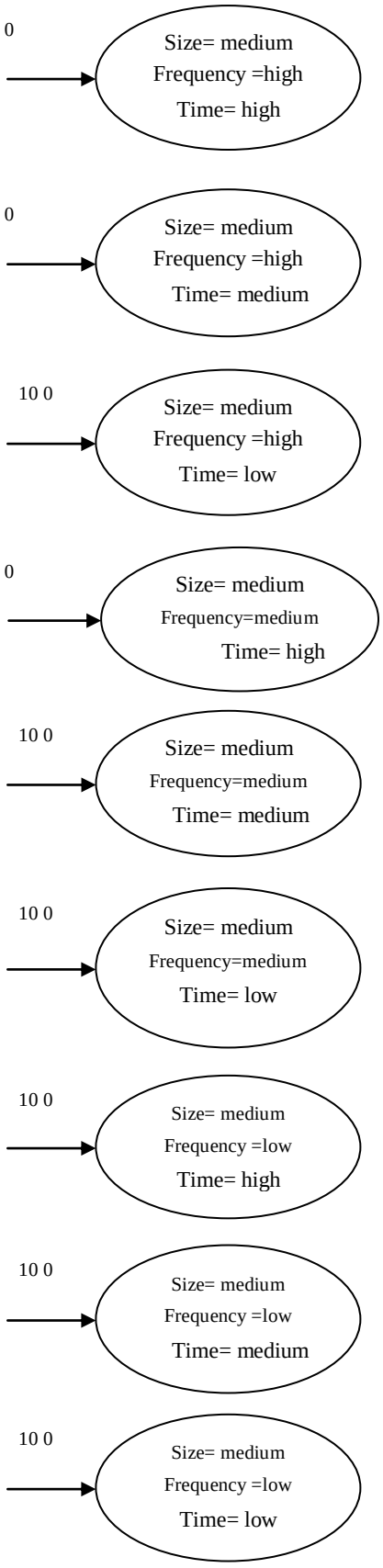
-Web Object's frequency (Frequency).

Each parameter has one of three values (high, medium, low) .So we have 3_3=27 states that represent as 27 nodes.

Parent agent has two actions delete or save the web object. Actions represented in figure 4.2.2 as arrows, a reward value has been associated to each node. The nodes that lead immediately to the goal which have high caching priority have an instant reward of 100.Others have zero reward.

4.2.2 Parent's States




0
Size= medium
Frequency =high
Time= high
0
Size= medium
Frequency =high
Time= medium
10 0
Size= medium
Frequency =high
Time= low
0
Size= medium
Frequency=medium
Time= high
10 0
Size= medium
Frequency=medium
Time= medium
10 0
Size= medium
Frequency=medium
Time= low
10 0
Size= medium
Frequency =low
Time= high
10 0
Size= medium
Frequency =low
Time= medium
10 0
Size= medium
Frequency =low
Time= low

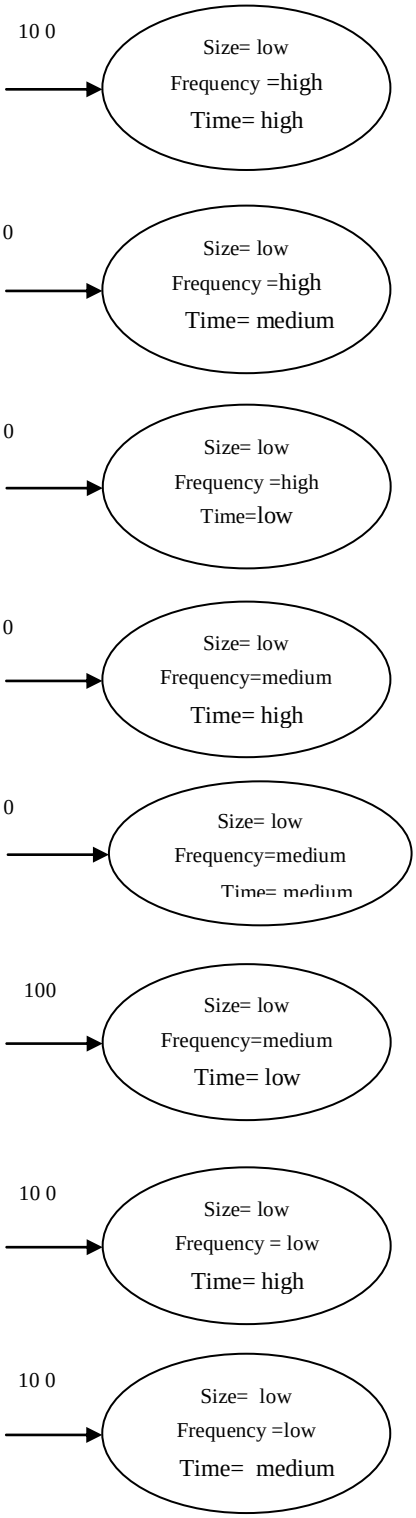


Figure 4.1:Parent' States

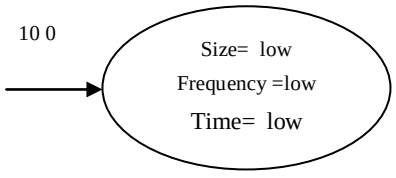


Figure 4.2.3 shows child cleaner agent's states, two parameters have been considered:

-Request time (Time).

-Web Object's frequency (Frequency).

Each parameter has one of three values (high, medium, low) .So we have 2_3=9 states that represent as 9 nodes.

Child agent has two actions delete or save the web object. Actions represented in figure 4.2.3 as arrows, a reward value has been associated to each node. The nodes that lead immediately to the goal which have high caching priority have an instant reward of 100.Others have zero reward.

4.2.3 Child's States

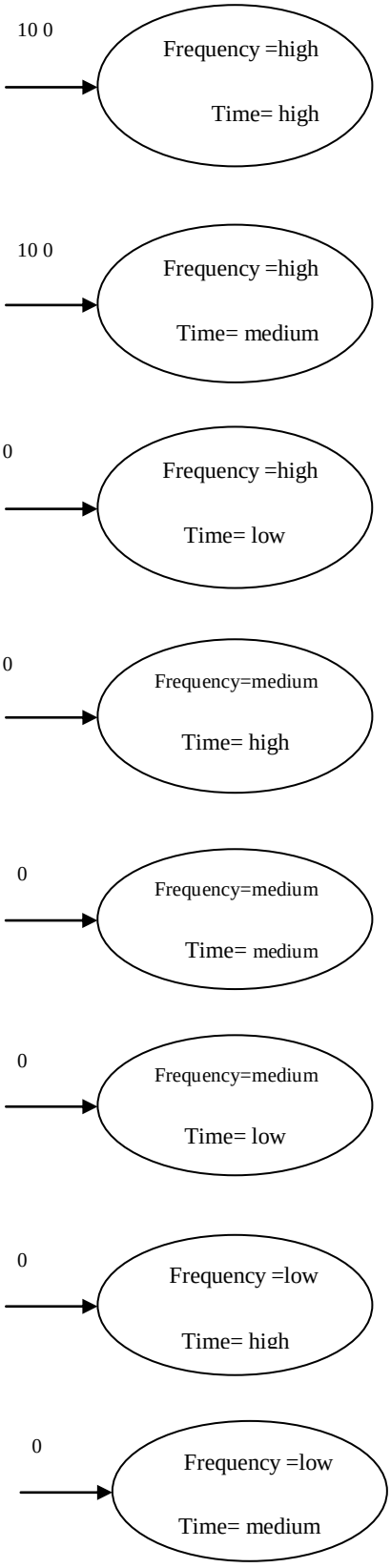


Figure 4.2: Child's State

CHAPTER 5: IMPLEMENTATION

MODEL IMPLEMENTATION

The proposed model has been implemented using JADE tool. At the beginning the monitoring, cache cleaner and coordinator agents' behaviors have been determined. In the second phase fuzzy logic has been used to model data that generated by WebTraff simulator. It generates two files webworkload.dat that represents the user request and the cachesim.dat that dispatches into parent and child caches. In the third phase the agents communication through messages to interact with each others.

5.1 Implementation using Fuzzy Logic

This section describes in detail how fuzzy logic can be utilized in cache performance enhancement. To design an adaptive fuzzy based system we consider three input variables web object size, frequency, and the time. We compute the output variable called the cache cleanup priority as shown in figure 5.1.

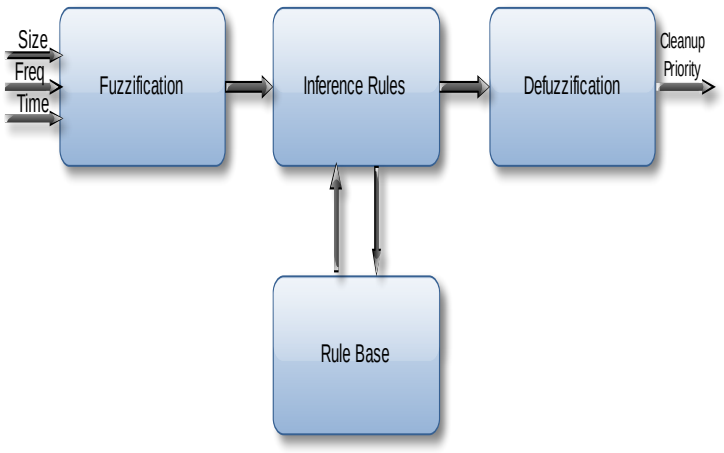


Figure 5.1: Cache cleaner fuzzy based system

5.1.1 Fuzzification Process

In the fuzzification step the Monitoring Agent reads the three input parameters from cache and workload files and converts the input parameters to linguistic values such as "high", "medium" or "low". The output parameter (CP) also has three linguistic terms "high", " medium" or "low", the cleaner agent delete the web object with high clean up priority.

Five samples of web proxy workload have been generated with different cache size using WebTraff simulator.

Maximum value and minimum value to parameters size, time and frequency have been calculated to each generated sampleanddivided to three ranges high, medium and low.

Tables 5.1, 5.2 and 5.3 explain the input fuzzy parameters and table 5.4 explains output fuzzy parameter to first sample (sampe1). Table 5.5 explains input fuzzy parameter's units.

Table5.1: Input Fuzzy Parameter (Size)

Web Object Size (S)	Fuzzy Values
0 – 50000	Low
50000 – 100000	Medium
100000 – 150000	High

Table 5.2: Input Fuzzy Parameter (Frequency)

Web Object Frequency (F)	Fuzzy Values
0 – 3	Low
3 – 7	Medium
7 – 10	High

Table 5.3: Input Fuzzy Parameter (Time)

Web Object request Time (T)	Fuzzy Values
0 – 70000	Low
70000 – 140000	Medium
140000 – 210000	High

Table 5.4: Out Fuzzy Parameters

cleanup Priority(CP)	Fuzzy values
0.0 - 0.3	Low
0.3 - 0.7	Medium
0.7 - 1.0	High

Table 5.5 : Input Fuzzy Parameter's Units

Input Fuzzy Parameters	Unit
Web object Size(S)	Byte
Web object Frequency(F)	Integer number
Time(T)	Second

5.1.2 Inference Rules Process

Mamdani-style[50] was used to perform the fuzzy inference process.

- Parent Cache Cleaner Agent's Rules

We use three input variables:

- Web object Size
- Web object Frequency
- Request Time

and one output variable:

- Cleanup_Priority.

- Each input parameter has one of three values (high, medium or low) .So we have 3_3= 27 rules.

Rule 1: **If** Size = high **and** Frequency = high **and** Time = high

then Cleanup_Priority =high

Rule 2: **If** Size = high **and** Frequency = high **and** Time = medium **then** Cleanup_Priority =high

Rule 3: **If** Size = high **and** Frequency = high **and** Time = low **then** Cleanup_Priority =medium

Rule 4: **If** Size = high **and** Frequency = medium **and** Time = high **then** Cleanup_Priority =low

Rule 5: **If** Size = high **and** Frequency = medium **and** Time = medium **then** Cleanup_Priority =low

Rule 6: **If** Size = high **and** Frequency = medium **and** Time = low **then** Cleanup_Priority =medium

Rule 7: **If** Size = high **and** Frequency = low **and** Time = high t**hen** Cleanup_Priority = low

Rule 8: **If** Size = high **and** Frequency = low **and** Time = medium **then** Cleanup_Priority = low

Rule 9: **If** Size = high **and** Frequency = low **and** Time = low **then** Cleanup_Priority =medium

Rule 10: **If** Size = medium **and** Frequency = high **and** Time = high **then** Cleanup_Priority =low

Rule 11: **If** Size = medium **and** Frequency = high **and** Time = medium **then** Cleanup_Priority =low

Rule 12: **If** Size = medium **and** Frequency = high **and** Time = low **then** Cleanup_Priority =high

Rule 13: **If** Size = medium **and** Frequency = medium **and** Time = high **then** Cleanup_Priority =medium

Rule 14: **If** Size=medium **and** Frequency = medium **and** Time= medium **then** Cleanup_Priority =high

Rule 15: **If** Size = medium **and** Frequency = medium **and** Time = low **then** Cleanup_Priority =medium

Rule 16: **If** Size = medium **and** Frequency = low **and** Time = high **then** Cleanup_Priority =high

Rule 17: **If** Size = medium **and** Frequency = low **and** Time = medium **then** Cleanup_Priority =high

Rule 18: **If** Size = medium **and** Frequency = low **and** Time = low **then** Cleanup_Priority =high

Rule 19: **If** Size = low **and** Frequency = high **and** Time = high **then** Cleanup_Priority =high

Rule 20: **If** Size = low **and** Frequency = high **and** Time = medium **then** Cleanup_Priority =high

Rule 21: **If** Size = low **and** Frequency = high **and** Time =low

then Cleanup_Priority =medium

Rule 22: **If** Size = low **and** Frequency = medium **and** Time = high **then** Cleanup_Priority =low

Rule 23: **If** Size = low **and** Frequency = medium **and** Time = medium **then** Cleanup_Priority =low

Rule 24: **If** Size = low **and** Frequency = medium **and** Time = low **then** Cleanup_Priority =low

Rule 25: **If** Size = low**and** Frequency = low **and** Time = high **then** Cleanup_Priority =medium

Rule 26: **If** Size = low **and** Frequency = low **and** Time = medium **then** CP_Linguistic=low

Rule 27: **If** Size = low **and** Frequency =low **and** Time = low **then** CP_Linguistic=low

The pervious rules have been summarized in table 5.6

Table 5.6: Parent's Inference rules

#	Size	Frequency	Time	Cleanup_Prority
1	high	high	High	High
2	high	high	Mediu	High
3	high	high	low	Medium
4	high	medium	high	Low
5	high	medium	mediu	Low
6	high	medium	low	Medium
7	high	low	high	Low
8	high	low	mediu	Low
9	high	low	low	Medium
10	medium	high	high	Low
11	medium	high	mediu	Low
12	medium	high	low	High
13	medium	medium	high	Medium
14	medium	medium	mediu	High
15	medium	medium	low	Medium
16	medium	low	high	High
17	medium	low	mediu	High
18	medium	low	low	High
19	low	high	high	High
20	low	high	mediu	High
21	low	high	low	Medium
22	low	medium	high	Low
23	low	medium	mediu	Low
24	low	medium	low	Low
25	low	low	high	Medium
26	low	low	mediu	Low
27	low	low	low	Low

- **Child Cache Cleaner Agent's Rules**

We use two input variables

- Web object Frequency
- Request Time

and one output variable:

- Cleanup_Priority.
- Each input parameter has one of three values (high, medium or low) .So we have 2_3 = 9 rules.

Rule1: **If** Frequency = high **and** Time = high **then** Cleanup_Priority=high

Rule2: **If** Frequency = high **and** Time = medium **then** Cleanup_Priority = high

Rule3: **If** Frequency = high **and** Time = low **then** Cleanup_Priority = medium

Rule4: **If** Frequency = medium **and** Time=high **then** Cleanup_Priority = low

Rule5: **If** Frequency = medium **and** Time = medium **then** Cleanup_Priority = high

Rule 6: **If** Frequency = medium **and** Time = low **then** Cleanup_Priority =medium

Rule 7: **If** Frequency = low **and** Time = high **then** Cleanup_Priority = high

Rule 8: **If** Frequency = low **and** Time = medium **then** Cleanup_Priority =low

Rule 9: **If** Frequency = low **and** Time = low **then** Cleanup_Priority = high

The pervious rules have been summarized in table 5.7

Table 5.7: Child's Inference rules

#	Frequency	Time	Cleanup_Prority
1	high	high	high
2	high	medium	high
3	high	low	medium
4	medium	high	low
5	medium	medium	high
6	medium	low	medium
7	low	high	high
8	low	medium	low
9	low	low	high

5.1.3 Defuzzification and Membership Function

Three membership functions are defined that showed the scale of the web object size, frequency and time: low, medium and high. The output cleanup priority membership function has also defined to show the scale of the web object's cleanup priority.

Triangular membership function has been used that specified by three parameters {a, b, c} as shown in equation 5.1 and figure 5.2:

$$\mathbf{Triangle}(\mathbf{x}; \mathbf{a}, \mathbf{b}, \mathbf{c}) = \begin{cases} \mathbf{0} & \mathbf{x} < \mathbf{a} \\ \dfrac{\mathbf{x} - \mathbf{a}}{\mathbf{b} - \mathbf{a}} & \mathbf{a} \leq \mathbf{x} \leq \mathbf{b} \\ \dfrac{\mathbf{c} - \mathbf{x}}{\mathbf{c} - \mathbf{b}} & \mathbf{b} \leq \mathbf{x} \leq \mathbf{c} \\ \mathbf{0} & \mathbf{x} > \mathbf{c} \end{cases} \quad 5.1$$

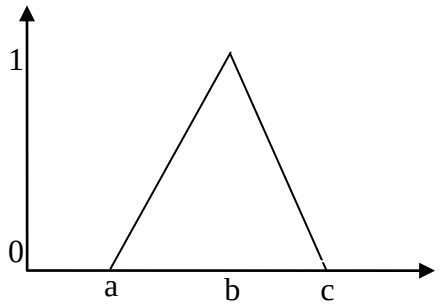


Figure 5.2: Triangular membership function

Figures 5.3, 5.4 and 5.5 explain web object size, frequency and time membership functions and figure 5.6 explains the cleanup priority membership function to the first generated sample (sampe1).

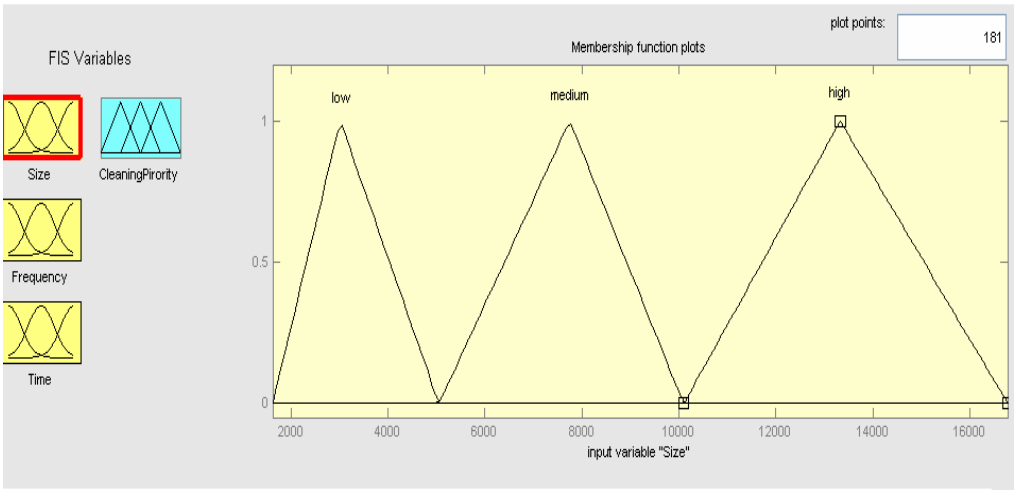


Figure 5.3: Web object size membership function

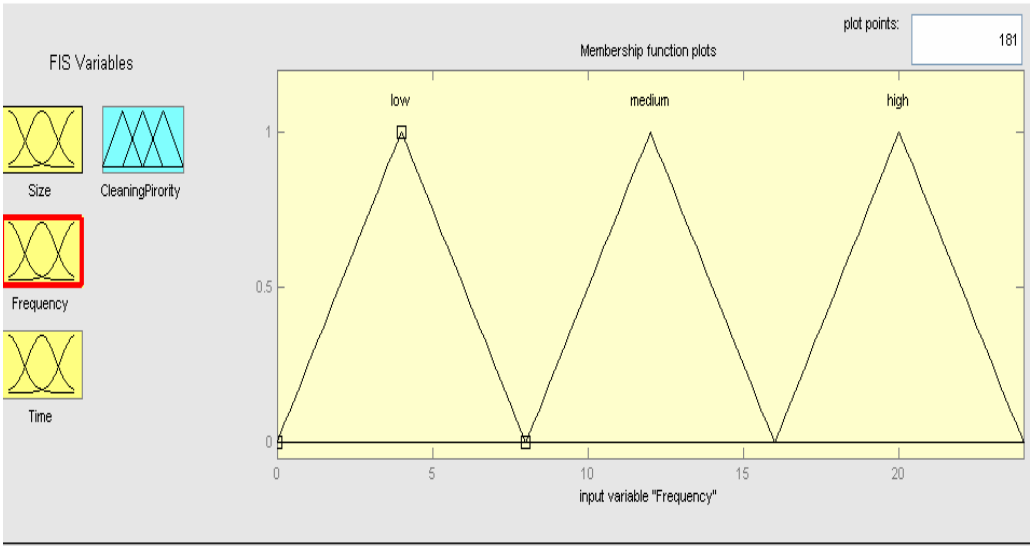


Figure 5.4: Web object frequency membership function

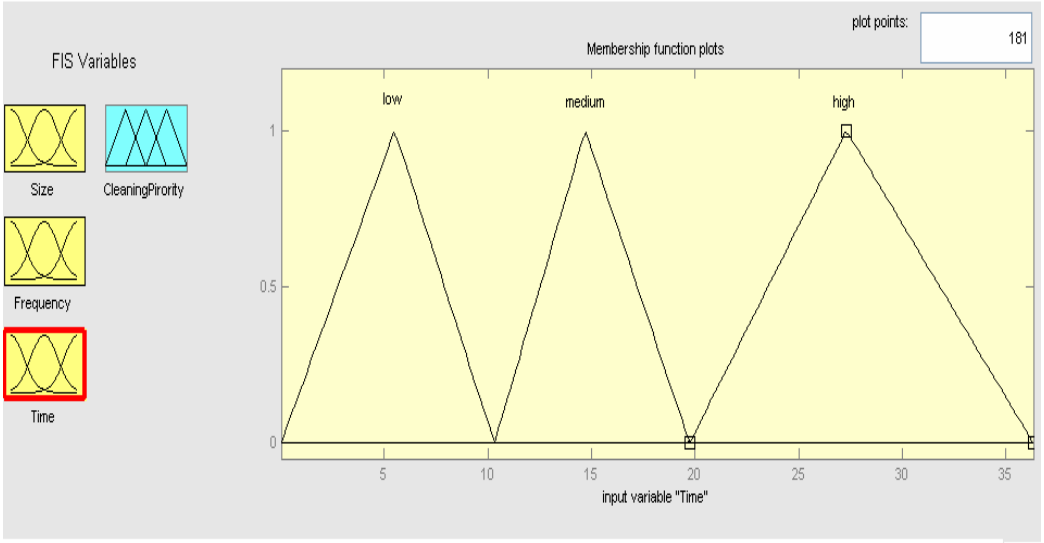


Figure 5.5 : Web object time membership function

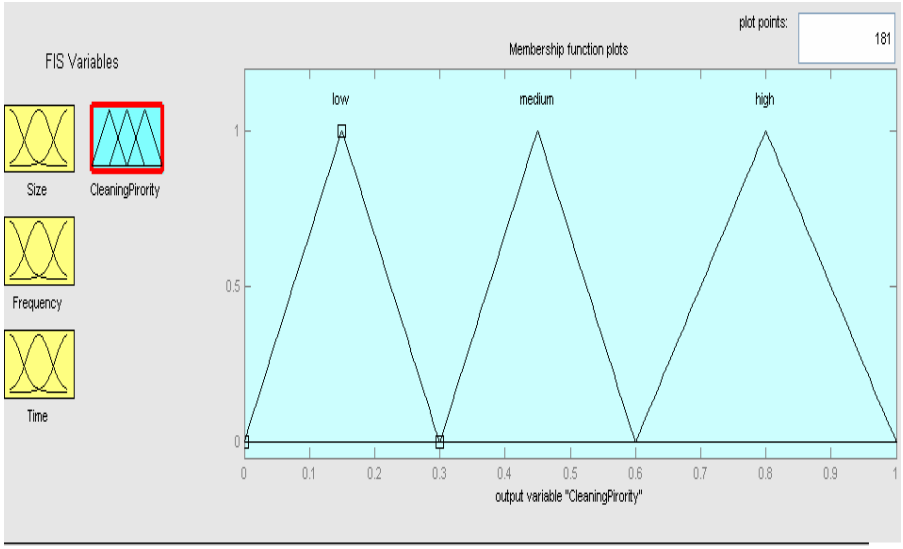


Figure 5.6 : cleanup priority membership function

5.2 Coordination Rules Implementation

5.2.1 Coordination Rules Simplified

To simplify previous coordination rules that are mentioned in Chapter Four, we rewrite them as a truth table as shown in table 5.8, to apply the Coordination truth table we assume that:

- We have two values low and high, we represent low value with 0 and high value with 1.
- When the parent frequency (P_Freq) equal 0 that means the web object in the parent cache with low frequency.
- When the parent frequency (P_Freq) equal 1 that means the web object in the parent cache with high frequency.
- When the child 1 frequency (C1_Freq) equal 0 that means the web object in the child 1 cache with low frequency.
- When the child 1 frequency (C1_Freq) equal 1 that means the web object in the child 1 cache with high frequency.
- When the child 2 frequency (C2_Freq) equal 0 that means the web object in the child 2 cache with low frequency.
- When the child 2 frequency (C2_Freq) equal 1 that means the web object in the child 2 cache with high frequency.
- When the parent time (P_time) equal 0 that means the web object in the parent cache is old object.
- When the parent time (P_time) equal 1 that means the web object in the parent cache is a new object.
- When the child 1 time (C1_Time) equal 0 that means the web object in the child 1 cache is old object.
- When the child 1 time (C1_Time) equal 1 that means the web object in the child 1 cache is a new.
- When the child 2 time (C2_Time) equal 0 that means the web object in the child 2 cache is old object.
- When the child 2 time (C2_Time) equal 1 that means the web object in the child 2 cache is a new.
- When (P_save) equal 1 that means the web object kept in the parent cache.

- When (P_save) equal 0 that means the web object deleted from the parent cache.
- When (C1_save) equal 1 that means the web object kept in child 1 cache.
- When (C1_save) equal 0 that means the web object deleted from child 1 cache.
- When (C2_save) equal 1 that means the web object kept in child 2 cache.
- When (C2_save) equal 0 that means the web object deleted from child 2 cache.

Table 5.7: Coordination Truth Table

#	P _Freq	C1 _Freq	C2_Freq	P_Time	C1_Time	C2_Time	P_ save	C1_ save	C2_ save
0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0	0	1
2	0	0	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0	1	0
4	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0
6	0	0	0	1	1	0	1	0	0
7	0	0	0	1	1	1	0	1	0
8	0	0	1	0	0	0	0	0	1
9	0	0	1	0	0	1	0	0	1
10	0	0	1	0	1	0	0	1	0
11	0	0	1	0	1	1	0	0	1
12	0	0	1	1	0	0	1	0	0
13	0	0	1	1	0	1	0	0	1
14	0	0	1	1	1	0	1	0	0
15	0	0	1	1	1	1	0	0	1
16	0	1	0	0	0	0	0	1	0
17	0	1	0	0	0	1	0	1	0
18	0	1	0	0	1	0	0	1	0
19	0	1	0	0	1	1	0	1	0
20	0	1	0	1	0	0	1	0	0
21	0	1	0	1	0	1	1	0	0
22	0	1	0	1	1	0	0	1	0
23	0	1	0	1	1	1	0	1	0
24	0	1	1	0	0	0	0	1	0
25	0	1	1	0	0	1	0	0	1
26	0	1	1	0	1	0	0	1	0
27	0	1	1	0	1	1	0	1	0
28	0	1	1	1	0	0	0	1	0
29	0	1	1	1	0	1	0	0	1
30	0	1	1	1	1	0	0	1	0
31	0	1	1	1	1	1	0	1	0

32	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
33	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
34	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
35	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
36	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
37	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
38	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**
39	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
40	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**
41	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
42	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**
43	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**
44	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
45	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
46	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**
47	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
48	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
49	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
50	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**
51	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
52	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
53	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
54	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**
55	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
56	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
57	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
58	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**
59	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**
60	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
61	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
62	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**0**
63	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**

To simplify truth table we use the sum of min term format:

5.2.2 Sum of Minterms

P_save =
m4+m5+m6+m12+m14+m20+m21+m32+m33+m34+m36+m37+m38+m39+m40+m42+m44+m45+m46+m47+m48+m49+m52+m53+m54+m55+m61+m62

C1_save =
m0+m2+m3+m7+m10+m16+m17+m18+m19+m22+m23+m24+m26+m27
+m28+m30+m31+m35+m50+m51+m56+m58+m59+m60+m63

C2_save = m1+m8+m9+m11+m13+m15+m25+m29+m41+m43+m57

P_save =
∑m(4,5,6,12,14,20,21,32,33,34,36,37,38,39,40,42,44,45,46,47,48,49,52,53,5
4,55,61,62)

C1_save =
∑m(0,2,3,7,10,16,17,18,19,22,23,24,26,27,28,30,31,35,50,51,56,58,59,60,6
3)

C2_save = ∑m(1,8,9,11,13,15,25,29,41,43,57)

5.3 Q-Learning Implementation

Cache cleaner agent repeatedly interacts with the environment and tries to estimate the optimal Q*(s, α). In particular, the agent starts with random estimates Q (s,α) for each state action pair, and then begins exploring the environment. During exploration it receives tuples in the form (s, R, α, s`) where s is the current state, R is the current reward, α is an action taken in state s, and s` is the resulting state after executing α. From each tuple, the agent updates its action value estimates as shown in equation 5.1:

$$\mathbf{Q(s, \alpha) := (1- \lambda)\, Q(s, \alpha) + \lambda[R + \gamma \max Q(s`, \alpha`)]} \quad 5.1$$

Where λ (0,1) is a learning rate that controls convergence.

The cache cleaner agent will explore from state to state until it reaches the goal. We'll call each exploration an episode. Each episode consists of the agent moving from the initial state to the goal state. Each time the agent arrives at the goal state, the program goes to the next episode.

The Q-Learning algorithm goes as follows:

1. Set the gamma parameter, and environment rewards in matrix R.

2. Initialize matrix Q to zero.

3. For each episode:

Select a random initial state.

Do while the goal state hasn't been reached.

Select one among all possible actions for the current state.

Using this possible action, consider going to the next state.

Get maximum Q value for this next state based on all possible actions.

Compute:

Q(state, action)=R(state, action)+gamma * Max[Q(next state, all actions)]

Set the next state as the current state.

End Do

End For

Figure 5.7: Q-Learning Algorithm

The Q-Learning algorithm shown in figure 5.7 is used by the cache cleaner agent to learn from experience. The input is the R matrix and the output is Q matrix. Each episode is equivalent to one training session. In each training session, the cleaner agent explores the environment (represented by matrix R), receives the reward (if any) until it reaches the goal state. The purpose of the training is to enhance the 'brain' of cache cleaner agent,

represented by matrix Q. More training results in a more optimized matrix Q. In this case, if the matrix Q has been enhanced, the cache cleaner agent will find the fastest route to the goal state. The Gamma parameter has a range of 0 to 1 (0 <= Gamma > 1). If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

To understand how the Q-learning algorithm works, we'll go through a few episodes step by step.

- We'll start by setting the value of the learning parameter Gamma = 0.8.
- Initialize matrix Q as a zero matrix.
- Initialize matrix R with the rewards values

R Matrix: We associate a reward value to each action, when the cache cleaner agent takes its optimal action that leads to the goal, it has an instant reward of 100. Other actions have zero reward.

The rows of matrix R represent the states of the agent, and the columns represent the possible actions.

We assume the number of states to be 27 states

State	Action	
0	0	100
1	0	100
2	0	100
3	0	100
4	0	100
5	0	100
6	0	100
7	0	100
8	0	100
9	0	100
10	0	100
11	0	100
12	0	100

- The agent read the first line on the cache that represents the first web object so the initial state will be the situation parameters of this object. The first row (state 0) of matrix R represents the first web object's situation parameter. There are two possible actions for the current state 0: to **delete** the web object or **save** it. By random selection, the agent selects to **delete** the web object.

- To use the matrix Q, the cache cleaner agent simply traces the sequence of states, from the initial state to goal state. The algorithm finds the actions with the highest reward values recorded in matrix Q for current state is shown as follows:

1. Set current state = initial state.

2. From current state, find the action with the highest Q value.

3. Set current state = next state.

4. Repeat Steps 2 and 3 until current state = goal state.

Figure 5.8: Algorithm to utilize the Q matrix

The algorithm shown in figure 5.8 will return the sequence of states from the initial state to the goal state.

The cleaner agent starts out knowing nothing, the matrix Q is initialized to zero.

- The transition rule of Q learning is a very simple formula:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

According to this formula, a value assigned to a specific element of matrix Q, is equal to the sum of the corresponding value in matrix R and the learning parameter Gamma, multiplied by the maximum value of Q for all possible actions in the next state.

Q(0, 1) = R(0, 1) + 0.8 * Max[Q(1, 0), Q(1, 1)] = 100 + 0.8 * 0 = 100

Since matrix Q is still initialized to zero, Q(1, 0), Q(1, 1), are all zero. The result of this computation for Q(0, 1) is 100 because of the instant reward

from R(1, 0).The next state 1, now becomes the current state. Cache cleaner agent's brain now contains an updated matrix Q.

Now the agent was in state 1. Also there are two possible actions for the current state 1: **delete** the web object or **save** it. By random selection, it selects to **save** the web object.

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(1, 0) = R(1, 0) + 0.8 * Max[Q(0, 0), Q(0, 1)] = 0 + 0.8 * 0 = 0

The result of this computation for Q(1, 0) is 0.

We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

The next state, 2, now becomes the current state and so on until the agent reach the goal state.

CHAPTER 6: TESTING AND RESULTS

6.1 Testing and Results

We use WebTraff simulator to generate Web proxy workloads and different cache size. The following figure explains the simulator interface.

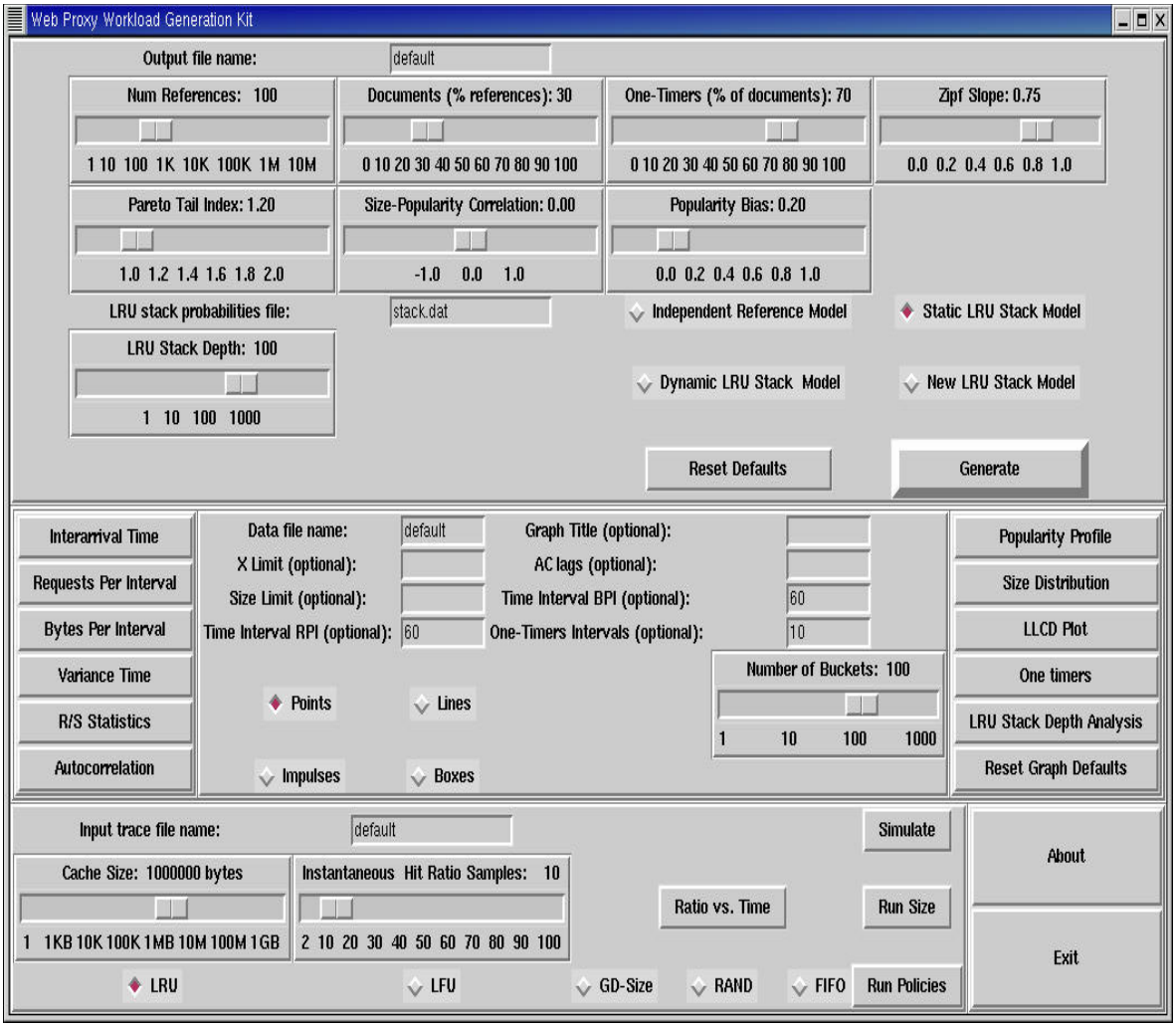


Figure 6.1:Figure Screen Shot of Graphical User Interface (GUI) for WebTraff Tool

And the following figure explains sample of workload which generated by the simulator.

Time-Stamp	Doc-Id	Size
0.01831	1	2885
0.17150	1	2885
1.37429	1	2885
4.24071	0	4241
5.54242	1	2885
7.12107	1	2885
7.61197	1	2885
8.13901	3	4245
8.14961	0	4241
8.38060	0	4241
9.10558	4	1624
9.50900	2	16782
10.12284	0	4241
10.31290	0	4241

Figure **6.2**: Sample of Web Workload Format Used in WebTraff

- We test the model using five samples ofWeb proxy workloads and different cache size start from 1 to 32768 k.
  - 1 M byte
  - 6 M byte
  - 500 M byte
  - 800 M byte
  - 1 G byte

6.2 Performance Evaluation

The standard performance metrics Hit Ratio (HR) and Byte Hit Ratio (BHR) are used to evaluate the performance of the proposed model. These can be calculated as follows:

$$\boldsymbol{HR} = \frac{\sum_{i=1}^{n} \partial i}{n} \qquad 6.1$$

$$\boldsymbol{BHR} = \frac{\sum_{i=1}^{n} b_i \partial i}{\sum_{i=1}^{n} b_i} \qquad 6.2$$

When *n*: total Number of requests

∂i: 1 if the request *i* is in the cache

∂i: 0 otherwise

bi: size in bytes

The new improvement mode's results (PCCIA) compare with traditional LRU, LFU and Size removable policies in terms of byte hit rate and hit rate.

Results have been shown in appendix.

The following figures give a comparison of PCCIA with traditional LRU, LFU and Size removable policies in terms of byte hit rate and hit rate.

The *x* coordinator represents cache size and *y* coordinator represents the terms Hit ratio and byte hit ratio.

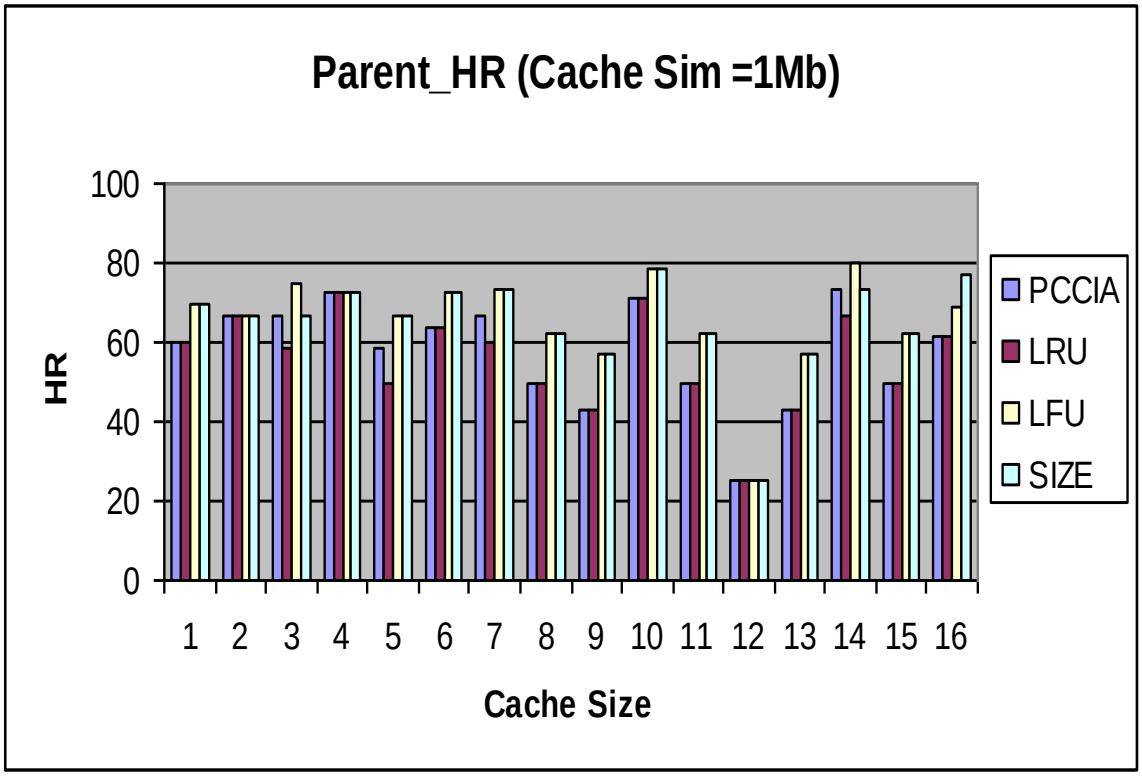


Figure 6.3: Parent HR (Cache Sim = 1Mb)

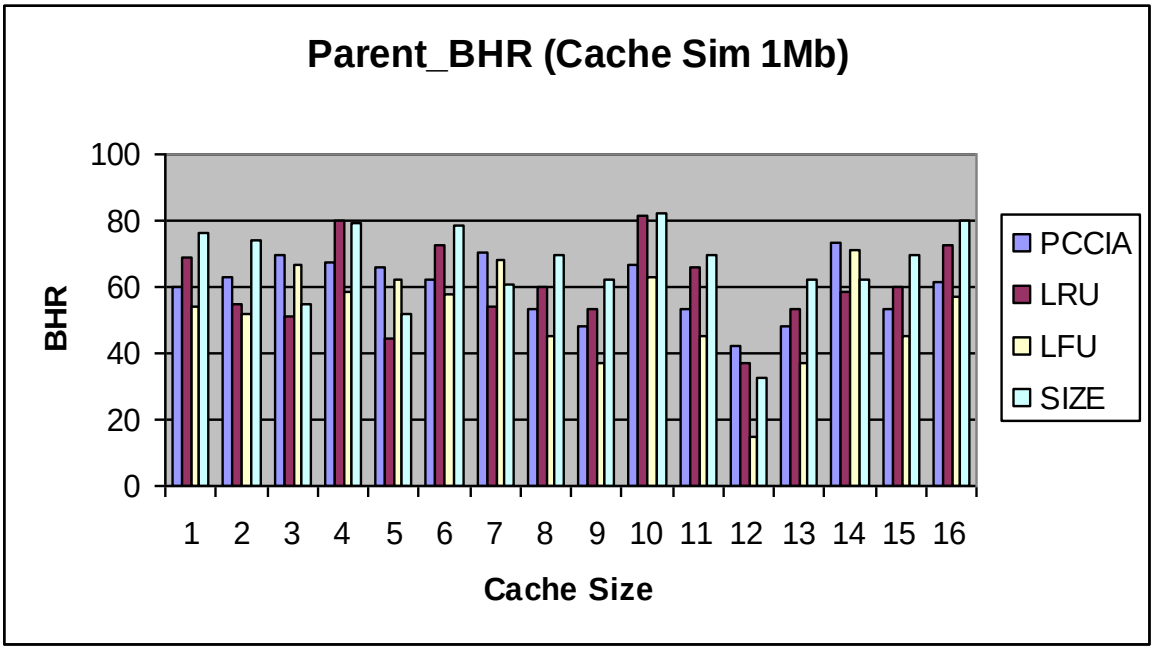


Figure 6.4: Parent BHR (Cache Sim = 1 Mb)

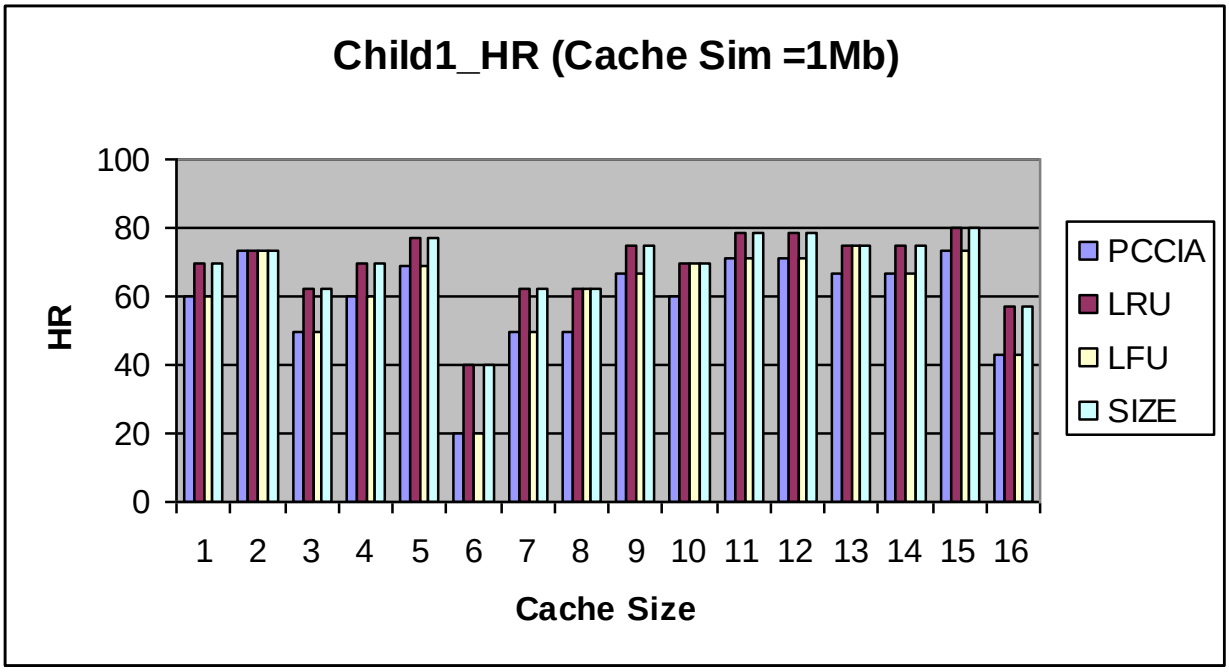


Figure 6.5: Child1 HR(Cache Sim = 1 Mb)

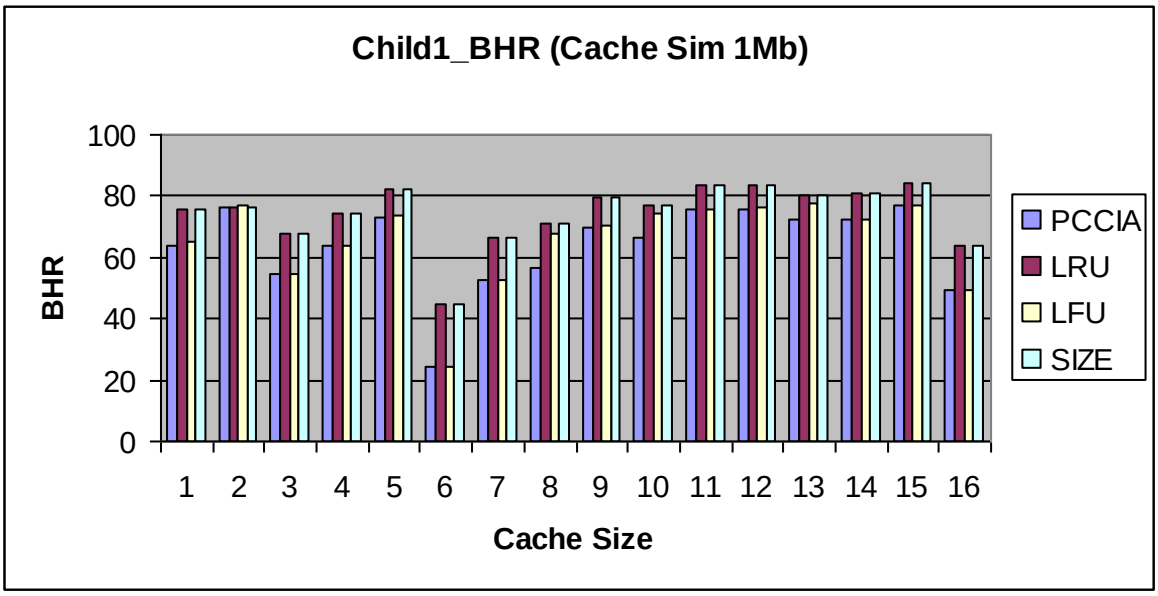


Figure 6.6: Child1 BHR (Cache Sim =1Mb)

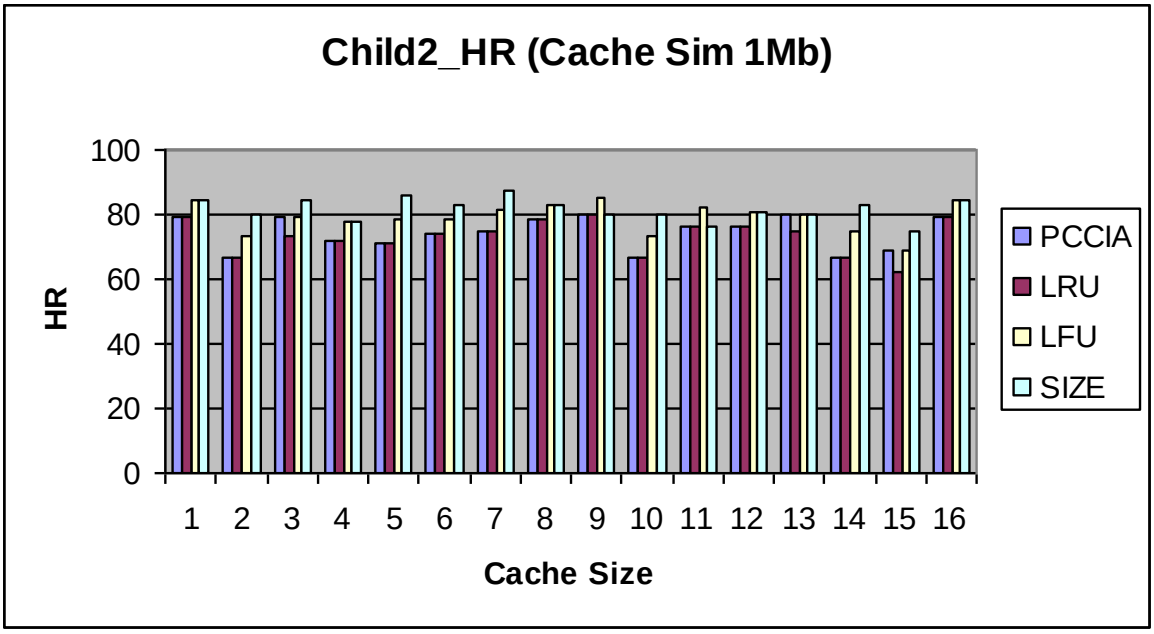


Figure 6.7: Child2 HR (Cache Sim = 1 Mb)

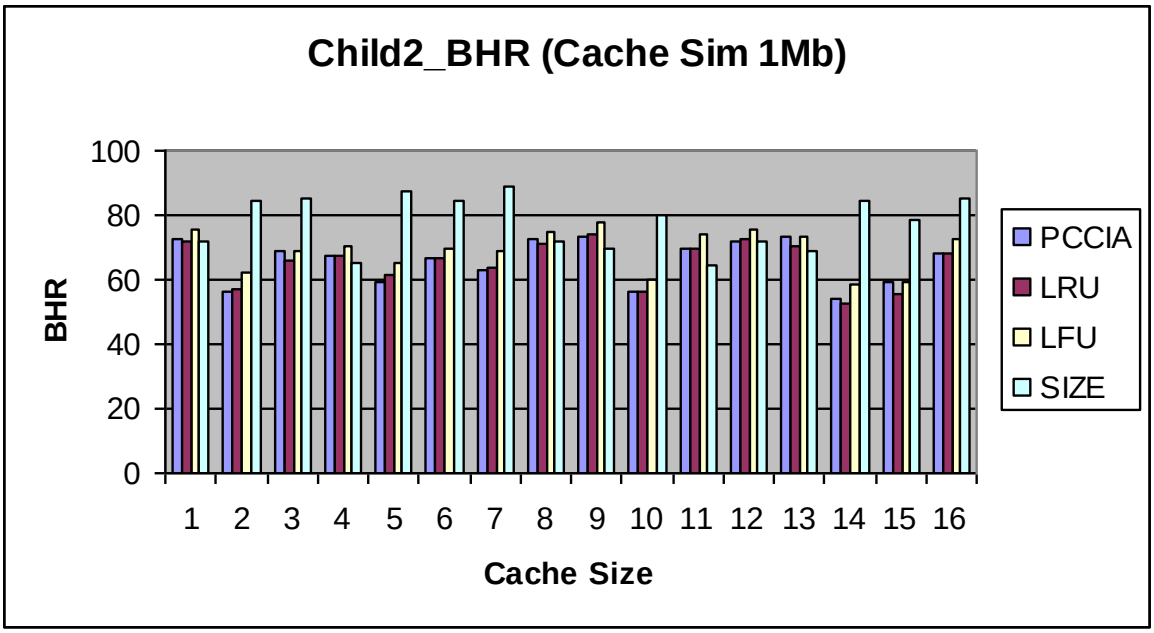


Figure 6.8: Child 2 BHR (Cache Sim =1 Mb)

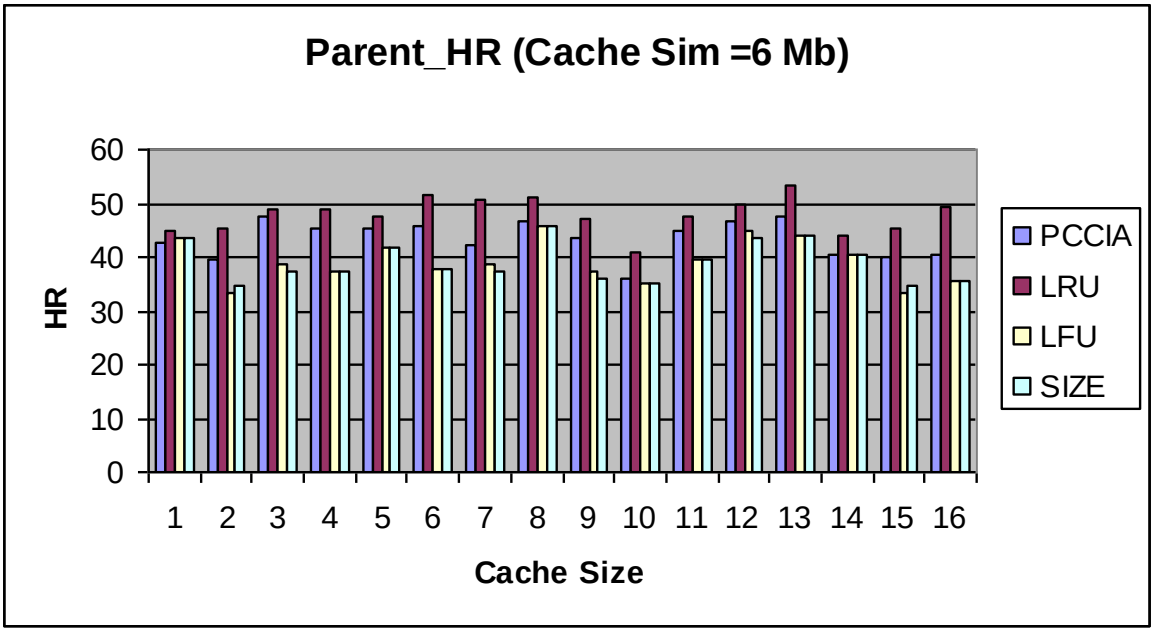


Figure 6.9: Parent HR (Cache Sim = 6 Mb)

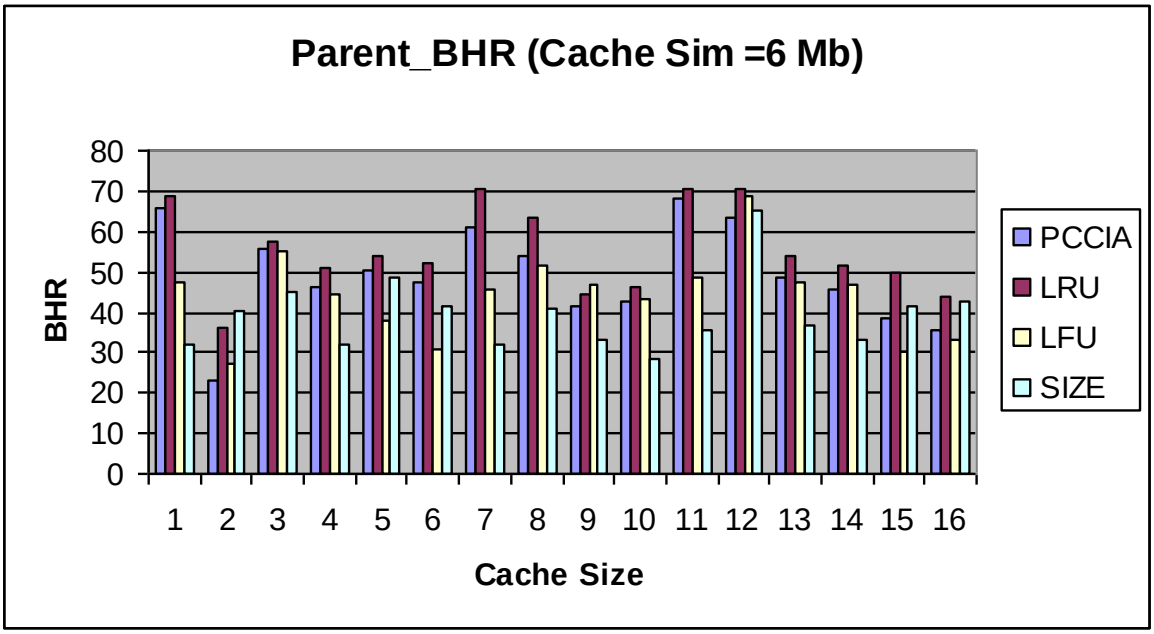


Figure 6.10: Parent BHR (Cache Sim= 6 Mb)

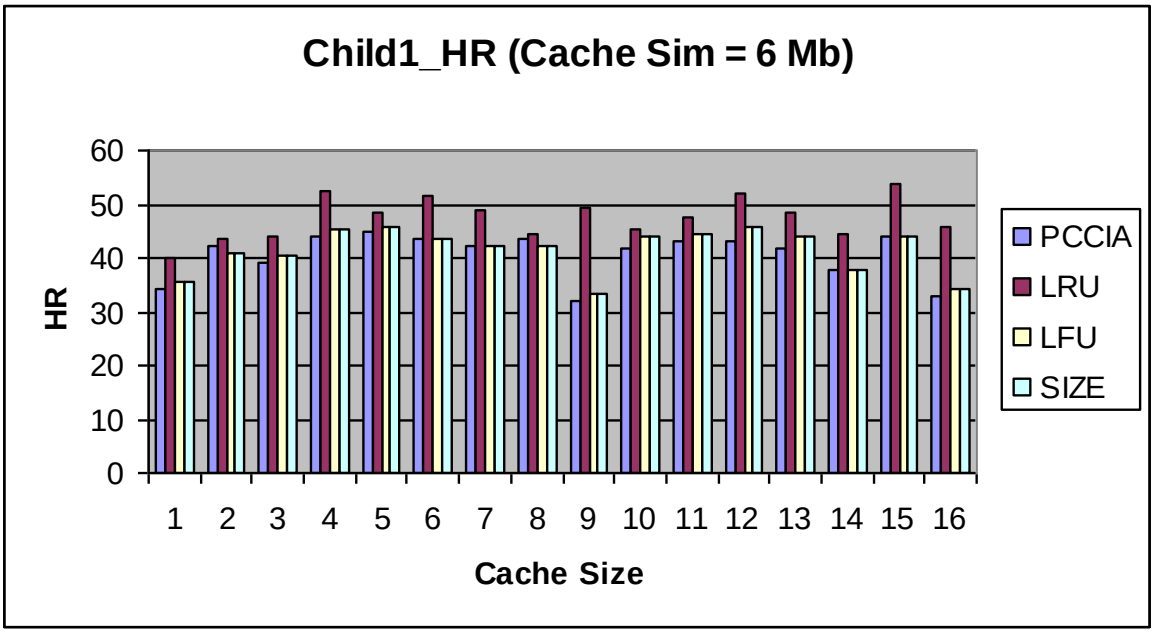


Figure 6.11: Child1 HR (Cache Sim = 6 Mb)

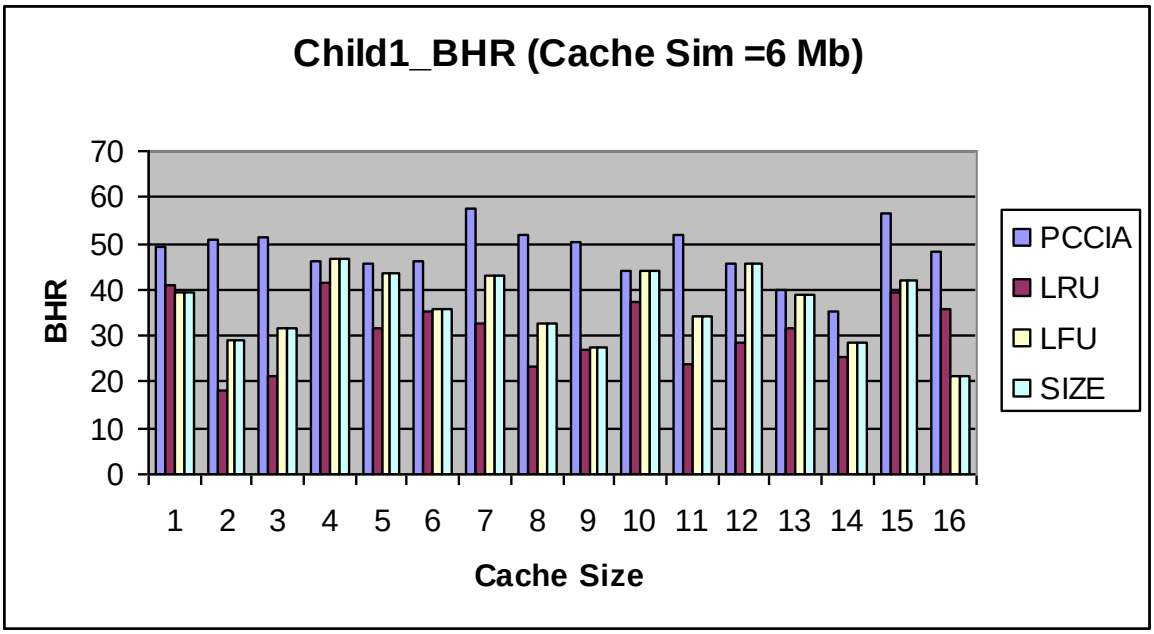


Figure 6.12: Child1 BHR (Cache Sim = 6 Mb)

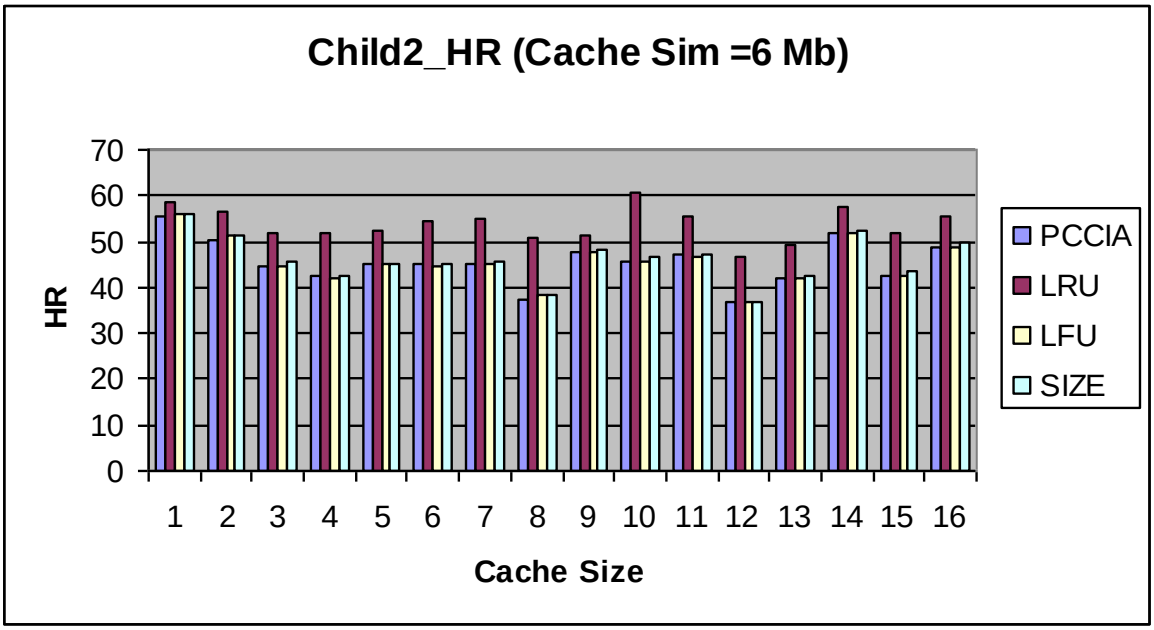


Figure 6.13: Child2 HR (Cache Sim = 6 Mb)

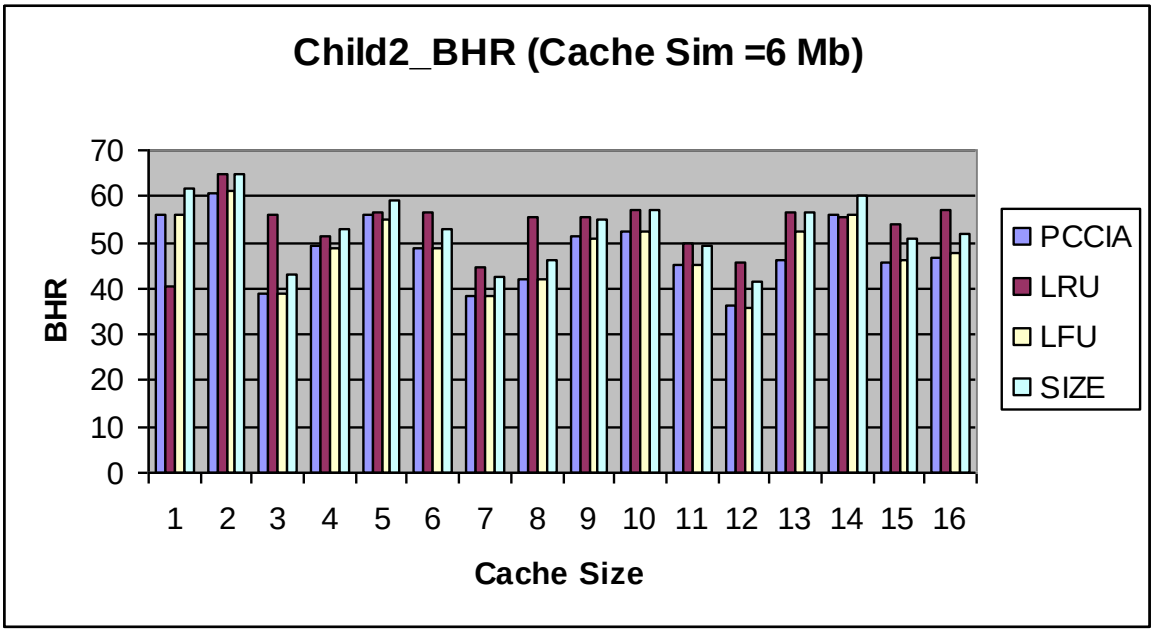


Figure 6.14: Child2 BHR (Cache Sim =6 Mb)

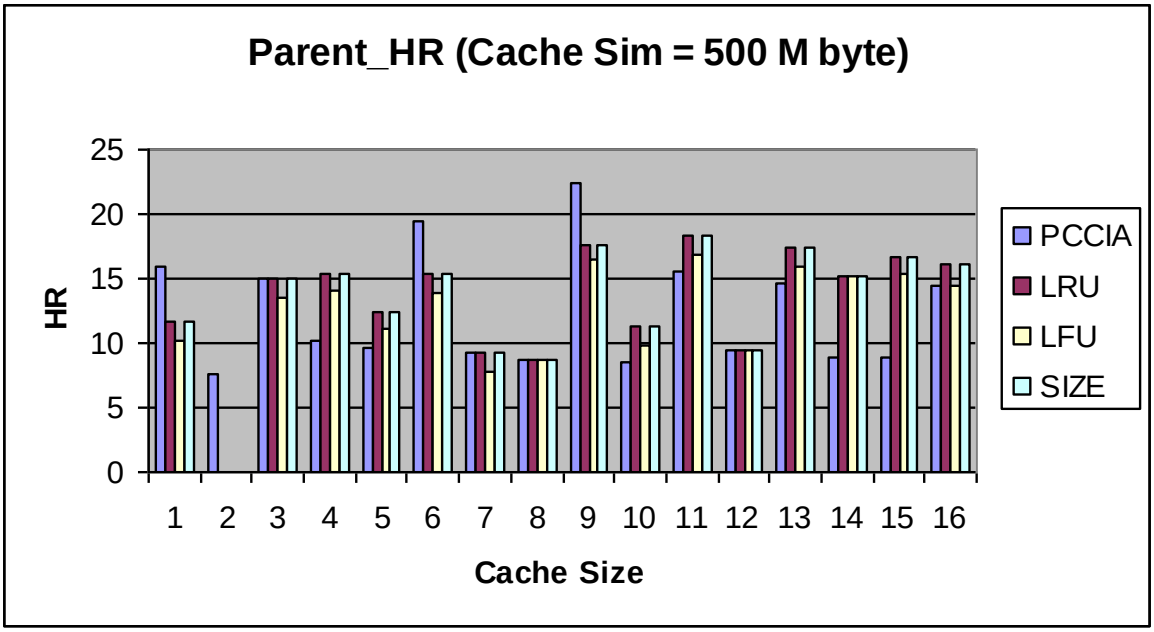


Figure 6.15: Parent HR (Cache Sim = 500 Mb)

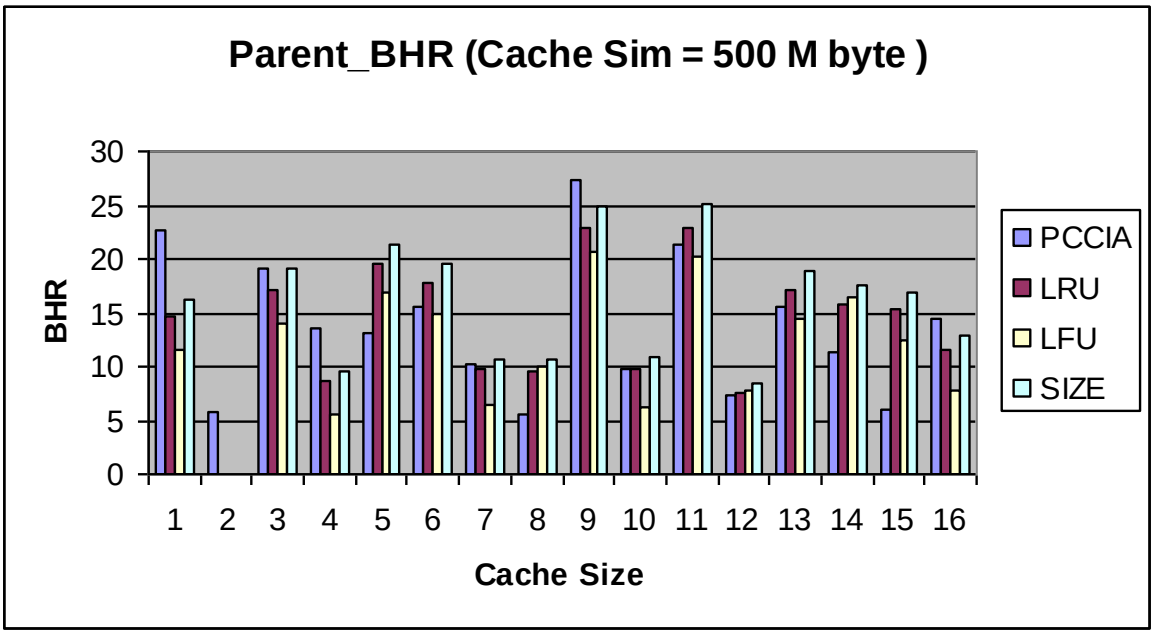


Figure 6.16: Parent BHR (Cache BHR = 500 Mb)

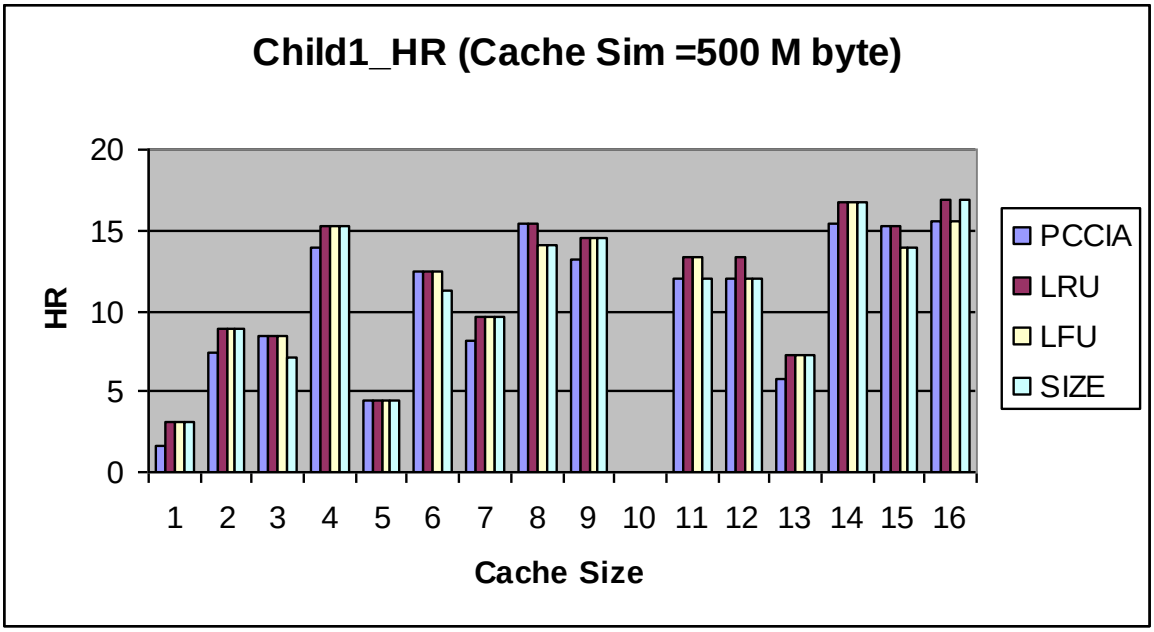


Figure 6.17: Child1 HR (Cache Sim = 500 Mb)

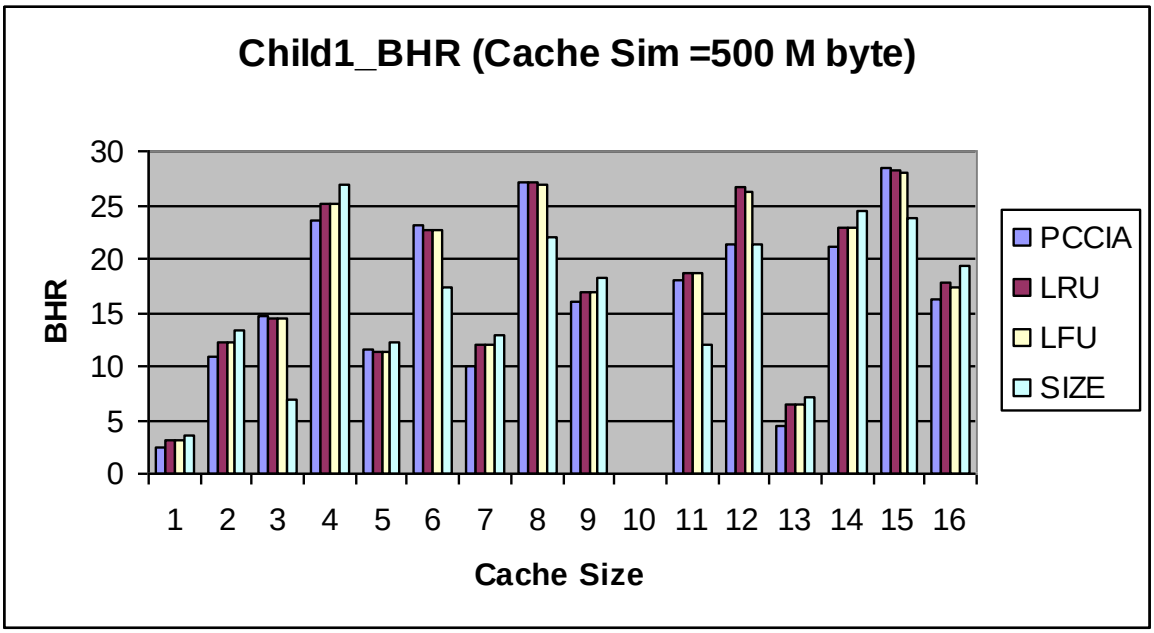


Figure 6.18: Child1 BHR (Cache Sim =500 Mb)

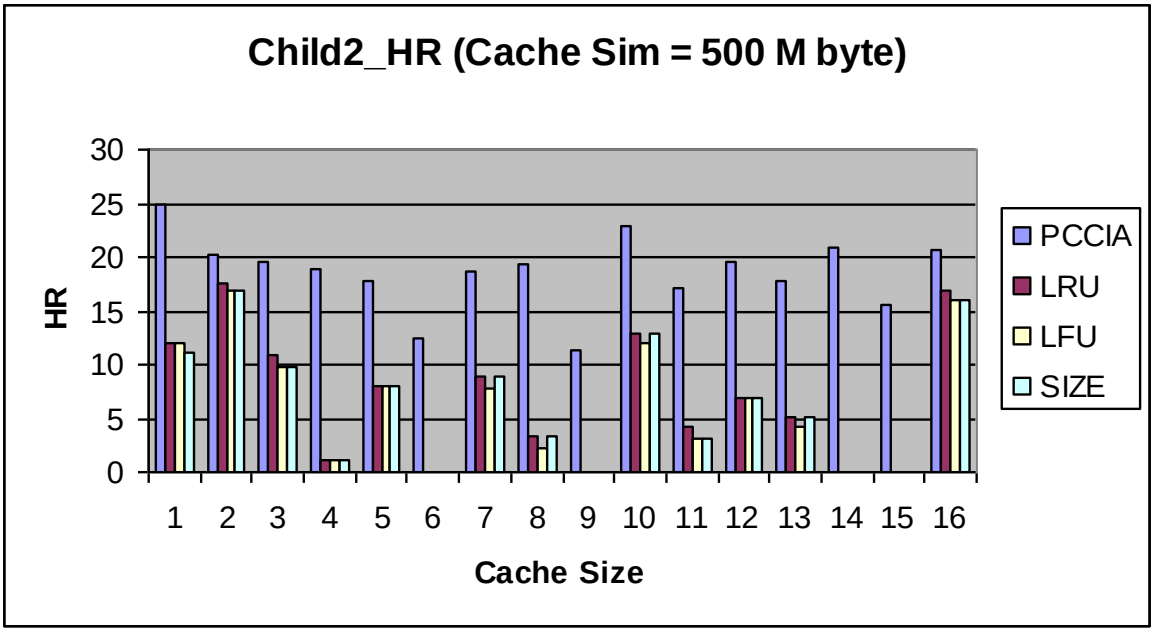


Figure 6.19: Child2 HR (Cache Sim =500 Mb)

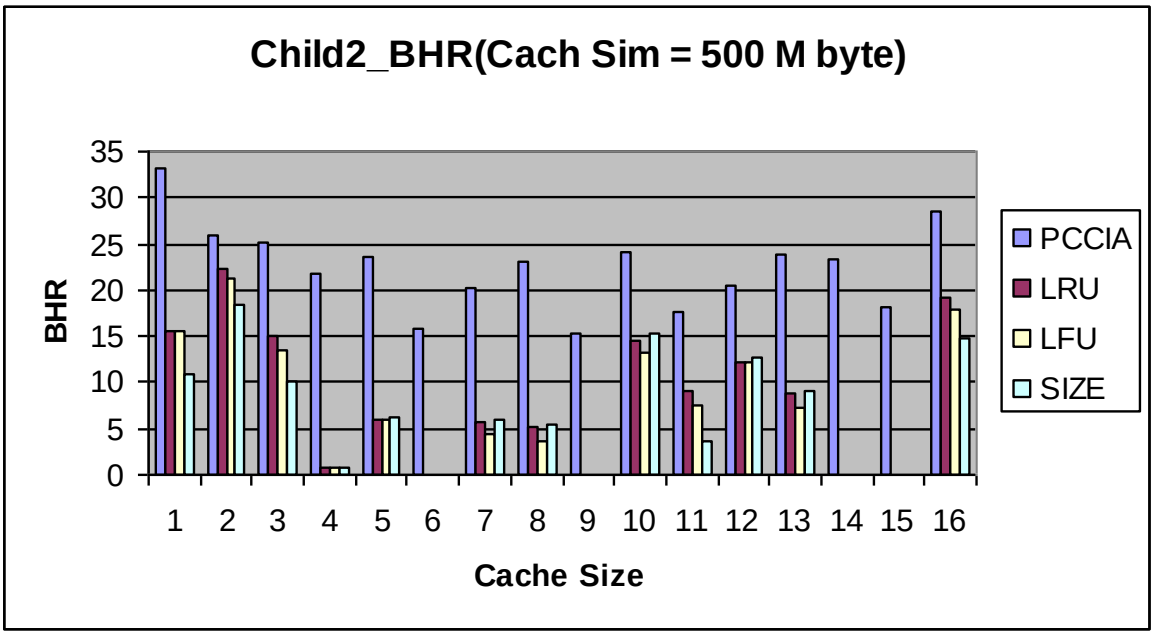


Figure 6.20: Child2 BHR (Cache Sim = 500 Mb)

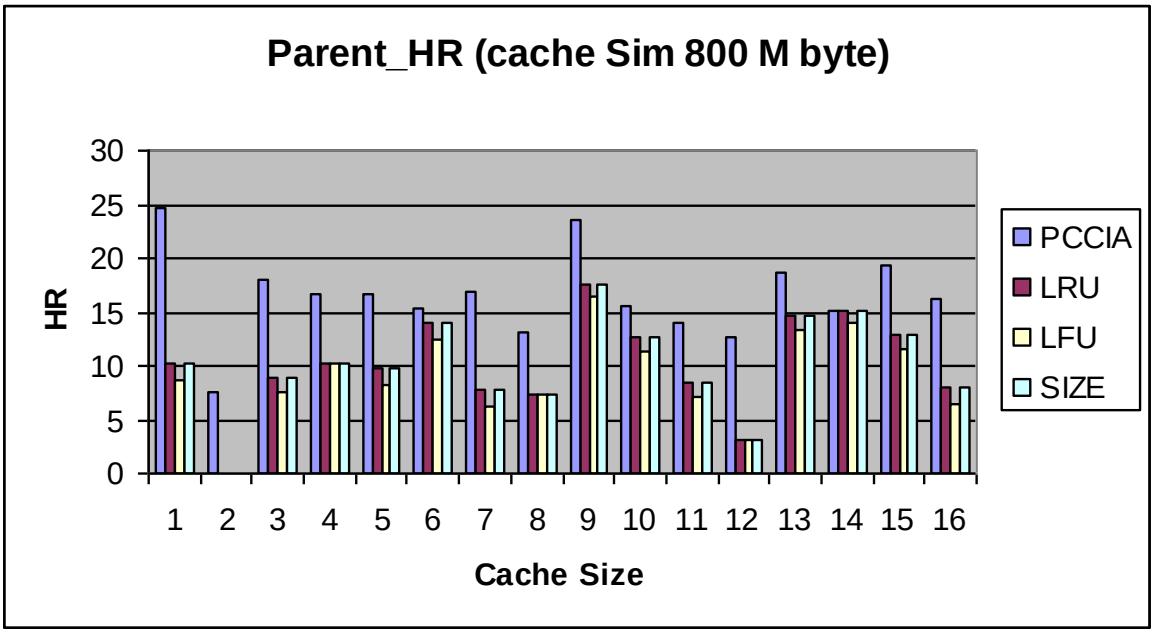


Figure 6.21: Parent HR (Cache Sim = 800 Mb)

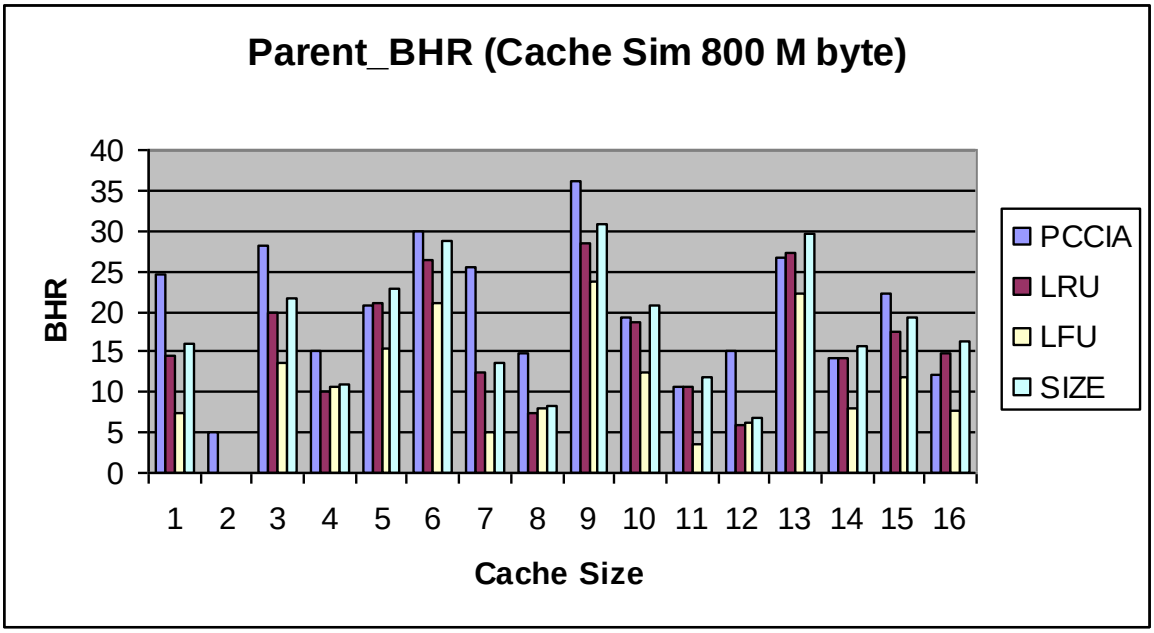


Figure 6.22: Parent BHR (Cache Sim 800 Mb)

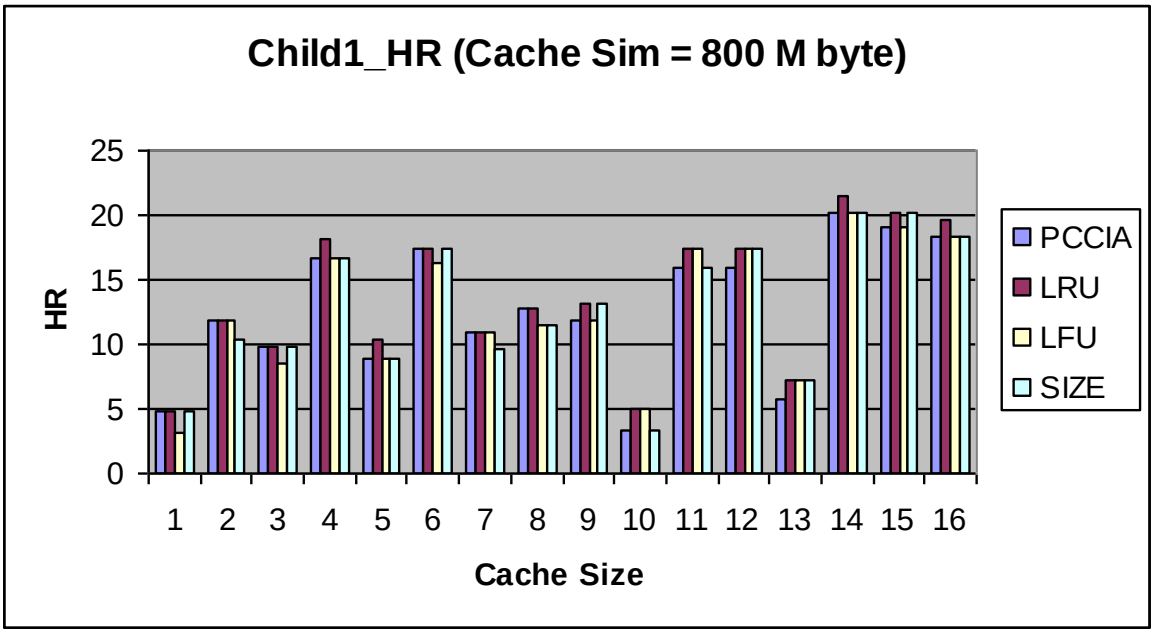


Figure 6.23: Child HR (Cache Sim = 800 Mb)

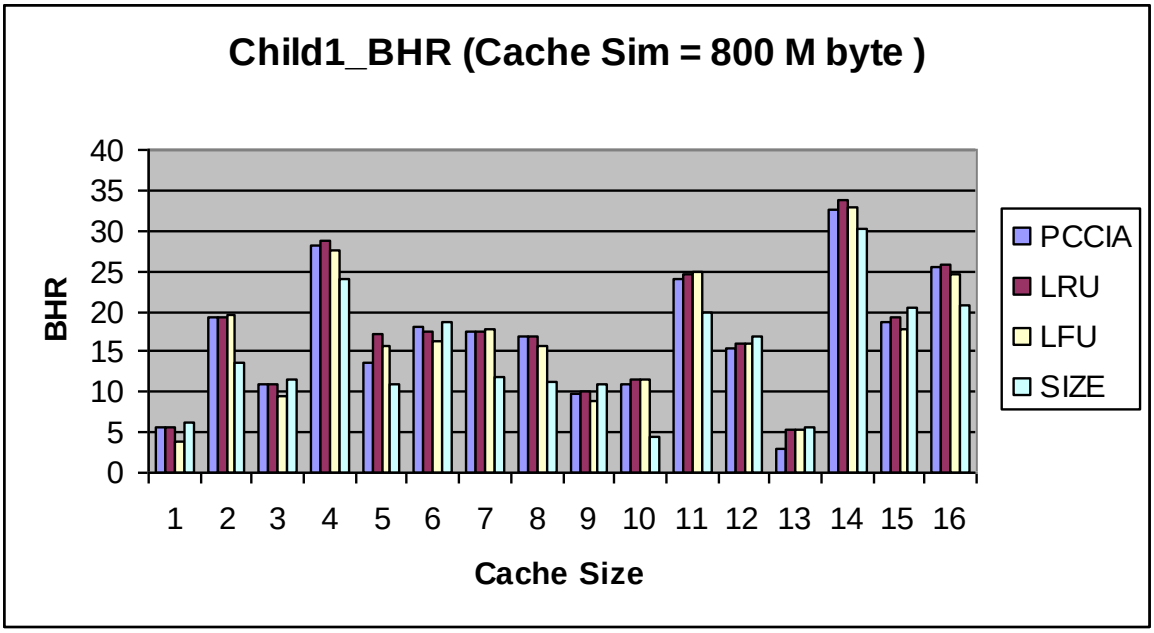


Figure 6.24: Child1 BHR (Cache Sim = 800 Mb)

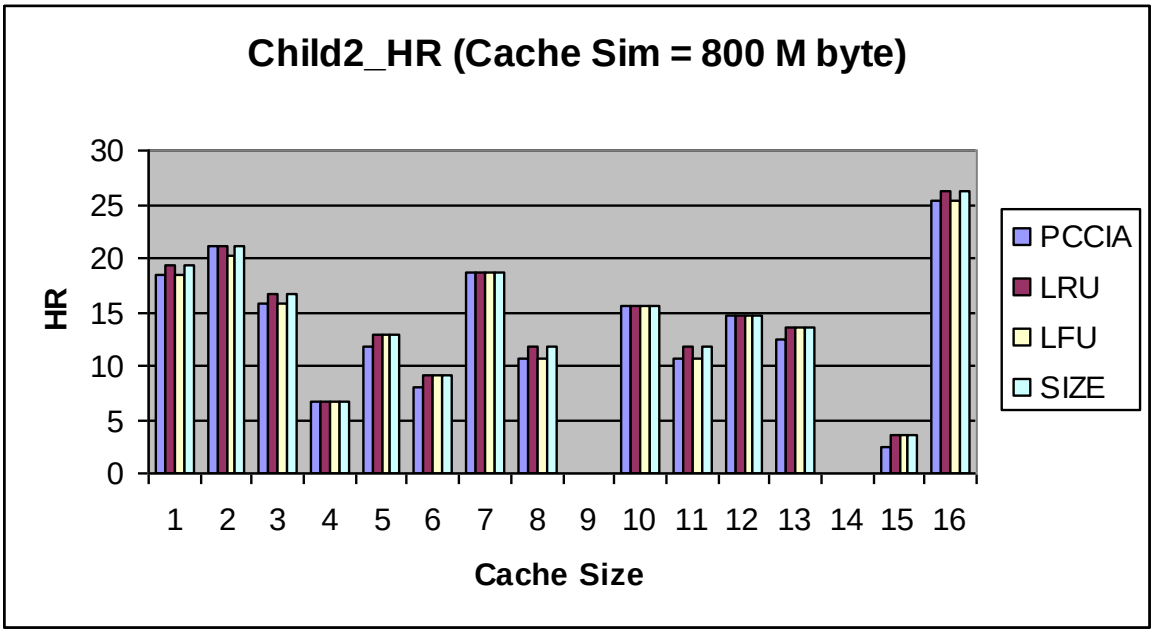


Figure 6.25: Child2 HR (Cache Sim = 800 Mb)

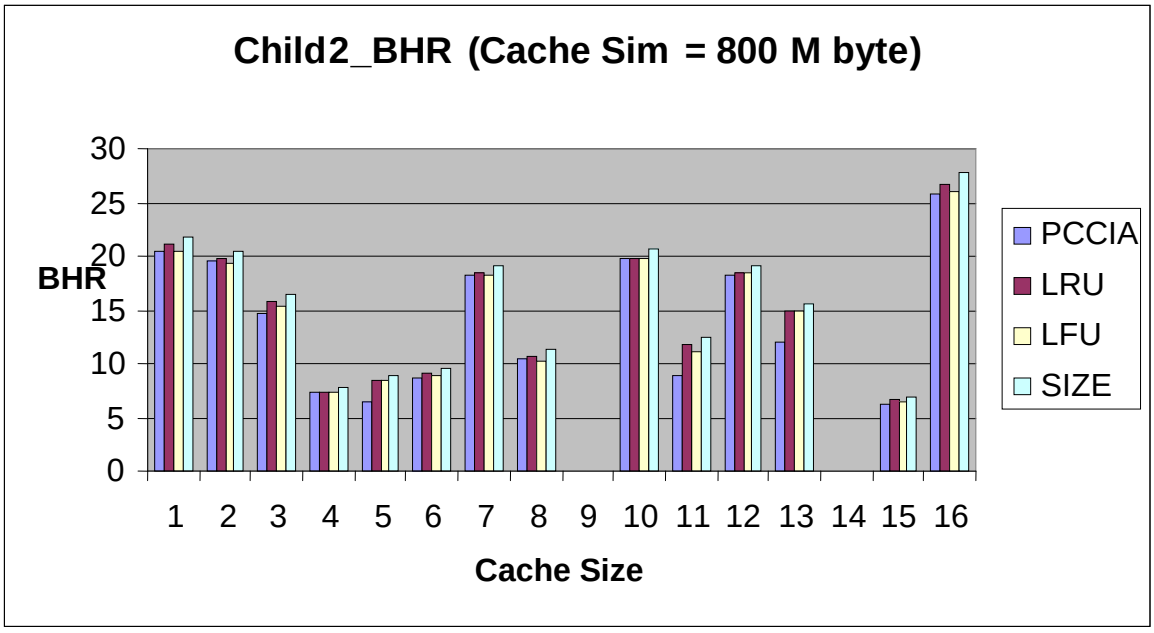


Figure 6.26: Child2 BHR (Cache Sim =800 Mb)

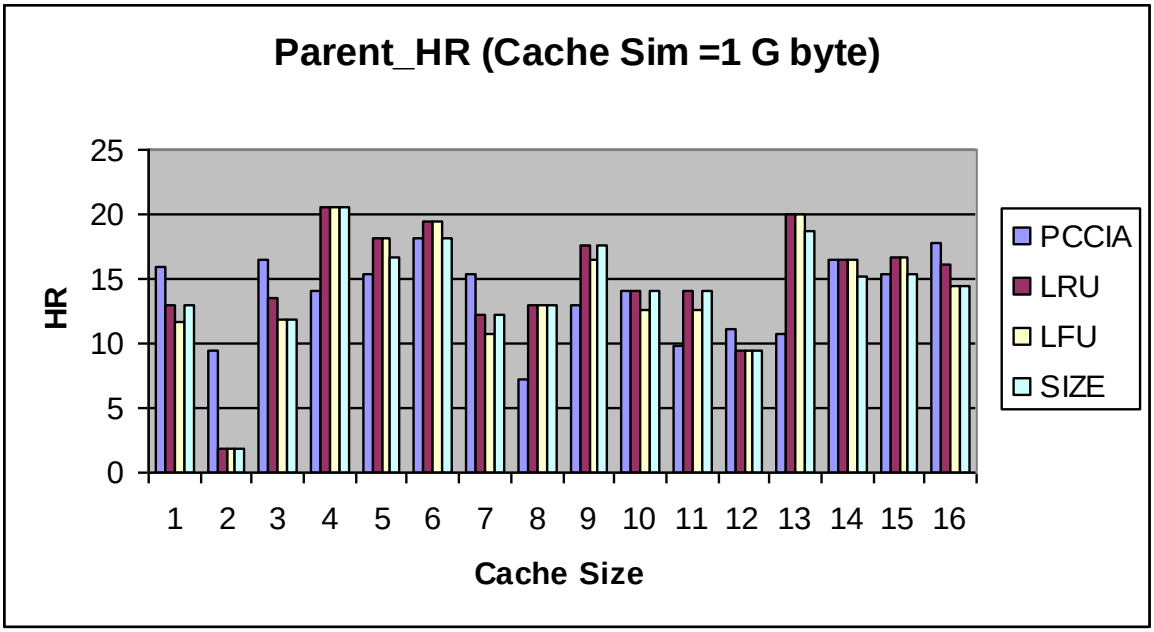


Figure 6.27: Parent HR (Cache Sim = 1 Gb)

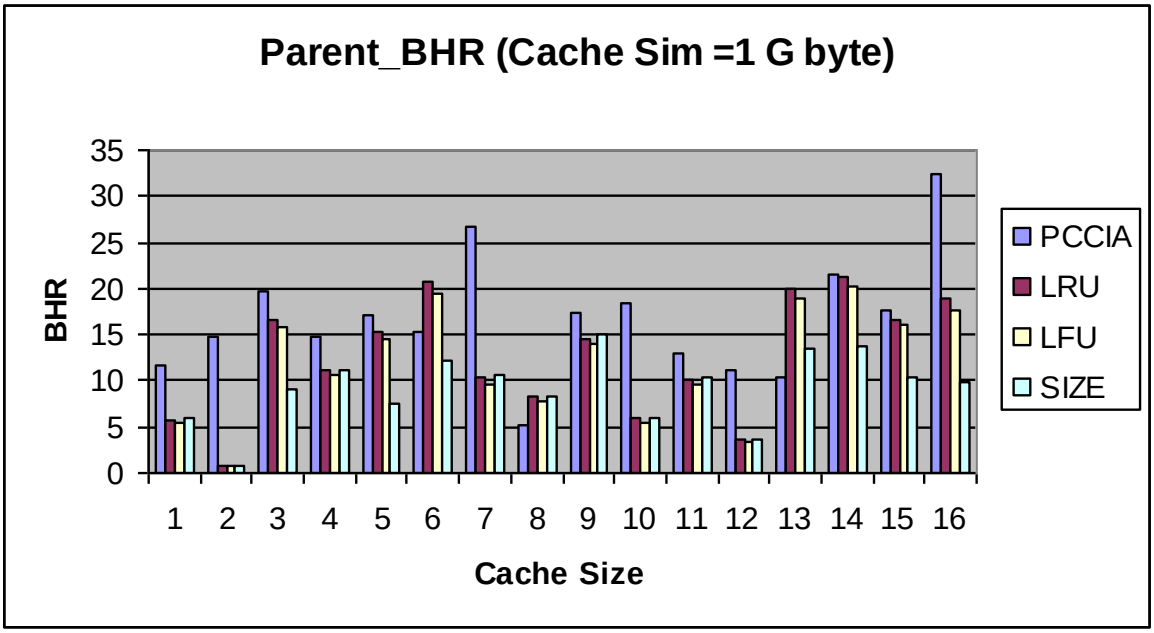


Figure 6.28: Parent BHR (Cache Sim = 1 G)

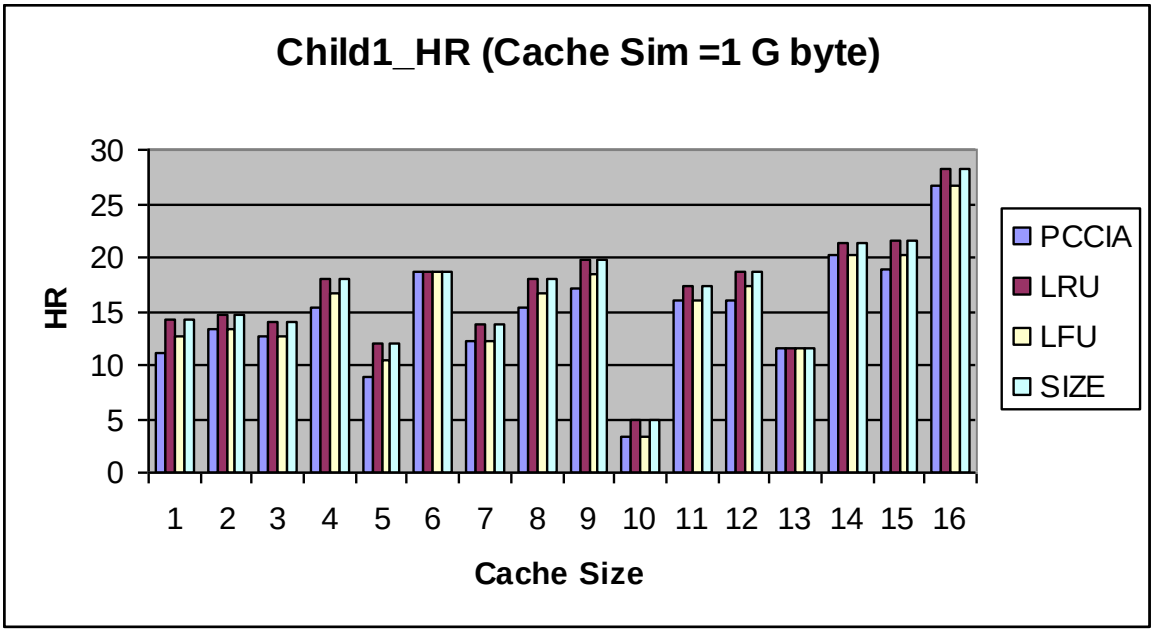


Figure 6.29: Child1 HR (Cache Sim = 1 Gb)

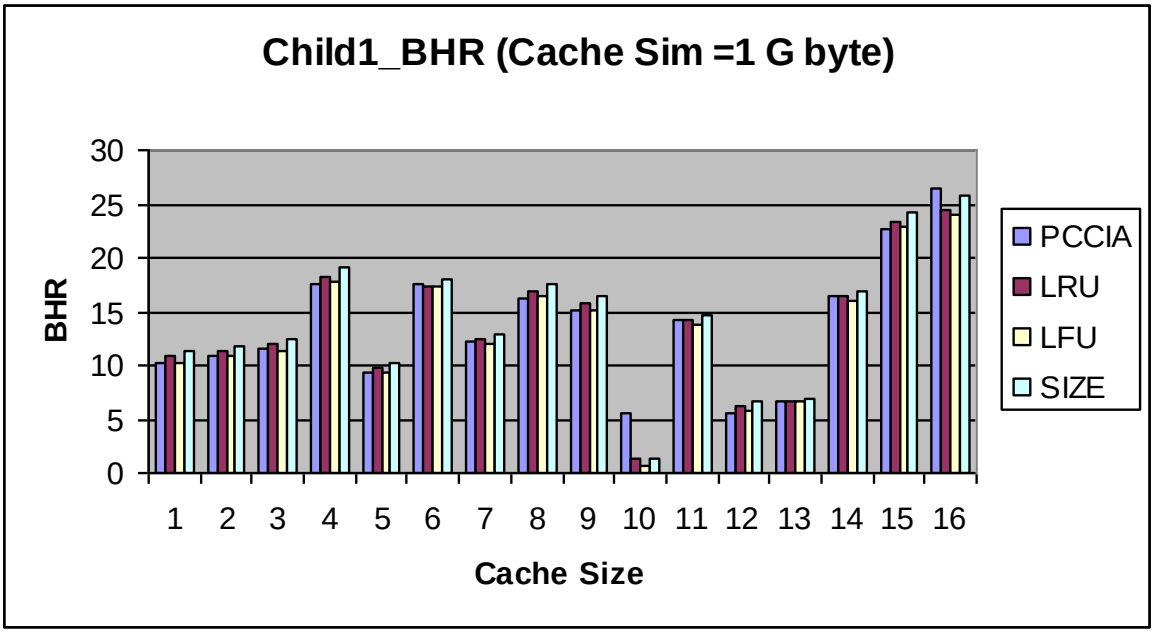


Figure 6.30: Child1 BHR (Cache Sim = 1 Gb)

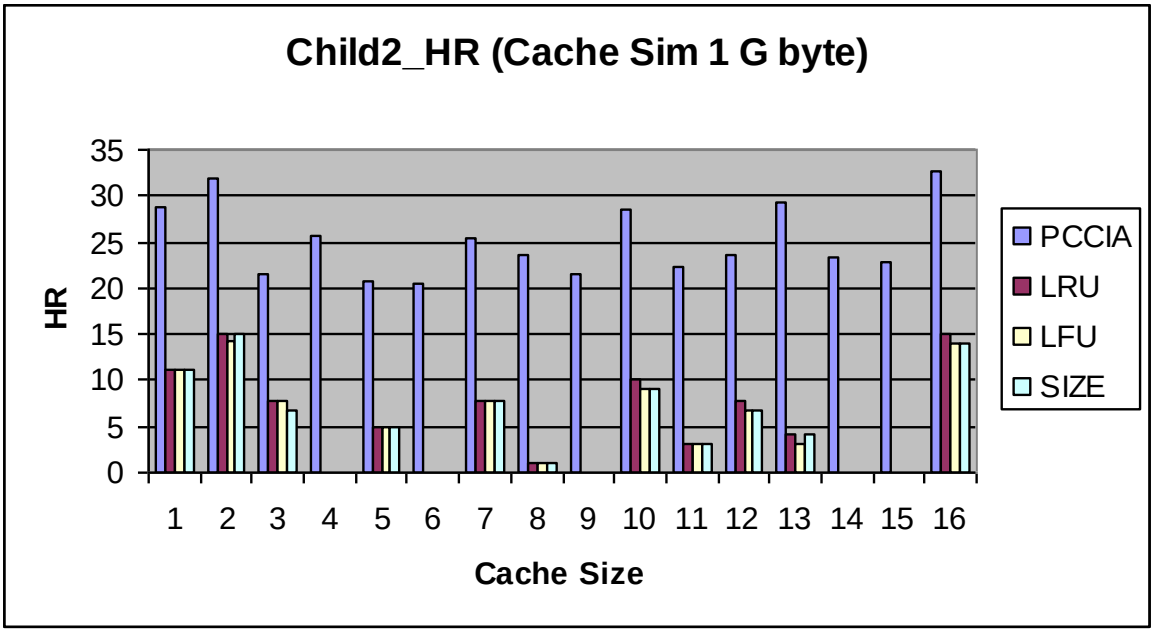


Figure 6.31: Child2 HR (Cache Sim= 1Gb)

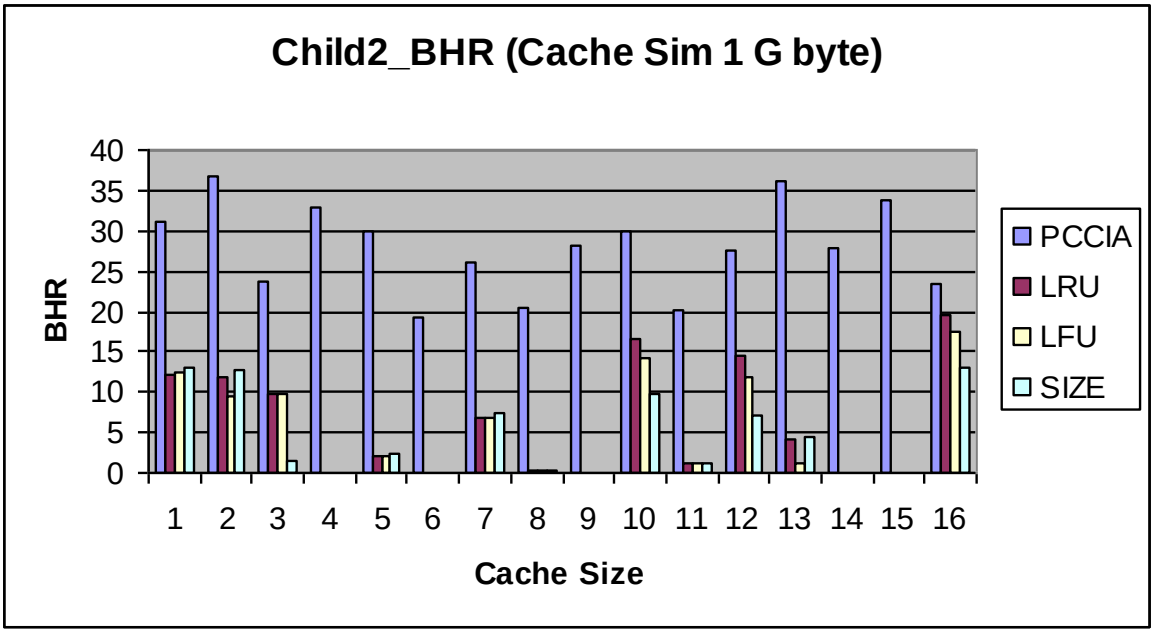
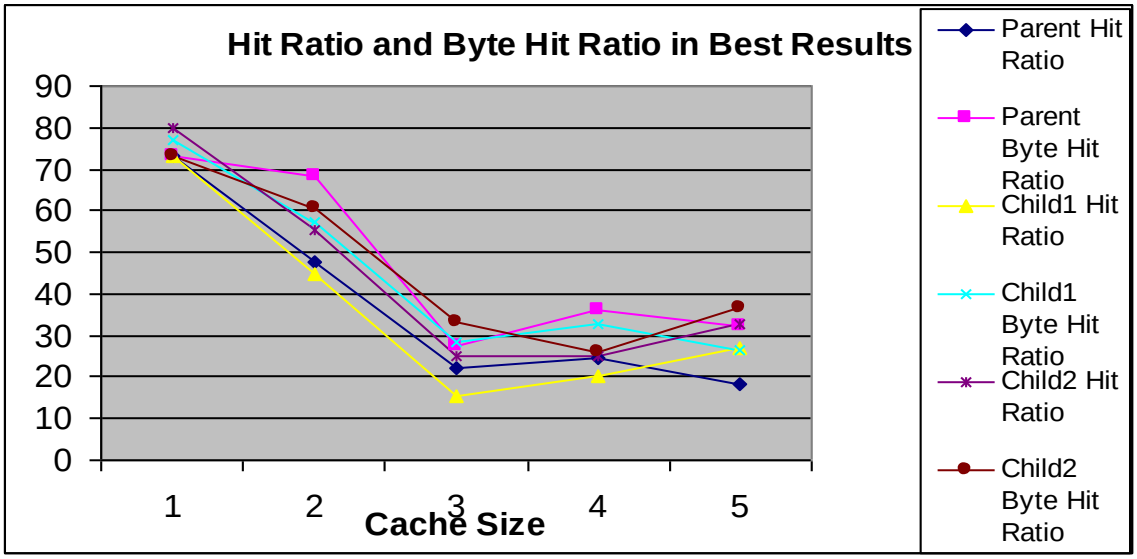
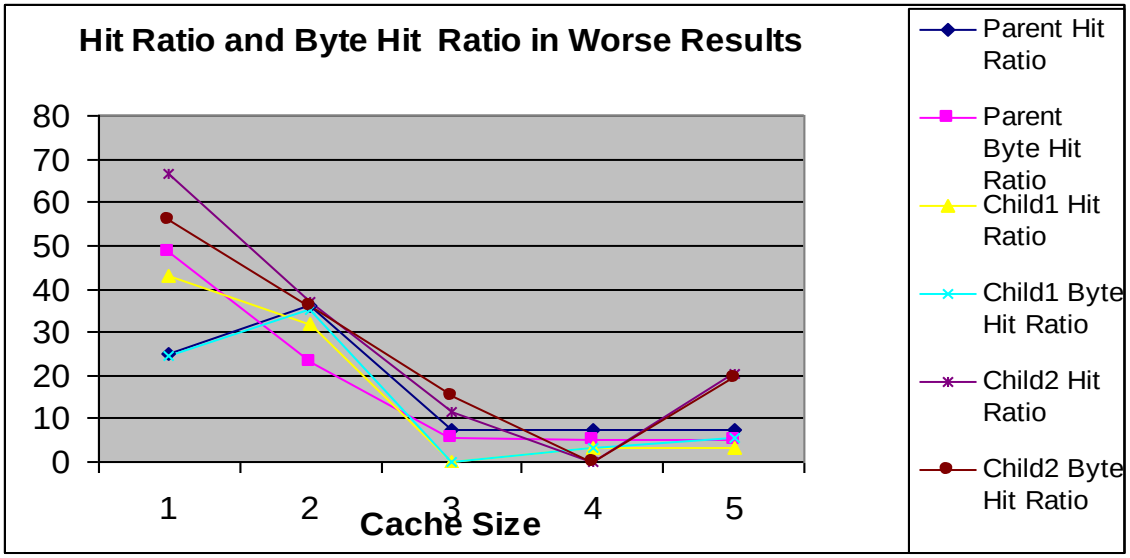


Figure 6.32: Child2 BHR (Cache Sim 1 Gb)



6.33: Hit Ratio and Byte Hit Ratio in Best Results



6.34: Hit Ratio and Byte Hit Ratio in Worse Results

6.3 Results Discussion

The simulation results are illustrated that the cache size plays a crucial role in improving the performance of the web cache. When the cache size increases, the new approach PCCIA has better performance over the LRU, LFU and Size traditional replacement polices in terms of hit rate and byte hit rate.

We can conclude some remarks from the simulation results as follows:

6.4 Simulation Results for Parent Cache

Figures 6.9, 6.15, 6.21, 6.27 and 6.33 give a comparison of PCCIA with LRU, LFU and Size algorithms in parent cache in term of hit rate.

- From Figure 6.9, in case of hit rate, for a cache size of 14 MB, there is a performance gain 6.66 (from66.67% to %73.33) over LRU.

- From Figure 6.15, in case of hit rate, for a cache size of 15 MB, there is a performance gain of 6.77% (from 33.33% to 40 %) over LFU and 5.33% (from 34.67% to 40 %) over Size.

- From Figure 6.21, in case of hit rate, for a cache size of 9 MB, there is a performance gain of 4.70% (from 17.65 % to 22.35 %) over LRU, 5.88% (from 16.47 % to 22.35 %) over LFU and 4.70% (from 17.65% to 22.35 %) over Size.

- From Figure 6.27, in case of hit rate, for a cache size of 1 MB, there is a performance gain of 14.50 % (from 10.14 % to 24.64 %) over LRU, 16.57 % (from 8.7 % to 24.64 %) over LFU and 14.50 % (from 10.14 % to 24.64 %) over Size.

- From Figure 6.33, in case of hit rate, for a cache size of 7 MB, there is a performance gain of 3.7 % (from 12.31 % to 15.38%) over LRU, 4.67 % (from 10.77% to 15.38 %) over LFU and 3.7% (from 12.31 % to 15.38 %) over Size.

Figures 6.10, 6.16, 6.22, 6.28 and 6.34 give a comparison of PCCIA with LRU, LFU and Size algorithms in parent cache in term of byte hit rate

- From Figure 6.10 In case of byte hit rate, for a cache size of 14 MB, there is a performance gain of 14.49 % (from 58.51 % to 73%) over LRU, 1.71 % (from 71.29 % to 73 %) over LFU and 10.48 % (from 62.52% to 73%) over Size.
- From Figure 6.16 In case of byte hit rate, for a cache size of 13 MB, there is a performance gain of 1.07 % (from 47.42% to 48.49 %) over LFU and 11.75 % (from 36.74% to 48.49 %) over Size.
- From Figure 6.22 In case of byte hit rate, for a cache size of 1MB, there is a performance gain of 7.99 % (from 14.66 % to 22.65%) over LRU, 11.13 % (from 11.52 % to 22.65%) over LFU and 6.39% (from 16.26% to 22.65%) over Size.
- From Figure 6.28 In case of byte hit rate, for a cache size of 1MB, there is a performance gain of 10.17 % (from 14.4% to 24.57%) over LRU, 17.05 % (from 7.52% to 24.57%) over LFU and 8.63 % (from 15.94% to 24.57%) over Size.
- From Figure 6.34 In case of byte hit rate, for a cache size of 16 MB, there is a performance gain of 13.50 % (from 18.95 % to32.45 %) over LRU, 14.78 % (from 17.67 % to 32.45%) over LFU and 22.49 % (from 9.96% to 32.45%) over Size.

6.5 Simulation Results for Child 1 Cache

Figures 6.11, 6.17, 6.23, 6.29 and 6.35 give a comparison of PCCIA with LRU, LFU and Size algorithms in child1 cache in term of hit rate.

- From Figure 6.11 In case of hit rate, for a cache size of 15 MB, there is a equality in performance PCCIA and LFU 73.33%.
- From Figure 6.17 In case of hit rate, for a cache size of 15 MB, there is a equality in performance PCCIA with LFU and Size 44.21 % and (from 44.21% to 53.68 %) under LRU
- From Figure 6.23 In case of hit rate, for a cache size of 15 MB, there is a performance gain of 1.27 % (from 13.92% to 15.19 %) over LFU and Size.
- From Figure 6.29 In case of hit rate, for a cache size of 15 MB, there is a equality in performance PCCIA and LFU 18.99
- From Figure 6.35 In case of hit rate, for a cache size of 16 MB, there is a performance gain to LRU and Size of 1.41 % (from 26.76 % to 28.17%) over PCCIA, PCCIA similar to LFU 26.76 %.

Figures 6.12, 6.18, 6.24, 6.30 and 6.36 give a comparison of PCCIA with LRU, LFU and Size algorithms in child 1 cache in term of byte hit rate.

- From Figure 6.12 In case of byte hit rate, for a cache size of 15 MB, there is a equality in performance PCCIA with LFU 77.18.
- From Figure 6.18 In case of byte hit rate, for a cache size of 15 MB, there is a performance gain 16.91(from 39.58% to 56.49 %) over LRU, 14.48 % (from 42.1% to 56.49 %) over LFU and Size.
- From Figure 6.24 In case of byte hit rate, for a cache size of 15 MB, there is a performance gain of 0.04 % (from 28.32 % to 28.36%) over LRU, 0.38 % (from 27.98 % to 28.36 %) over LFU and4.61% (from 23.75 % to 28.36 %) over Size.
- From Figure 6.30 In case of byte hit rate, for a cache size of 16 MB, there is a performance gain 0.86%(from 24.59 % to 25.45%) over LFU and 4.73 % (from 20.72 % to 25.45 %) over Size.

- From Figure 6.36 In case of byte hit rate, for a cache size of 16 MB, there is a performance gain of 1.94 % (from 24.54 % to 26.48 %) over LRU, 2.46 % (from 24.02 % to 26.48 %) over LFU and 0.79 % (from 25.69 % to 26.48 %) over Size.

6.6 Simulation Results for Child 2 Cache

Figures 6.13, 6.19, 6.25, 6.31 and 6.37 give a comparison of PCCIA with LRU, LFU and Size algorithms in child 2 cache in term of hit rate.

- From Figure 6.13 In case of hit rate, for a cache size of 15 MB, there is a performance gain of 6.70 % (from 62.5 % to 68.75 %) over LRU.
- From Figure 6.19 In case of hit rate, for a cache size of 16 MB, there is a equality in performance PCCIA with LFU 48.76 %
- From Figure 6.25 In case of hit rate, for a cache size of 16 MB, there is a performance gain of 3.04 % (from 11.51% to % 14.55) over LRU, 6.67 % (from 7.88 % to 14.55%) over LFU and 1.61 % (from 12.94 % to 14.55 %) over Size.
- From Figure 6.31 In case of hit rate, for a cache size of 16 MB, there is a equality in performance PCCIA with LFU 25.23
- From Figure 6.37 In case of hit rate, for a cache size of 16 MB, there is a performance gain of 17.76 % (from 14.95 % to 32.71%) over LRU, 18.69 % (from 14.02% to 32.71%) over LFU and 18.69 % (from 14.02 % to 32.71 %) over Size.

Figures 6.14, 6.20, 6.26, 6.32, 6.38 give a comparison of PCCIA with LRU, LFU and Size algorithms in child 2 cache in term of byte hit rate.

- From Figure 6.14 In case of byte hit rate, for a cache size of 15 MB, there is a performance gain of 3.92 % (from 55.69 % to 59.61 %) over LRU.
- From Figure 6.20 In case of byte hit rate, for a cache size of 14 MB, there is a performance gain of 0.64% (from 55.38 % to % 56.02) over LRU, 0.04 % (from 55.98 % to 56.02 %) over LFU
- From Figure 6.26 In case of byte hit rate, for a cache size of 16 MB, there is a performance gain of 9.22 % (from 19.2 % to 28.42 %) over LRU, 10.53 % (from 17.89 % to 28.42 %) over LFU and 13.73 % (from 14.69 % to 28.42 %) over Size.

- From Figure 6.32 In case of byte hit rate, for a cache size of 10 MB, there is a performance gain of 0.3 % (from 19.83 %to 19.86 %) over LRU, 0.13 % (from 19.73 % to 19.86 %) over LFU.
- From Figure 6.32 In case of byte hit rate, for a cache size of 13 MB, there is a performance gain of 31.99 % (from 4.14 % to 36.13 %) over LRU, 35.28% (from 1.08 % to 36.13%) over LFU and 32.8 % (from 4.5 % to 36.13 %) over Size.

CHAPTER 7:

CONCLUSION AND RECOMMENDATIONS

7.1 Conclusion

We conclude that intelligent agent, can be used to monitor the proxy cache and control the cleanup task.

Cache cleaner agent can remove the web object proactively when it has high clean up priority.

Fuzzy logic can be used to combine LRU, LFU and Size replacement polices to optimize proxy cache performance.

Simulation results show that the new approach PCCIA performs better than LRU, LFU and Size replacement polices in terms of hit rate and byte hit rate when the cache size increase. Simulation results achieved a hit ratio of 73.33 % in the best result and 7.25% in the worst result, and a byte hit ratio of 73.00% in the best result and 5.12% in the worst result on the parent cache side. Results for the child1 cache side are a hit ratio of 73.33% in the best result and 0% in the worst result, and a byte hit ratio of 77.18 % in the best result and 0% in the worst result. Results for the child2 cache side are a hit ratio of 80.00 % in the best result and 0% in the worst result, and a byte hit ratio of 73.05% in the best result and 0% in the worst result.

Reactive Coordination has been applied between the parent and child cleaner agents to achieve the cleanup task in efficient way.

Q_learning algorithm has been implemented to avoid difficult calculation when it reached a similar state and take a suitable action.

7.2 Future Work

- In the future, other simulator can be used to generate cache size large than 1 G byte.
- This method ignored latency time (download time of objects) in taken the replacement decision.
- Moreover, Inference rules and coordination rules can be simplified to help the cleaner agent to take a quick decision

REFERENCES

1. V.R.R. P. N. Vijaya Kumar, " Novel Web Proxy Cache Replacement Algorithms using Machine Learning Techniques for Performance Enhancement," *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY*, 2014.

2. S.M.S. Sarina Sulaiman, Ajith Abraham2 and Shahida Sulaiman, " Rough Set Granularity in Mobile Web Pre-Caching", 2009.

3. G.K. RAJEEV TIWARI, LALIT GARG, " Robust Distributed Web caching: A Dynamic clustering Approach," vol. 3 No.2, 2011.

4. S.M.S. Sarina Sulaiman, Fadni Forkan, Ajith Abraham, "Autonomous SPY©: Intelligent web proxy caching detection using neurocomputing and particle swarm optimization" *ACM*, 2009, pp. 6.

5. S.M.S. Sarina Sulaiman, Ajith braham, "Rough Web Caching," 2009.

6. S.M.S. Waleed Ali , and Abdul Samad Ismail, "A Survey of Web Caching and Prefetching " *Book A Survey of Web Caching and Prefetching* Series A Survey of Web Caching and Prefetching 3, ed., Editor eds., 2011, pp. 25.

7. S.M.S. Sarina Sulaiman, Fadni Forkan, Ajith Abraham and Azmi Kamis, " Intelligent Web Caching for E-learning Log Data," 2009.

8. J.L. F. Khalil, H. Wang, "An Integrated Model for Next Page Access Prediction", 2009.

9. S.M.S.m. Waleed Ali prowalid, “Integration of Least Recently Used Algorithm and Neuro-Fuzzy System into Client-side Web Caching

” *International Journal of Computer Science and Security (IJCSS)*, vol. 3, no. 1, 2009, pp. 15.

10. R.C.P. J. B. PATIL, B. V. PAWAR, “Integrating Intelligent Predictive Caching and Static Prefetching in Web Proxy Servers

” *International Journal on Computer Science and Engineering (IJCSE)*, vol. 3, 2011, pp. 8.

11. S.M.S. Sarina Sulaiman, Ajith Abraham, “Rough Neuro-PSO Web Caching and XML Prefetching for Accessing Facebook from Mobile Environment”, 2009.

12. J.S.Greshma, “Latency Reduction in mobile environment,” *Elsevier*, 2012.

13. P.B.a.S. Raheja, “A Vague Improved Markov Model Approach for Web Page Prediction”,*International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. .5, No.2, 2014.

14. J.K.G. JITENDRA SINGH KUSHWAH, BRIJESH PATEL#3, “Review LRU Algorithm to Implement Proxy Server with Caching Policies,” *International Journal of Engineering Science and Technology (IJEST)*, vol. 3 No.10, 2011.

15. B.V.P. J. B. Patil1, R. C. Patel “Improving Performance on WWW using Intelligent Predictive Caching for Web Proxy Servers ” *IJCSI*

International Journal of Computer Science Issues, vol. 8, no. 1, January 2011.

16. A.J. Sirshendu Sekhar Ghosh, "Energy Efficient Proxy Prefetch-Cache Framework in Clustered Architecture," *IJCTA Int.J.Computer Technology & Applications*, vol. 5 (2),669-677, 2014.

17. Y.H. Yongrui Xu " A High Scalability and High Cache Hit Ratio Replacement Algorithm," *Book A High Scalability and High Cache Hit Ratio Replacement Algorithm*, Series A High Scalability and High Cache Hit Ratio Replacement Algorithm, ed., Editor ed.^eds., 2011, pp.

18. H.E.a.S. Romano, "Comparison of Function Based Web Proxy Cache Replacement Strategies," *IEEE*, 2009.

19. B.H. Naizheng, Chen, "A Least Grade Page Replacement Algorithm for Web Cache Optimization," *Knowledge Discovery and Data Mining, 2008. WKDD 2008. First International Workshop on*, pp. 469-472.

20. D.N.A.G. K. Geetha, Monikandan S, "SEMALRU: An Implementation of modified web cache replacement algorithm," *IEEE*, 2009.

21. E.M. Georgios Kastaniotis, Vasileios Dimitsas, Christos Douligeris , Dimitris K. Despotis, "Web Proxy Caching Object Replacement: Frontier Analysis to Discover the 'Good-Enough' Algorithms," *IEEE*, 2008.

22. E.C. F.J. Gonzalez-Canete, A. Trivino-Cabrer, "An Application Level Caching Scheme Implementation For MANETS Using NS-2," *International Conference Applied Computing*, 2009.

23. G.P. Sajeev, and M.P. Sebastian, "A scheme for adaptive web caching based on multi level object classification," *Intelligent and Advanced Systems (ICIAS), 2010 International Conference on*, pp. 1-6.

24. T. Berczes, "Approximation approach to performance evaluation of Proxy Cache Server systems," 2009.

25. S.M. Abid, and H. Youssef, "Impact of One-Timer/N-Timer Object Classification on the Performance of Web Cache Replacement Algorithms," *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, pp. 208-211.

27. F.K.G.K.T.M.P.a.S. Fdida, "The effect of caching on a model of content and access provider revenues in information-centric networks————————————," 2013.

28. V.P.a.G. Dan, "Content-peering Dynamics of Autonomous Caches in a Content-centric Network," *IEEE*, 2013.

29. P. Jomsri, "Improving the Performance of Proxy Server by Using Data Mining Technique", 2013.

30. S.S.a.V.S. Vijayan R, "A Fuzzy based Anti Spam SMTP Proxy Server Engine with Performance Tuning" *IJARCS*, 2010.

31. S.S. Waleed Ali, Norbahiah Ahmad, "Performance Improvement of Least-Recently-Used Policy in Web Proxy Cache Replacement Using Supervised Machine Learning," *Int. J. Advance*, vol. 6, 2014.

32. N.e.-K.H.A.R. Mohamed, “A Proposed Model for Web Proxy Caching Techniques to Improve Computer Networks Performance ” *I.J. Information Technology and Computer Science,*, 2013

33. A.P.a.S. Sharma, “Reviewof Web Pre-Fetching and Caching Algorithms ” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. No. 3, no. No. 1, 2014.

34. D.R. CH, “Study of The Web Caching Algorithms for Performance Improvement of The Response Speed,” *Indian Journal of Computer Science and Engineering*, vol. 3 No. 2, 2012.

35. N.G. K Muralidhar, “Improving the Performance of Browsers Using Fuzzy Logic,” *International Journal of Engineering Research and Development*, vol. 3, no. 1, 2012.

36. R.C.P. J. B. PATIL, B. V. PAWAR, “Integrating Intelligent Predictive Caching and Static Prefetching in Web Proxy Servers

” *International Journal on Computer Science and Engineering (IJCSE)*, vol. 3, 2011, pp. 8.

37. A.K. Aparna N. Gupta, “A Review: Study of Various Clustering Techniques in Web Usage Mining,” *International Journal of Advanced Research in Computer and Communication Engineering* vol. 3, no. 3, 2014.

38. S.M.S.a.A.A. Sarina Sulaiman, “Intelligent Web Caching Using Adaptive Regression Trees, Splines, Random Forests and Tree Net,” 2009.

39. C. Kumar, “Performance evaluation for implementations of a network of proxy caches”, 2009.

40. M.K.a.P.N. Abdullah Balamash, “Performance Analysis of a Client-Side Caching/Prefetching System for Web Traffic.”

41. S.M.S. Sarina Sulaiman, Ajith Abrahamy, Shahida Sulaiman, “Intelligent Web Caching Using Machine Learning Methods ”, 2011.

42. S.M.S. Sarina Sulaiman, Ajith braham, “Rough Web Caching,” 2009.

43. P. Jomsri, “Improving the Performance of Proxy Server by Using Data Mining Technique,” *World Academy of Science, Engineering and Technology*, 2013.

44. M.K. Jinsuk Baek, Paul S. Fisher, and Elva J. Jones, “AWeb Object Management Policy for Cooperative Hybrid Caching Architecture” *IEEE*, 2009.

45. S.M.S.a.A.S.I. Waleed Ali, “Intelligent Bayesian Network-Based Approaches for Web Proxy Caching,” 2011.

46. H.E. Jake Cobb, “Web proxy cache replacement scheme based on back-propagation neural network,” *ScienceDirect*, vol. 81, no. 9, 2008, pp. 20.

47. H.E. Jake Cobb, “Traing and Simulation of NeuralNet Works for Web Proxy Cache Replacement,” 2006.

48. B.V.P. J. B. Patil, “Trace driven simulation of GDSF\# and existing caching algorithms for web proxy servers,” *Book Trace driven simulation of*

GDSF\# and existing caching algorithms for web proxy servers, Series Trace driven simulation of GDSF\# and existing caching algorithms for web proxy servers, ed., Editor ed.^eds., World Scientific and Engineering Academy and Society (WSEAS), 2007, pp.

49. P.B. S. HIRANPONGSIN, "NTELLIGENT CACHE FARMING ARCHITECTURE WITH THE RECOMMENDER SYSTEM," 2008.

50. G.V. Maria Cala Calzarossa, "A Fuzzy Algoritm for Web Caching," 2003.

51. J.X.a.J. Wang, "A Similarity-Aware Multiagent-BasedWeb Content Management Scheme," *Springer*, 2006.

52. M.H. Mojtaba Sabeghi1, Yaghmaee,Ferdowsi, "Using Fuzzy Logic to Improve Cache Replacement Decisions" *IJCSNS International Journal of Computer Science and Network Security*, vol. 6, 2006, pp. 7.

53. M. Piatek, "Distributed Web Proxy Caching in a Local Network Environment," 2004.

54. J.L.a.J.C.S.L. Alan T.S, "COPACC: A Cooperative Proxy-Client Caching System for On-Demand Media Streaming ", 2005.

55. E.C. Wenzhong Li, Yilin Wang, Daoxu Chen, and Sanglu Lu, "Cache Placement Optimization in Hierarchical Networks: Analysis and Performance Evaluation," 2006.

56. O. Hyung Rai, and S. Hwangjun, "[A Novel Dynamic and Scalable Caching Algorithm of Proxy Server for Multimedia Objects," *J. VLSI Signal Process. Syst.*, vol. 46, no. 2-3, 2007, pp. 103-112.

57. C.K.a.J.B. Norris, "A new approach for a proxy-level web caching mechanism," *Elsevier*, 2008.

58. A.V. George Pallis , Jaroslav Pokorny "A clustering-based prefetching scheme on a Web cache environment," *Elsevier*, 2008.

60. Y.W.T.a.Y.K. Chang, "A Novel Cooperative Caching Scheme for Wireless Ad Hoc Networks: GroupCaching", 2007.

61. S.T. Richa Gupta, "Pair of Replacement Algorithms MFMR and AFLRU on L1 and L2 Cache for Proxy Server," *IEEE*, 2005.

62. B.L. Xiang Zhan Yu, "Anew Approach to fragment Level Caching of Dynamically Generatted Web Content in Forward Proxies " *International Conference on Machine Learning and Cybernetics*, pp. 6.

63. F.B. Fernando Duarte, Virgilio Almeida, Jussara Almeida, "Locality of Refrence in an Hierarrchy of Web Caches," 2006.

64. , "Improving Performance of World Wide Web by Adaptive Web Traffic Reduction," 2006.

66. S.B.a.J.K. Liang Wang, "Cooperation Policies for Efficient In-Network Caching " *ACM*, 2013.

67. O.G. Simon Gay, "InteractionBased Space Representation for EnvironmentAgnostic Agents," 2013.

68. J.Z. Qing Li, and Xinzhong Zhu, “Mobile Learning Support with Statistical Inference-Based Cache Management,” *Springer*, 2008.

69. L.B.a.P. Ciancarini, “A Perspective on Multiagent Coordination Models,” *Springer*, 2003.

70. D.A.W. P. Sengottuvelan, Dr. A. Shanmugam, “An Approach for CMA Coordination behavior in Teamwork of Multi-Agent System,” *International Journal of Recent Trends in Engineering*, vol. 1.No 1, 2009.

71. S.D.R.a.C. Skinner, “Designing a Multi-Agent Coordination Competition,” 2009.

72. B.C.-d. Patrick Beaumont, “Multiagent Coordination Techniques for Complex Environments,” 2004.

73. S.J.O. Miller, “Multi-Agent Based Techniques for Coordinating the Distribution of Electricity in a Micro-Grid Environment”, 2010.

74. D.-y.L. Li Jiang, “A Survey of Multi-agent Coordination,” 2001.

75. M.R. Nicola Gatti, Tuomas Sandholm and Carnegie Mellon, “On the Verification and Computation of Strong Nash Equilibrium”, 2010.

76. R.P. Yakov Babichenko∗, “Approximate Nash Equilibria via Sampling,” July 19, 2013.

77. A.R.-K. Mostafa Sahraei-Ardakani, Majid Nili-Ahmadabadi, “Hierarchical Nash-Q Learning in Continuous Games” *IEEE*, 2008.

78. Y. Averboukh, “Universal Nash Equilibrium Strategies for Differential Games”, June 2013.

79. A.R.-K. M. Sahraee Ardakani, M. Nili Ahmadabadi, “Hierarchical Nash-Cournot Q-Learning in Electricity Markets” *The International Federation of Automatic Control*, July 6-11, 2008.

80. b. Wenzhong Li a, Edward Chan b, Guofu Feng c, Daoxu Chen a, Sanglu Lu a, “Analysis and performance study for coordinated hierarchical cache placement strategies,” *Elsevier*, 2010.

81. S.L. Maxim Claeys, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck and Filip De Turck, “Design of a Q-Learning-based Client Quality Selection Algorithm for HTTP Adaptive Video Streaming,” 2013.

82. E.M.d.C.a.L.E.S. Pablo Hernandez-Leal, Learning against non-stationary opponents,” 2013.

83. M.A.-L. Kenji Kawaguchi, “A Greedy Approximation of Bayesian Reinforcement Learning with Probably Optimistic Transition Model,” 2013.

84. E.M.H. Eric Sodomka, Michael L. Littman, Amy Greenwald, “Coco-Q: Learning in Stochastic Games with Side Payments” *International Conference on Machine Learning*, vol. 28, 2013.

85. H.Y. Dimitri P. Bertsekas, “Q-Learning and Enhanced Policy Iteration in Discounted Dynamic Programming,” *MATHEMATICS OF OPERATIONS RESEARCH*, vol. 37, No. 1, 2012, pp. 66–94.

86. D.N.S. P. Singhala, B. Patel, “Temperature Control using Fuzzy Logic,” *International Journal of Instrumentation and Control Systems (IJICS),* vol. .4, No.1, January 2014.

87. L. A.Zadeh, “Towardextendedfuzzylogic-Afirststep,” 2009.

88. N.M.C. Williamson, “WebTraff: A GUI for Web Proxy Cache Workload Modeling and Analysis,” 2003.

89. W.C. Giovanni Caire, Francisco Garijo, Jorge Gomez, Juan Pavon,Francisco Leal, Paulo Chainho, Paul Kearney, Jamie Stark, Richard Evans, and Philippe Massonet, “Agent Oriented Analysis Using Message/UML,” *Springer*, 2002.

90. W.C. Giovanni Caire, Francisco Garijo, *The Message Methodology*.

91. M.i. Wooldridge, “An Introduction to MultiAgent Systems.”

92. M. Mihaylov, “Decentralized Coordination in Multi-Agent Systems

”, July, 2012.

93. J.R. Kok, “Coordination and Learning in Cooperative Multiagent Systems,” 2006.

94. C.P. Morgan, Algorithms for Reinforcement Learning, 2013.

95. N.V. Reinoud Elhorst, Anytime algorithms for multi-agent decision making, 2004.

96. G.C. Fabio Bellifemine, Tiziana Trucco, *Jade Administrator's Guide* 2005.

97. “http://www.squid-cache.org/.”

98. M. Negnevitsky, *Artificial Intelligence A Guide to Intelligent Systems* 2005.

APPENDIX

Result's Tables:

Table 1: Parent HR Cache Sim =1 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	60	60	70	70
2	66.67	66.67	66.67	66.67
4	66.67	58.33	75	66.67
8	72.73	72.73	72.73	72.73
16	58.33	50	66.67	66.67
32	63.64	63.64	72.73	72.73
64	66.67	60	73.33	73.33
128	50	50	62.5	62.5
256	42.86	42.86	57.14	57.14
512	71.43	71.43	78.57	78.57
1024	50	50	62.5	62.5
2048	25	25	25	25
4096	42.86	42.86	57.14	57.14
8192	73.33	66.67	80	73.33
16384	50	50	62.5	62.5
32768	61.54	61.54	69.23	76.92

Table 2: Table Parent BHR Cache Sim = 1 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	59.8	69.12	54.23	76.4
2	62.65	54.76	51.56	74.41
4	69.31	51.39	66.75	55.01
8	67.23	80	58.66	79.5
16	65.72	44.44	62.42	51.62
32	62.32	72.31	57.67	78.82
64	70.46	54.36	68.34	61.02
128	53.61	59.87	45.36	69.4
256	48.36	53.17	37.35	61.96
512	66.59	81.42	63.29	82.49
1024	53.61	65.97	45.36	69.4
2048	42.56	37.3	15.07	32.65
4096	48.36	53.17	37.35	61.96
8192	73	58.51	71.29	62.52
16384	53.61	59.87	45.36	69.4
32768	61.71	72.84	57.18	80.33

Table 3: Child 1 HR Cache Sim = 1 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	60	70	60	70
2	73.33	73.33	73.33	73.33
4	50	62.5	50	62.5
8	60	70	60	70
16	69.23	76.92	69.23	76.92
32	20	40	20	40
64	50	62.5	50	62.5
128	50	62.5	62.5	62.5
256	66.67	75	66.67	75
512	60	70	70	70
1024	71.43	78.57	71.43	78.57
2048	71.43	78.57	71.43	78.57
4096	66.67	75	75	75
8192	66.67	75	66.67	75
16384	73.33	80	73.33	80
32768	42.86	57.14	42.86	57.14

Table 4: Child1 BHR Cache Sim = 1 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	63.63	75.51	64.96	75.51
2	76.62	76.62	77.18	76.62
4	54.57	67.88	54.57	67.88
8	63.63	74.54	63.63	74.54
16	73.18	81.94	73.91	81.94
32	24.61	44.89	24.61	44.89
64	52.31	66.2	52.31	66.2
128	56.62	70.79	67.72	70.79
256	69.74	79.63	70.67	79.63
512	66.19	77.24	74.6	77.24
1024	75.34	83.39	75.96	83.39
2048	75.96	83.81	76.55	83.81
4096	72.31	80.25	77.86	80.25
8192	72.31	80.8	72.31	80.8
16384	77.18	84.26	77.18	84.26
32768	49.47	64.08	49.47	64.08

Table 5: Child 2 HR Cache Sim = 1 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	78.95	78.95	84.21	84.21
2	66.67	66.67	73.33	80
4	78.95	73.68	78.95	84.21
8	72.22	72.22	77.78	77.78
16	71.43	71.43	78.57	85.71
32	73.91	73.91	78.26	82.61
64	75	75	81.25	87.5
128	78.26	78.26	82.61	82.61
256	80	80	85	80
512	66.67	66.67	73.33	80
1024	76.47	76.47	82.35	76.47
2048	76.19	76.19	80.95	80.95
4096	80	75	80	80
8192	66.67	66.67	75	83.33
16384	68.75	62.5	68.75	75
32768	78.95	78.95	84.21	84.21

Table 6: Child 2 BHR Cache Sim = 1 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	72.53	71.53	75.49	71.87
2	56.29	57.14	62.52	84.26
4	68.86	65.67	68.86	85.14
8	67.21	67.21	70.25	65.13
16	59.32	61.28	65.45	87.12
32	66.66	66.66	69.61	84.54
64	63.12	63.77	68.68	88.83
128	72.23	71.37	74.81	71.63
256	73.4	73.74	77.41	69.82
512	56.12	56.12	60.2	79.99
1024	69.32	69.77	73.94	64.46
2048	71.58	72.87	75.62	71.9
4096	73.05	70.29	73.05	68.92
8192	53.81	52.58	58.79	84.33
16384	59.61	55.69	59.61	78.28
32768	67.9	68.39	72.74	84.87

Table 7: Parent HR Cache Sim = 6 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	42.5	45	43.75	43.75
2	39.39	45.45	33.33	34.85
4	47.44	48.72	38.46	37.18
8	45.35	48.84	37.21	37.21
16	45.35	47.67	41.86	41.86
32	45.95	51.35	37.84	37.84
64	42.17	50.6	38.55	37.35
128	46.74	51.09	45.65	45.65
256	43.37	46.99	37.35	36.14
512	36.14	40.96	34.94	34.94
1024	44.74	47.37	39.47	39.47
2048	46.88	50	44.79	43.75
4096	47.67	53.49	44.19	44.19
8192	40.26	44.16	40.26	40.26
16384	40	45.33	33.33	34.67
32768	40.23	49.43	35.63	35.63

Table 8: Parent BHR Cache Sim = 6 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	65.92	68.5	47.13	32.21
2	23.22	35.9	27.45	40.29
4	55.64	57.77	55.04	45.22
8	46.02	51.13	44.7	32.14
16	50.44	53.88	37.91	48.38
32	47.13	52.19	30.66	41.49
64	61.32	70.55	45.45	31.81
128	53.72	63.63	51.43	41.04
256	41.54	44.62	46.53	33.4
512	42.48	45.96	43.32	28.55
1024	68.34	70.5	48.62	35.83
2048	63.45	70.24	68.76	64.96
4096	48.49	53.92	47.42	36.74
8192	45.56	51.83	46.63	33.45
16384	38.67	49.96	30.44	41.21
32768	35.83	44.1	32.95	42.43

Table 9: Child 1 HR Cache Sim = 6 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	34.29	40	35.71	35.71
2	42.17	43.37	40.96	40.96
4	39.29	44.05	40.48	40.48
8	43.9	52.44	45.12	45.12
16	44.71	48.24	45.88	45.88
32	43.68	51.72	43.68	43.68
64	42.05	48.86	42.05	42.05
128	43.48	44.57	42.39	42.39
256	31.88	49.28	33.33	33.33
512	41.86	45.35	44.19	44.19
1024	43.18	47.73	44.32	44.32
2048	43.33	52.22	45.56	45.56
4096	41.57	48.31	43.82	43.82
8192	37.97	44.3	37.97	37.97
16384	44.21	53.68	44.21	44.21
32768	32.86	45.71	34.29	34.29

Table 10: Child 1 BHR Cache Sim = 6 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	49.35	40.72	39.54	39.54
2	50.91	18.03	29.1	29.1
4	51.19	21.24	31.5	31.5
8	46.29	41.44	46.77	46.77
16	45.71	31.46	43.54	43.54
32	46.32	35.38	35.84	35.84
64	57.51	32.68	42.91	42.91
128	51.94	23.26	32.87	32.87
256	50.28	26.94	27.31	27.31
512	43.88	37.22	44.05	44.05
1024	52.01	24.11	34	34
2048	45.54	28.45	45.46	45.46
4096	39.71	31.38	39.13	39.13
8192	35.34	25.46	28.66	28.66
16384	56.49	39.58	42.1	42.1
32768	48.06	35.89	21.06	21.06

Table 11: Child 2 HR Cache Sim = 6 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	55.47	58.59	56.25	56.25
2	50.39	56.59	51.16	51.16
4	44.83	51.72	44.83	45.69
8	42.73	51.82	41.82	42.73
16	44.86	52.34	44.86	44.86
32	45.3	54.7	44.44	45.3
64	44.86	55.14	44.86	45.79
128	37.23	51.06	38.3	38.3
256	47.62	51.59	47.62	48.41
512	45.87	60.55	45.87	46.79
1024	47.37	55.26	46.49	47.37
2048	36.96	46.74	36.96	36.96
4096	41.75	49.51	41.75	42.72
8192	51.64	57.38	51.64	52.46
16384	42.59	51.85	42.59	43.52
32768	48.76	55.37	48.76	49.59

Table 12: Child 2 BHR Cache Sim = 6 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	55.81	40.31	55.98	61.68
2	60.67	64.96	60.96	64.74
4	39.03	55.98	38.98	43.12
8	49.22	51.1	48.89	53.1
16	56.16	56.61	55.05	58.92
32	48.93	56.73	48.78	52.81
64	38.56	44.83	38.46	42.73
128	41.88	55.26	42.02	46.19
256	51.11	55.46	51.03	55.02
512	52.58	57.05	52.52	56.88
1024	45.34	49.64	45.08	49.43
2048	36.2	45.79	35.92	41.34
4096	45.91	56.44	52.28	56.59
8192	56.02	55.38	55.98	60.25
16384	45.66	54.1	46.34	50.66
32768	46.85	56.8	47.57	51.83

Table 13: Parent HR Cache Sim = 500 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	15.94	11.59	10.14	11.59
2	7.55	0	0	0
4	14.93	14.93	13.43	14.93
8	10.26	15.38	14.1	15.38
16	9.72	12.5	11.11	12.5
32	19.44	15.28	13.89	15.28
64	9.23	9.23	7.69	9.23
128	8.7	8.7	8.7	8.7
256	22.35	17.65	16.47	17.65
512	8.45	11.27	9.86	11.27
1024	15.49	18.31	16.9	18.31
2048	9.52	9.52	9.52	9.52
4096	14.67	17.33	16	17.33
8192	8.86	15.19	15.19	15.19
16384	8.97	16.67	15.38	16.67
32768	14.52	16.13	14.52	16.13

Table 14: Parent BHR Cache Sim = 500 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	22.65	14.66	11.52	16.26
2	5.71	0	0	0
4	19.14	17.16	14.06	19.07
8	13.62	8.74	5.6	9.63
16	13.22	19.52	16.93	21.36
32	15.54	17.83	14.99	19.65
64	10.24	9.68	6.45	10.71
128	5.61	9.59	9.91	10.67
256	27.35	22.95	20.71	24.87
512	9.69	9.71	6.31	10.81
1024	21.31	22.78	20.13	25.07
2048	7.42	7.49	7.74	8.35
4096	15.46	17.18	14.52	18.8
8192	11.28	15.87	16.35	17.48
16384	5.91	15.31	12.38	16.89
32768	14.55	11.51	7.88	12.94

Table 15: Child1 HR Cache Sim = 500 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	1.59	3.17	3.17	3.17
2	7.35	8.82	8.82	8.82
4	8.45	8.45	8.45	7.04
8	13.89	15.28	15.28	15.28
16	4.48	4.48	4.48	4.48
32	12.5	12.5	12.5	11.25
64	8.22	9.59	9.59	9.59
128	15.38	15.38	14.1	14.1
256	13.16	14.47	14.47	14.47
512	0	0	0	0
1024	12	13.33	13.33	12
2048	12	13.33	12	12
4096	5.8	7.25	7.25	7.25
8192	15.48	16.67	16.67	16.67
16384	15.19	15.19	13.92	13.92
32768	15.49	16.9	15.49	16.9

Table 16: Child1 BHR Cache Sim = 500 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	2.37	3.2	3.21	3.52
2	10.93	12.22	12.24	13.26
4	14.61	14.52	14.54	6.82
8	23.59	25.06	25.1	26.95
16	11.52	11.31	11.33	12.26
32	23.13	22.62	22.65	17.4
64	10.05	11.99	12.01	12.99
128	27.18	27.17	26.78	21.99
256	16	16.95	16.98	18.26
512	0	0	0	0
1024	18.05	18.72	18.75	12.1
2048	21.27	26.69	26.29	21.27
4096	4.46	6.51	6.52	7.09
8192	21.03	22.83	22.87	24.41
16384	28.36	28.32	27.98	23.75
32768	16.22	17.85	17.35	19.37

Table 17: Child 2 HR Cache Sim = 500 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	25	12.04	12.04	11.11
2	20.17	17.65	16.81	16.81
4	19.61	10.78	9.8	9.8
8	18.89	1.11	1.11	1.11
16	17.82	7.92	7.92	7.92
32	12.5	0	0	0
64	18.63	8.82	7.84	8.82
128	19.35	3.23	2.15	3.23
256	11.39	0	0	0
512	22.94	12.84	11.93	12.84
1024	17.02	4.26	3.19	3.19
2048	19.61	6.86	6.86	6.86
4096	17.71	5.21	4.17	5.21
8192	20.78	0	0	0
16384	15.66	0	0	0
32768	20.56	16.82	15.89	15.89

Table 18: Child2 BHR Cache Sim = 500 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	33.11	15.57	15.53	10.96
2	26.01	22.33	21.14	18.39
4	25.05	14.93	13.59	10.15
8	21.83	0.69	0.69	0.73
16	23.69	6.08	6.06	6.33
32	15.74	0	0	0
64	20.14	5.82	4.35	6.06
128	23.09	5.15	3.66	5.38
256	15.34	0	0	0
512	24	14.63	13.31	15.17
1024	17.67	9.03	7.59	3.56
2048	20.48	12.19	12.16	12.66
4096	23.75	8.76	7.31	9.12
8192	23.34	0	0	0
16384	18.26	0	0	0
32768	28.42	19.2	17.89	14.69

Table 19: Parent HR Cache Sim = 800 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	24.64	10.14	8.7	10.14
2	7.55	0	0	0
4	17.91	8.96	7.46	8.96
8	16.67	10.26	10.26	10.26
16	16.67	9.72	8.33	9.72
32	15.28	13.89	12.5	13.89
64	16.92	7.69	6.15	7.69
128	13.04	7.25	7.25	7.25
256	23.53	17.65	16.47	17.65
512	15.49	12.68	11.27	12.68
1024	14.08	8.45	7.04	8.45
2048	12.7	3.17	3.17	3.17
4096	18.67	14.67	13.33	14.67
8192	15.19	15.19	13.92	15.19
16384	19.23	12.82	11.54	12.82
32768	16.13	8.06	6.45	8.06

Table 20: Parent HR Cache Sim = 800 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	24.57	14.4	7.52	15.94
2	5.14	0	0	0
4	28.05	19.75	13.58	21.71
8	15.16	10.06	10.55	11.09
16	20.61	20.99	15.39	22.89
32	29.92	26.4	20.97	28.83
64	25.36	12.36	5.09	13.73
128	14.96	7.54	7.93	8.35
256	36.02	28.48	23.81	30.75
512	19.23	18.81	12.5	20.71
1024	10.55	10.78	3.5	11.97
2048	15.2	6.02	6.35	6.71
4096	26.72	27.35	22.21	29.73
8192	14.24	14.31	7.93	15.72
16384	22.29	17.57	11.76	19.18
32768	12.05	14.8	7.73	16.43

Table 21: Child1 HR Cache Sim = 800 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	4.76	4.76	3.17	4.76
2	11.76	11.76	11.76	10.29
4	9.86	9.86	8.45	9.86
8	16.67	18.06	16.67	16.67
16	8.96	10.45	8.96	8.96
32	17.5	17.5	16.25	17.5
64	10.96	10.96	10.96	9.59
128	12.82	12.82	11.54	11.54
256	11.84	13.16	11.84	13.16
512	3.33	5	5	3.33
1024	16	17.33	17.33	16
2048	16	17.33	17.33	17.33
4096	5.8	7.25	7.25	7.25
8192	20.24	21.43	20.24	20.24
16384	18.99	20.25	18.99	20.25
32768	18.31	19.72	18.31	18.31

Table 22: Child1 BHR Cache Sim = 800 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	5.75	5.73	3.99	6.17
2	19.34	19.29	19.49	13.49
4	11.07	10.93	9.39	11.7
8	28.25	28.62	27.53	24.09
16	13.77	17.18	15.78	11.08
32	17.98	17.6	16.32	18.68
64	17.57	17.62	17.78	11.99
128	16.82	16.84	15.58	11.39
256	9.74	10.17	8.76	10.83
512	10.83	11.49	11.62	4.32
1024	24.01	24.63	24.85	19.78
2048	15.53	15.98	16.14	17.02
4096	3.01	5.25	5.31	5.62
8192	32.52	33.9	33	30.28
16384	18.78	19.23	17.88	20.48
32768	25.45	25.79	24.59	20.72

Table 23: Child 2 HR Cache Sim = 800 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	18.52	19.44	18.52	19.44
2	21.01	21.01	20.17	21.01
4	15.69	16.67	15.69	16.67
8	6.67	6.67	6.67	6.67
16	11.88	12.87	12.87	12.87
32	7.95	9.09	9.09	9.09
64	18.63	18.63	18.63	18.63
128	10.75	11.83	10.75	11.83
256	0	0	0	0
512	15.6	15.6	15.6	15.6
1024	10.64	11.7	10.64	11.7
2048	14.71	14.71	14.71	14.71
4096	12.5	13.54	13.54	13.54
8192	0	0	0	0
16384	2.41	3.61	3.61	3.61
32768	25.23	26.17	25.23	26.17

Table 24: Child 2 BHR Cache Sim = 800 Mb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	20.39	21.03	20.53	21.84
2	19.65	19.82	19.33	20.55
4	14.71	15.81	15.29	16.47
8	7.35	7.38	7.33	7.73
16	6.47	8.41	8.35	8.8
32	8.73	9.04	8.98	9.5
64	18.24	18.37	18.27	19.16
128	10.47	10.76	10.22	11.26
256	0	0	0	0
512	19.86	19.83	19.73	20.61
1024	8.8	11.78	11.21	12.34
2048	18.32	18.44	18.35	19.2
4096	12.05	14.9	14.81	15.59
8192	0	0	0	0
16384	6.2	6.59	6.55	6.92
32768	25.88	26.64	26.08	27.74

Table 25: Parent HR Cache Sim = 1Gb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	15.94	13.04	11.59	13.04
2	9.43	1.89	1.89	1.89
4	16.42	13.43	11.94	11.94
8	14.1	20.51	20.51	20.51
16	15.28	18.06	18.06	16.67
32	18.06	19.44	19.44	18.06
64	15.38	12.31	10.77	12.31
128	7.25	13.04	13.04	13.04
256	12.94	17.65	16.47	17.65
512	14.08	14.08	12.68	14.08
1024	9.86	14.08	12.68	14.08
2048	11.11	9.52	9.52	9.52
4096	10.67	20	20	18.67
8192	16.46	16.46	16.46	15.19
16384	15.38	16.67	16.67	15.38
32768	17.74	16.13	14.52	14.52

Table 26: Parent BHR Cache Sim = 1Gb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	11.76	5.82	5.41	5.96
2	14.69	0.73	0.68	0.76
4	19.76	16.72	15.73	8.98
8	14.67	11.04	10.55	11.26
16	17.16	15.24	14.44	7.47
32	15.23	20.67	19.48	12.22
64	26.73	10.38	9.61	10.67
128	5.12	8.22	7.8	8.41
256	17.37	14.62	13.88	14.91
512	18.35	5.93	5.46	6.08
1024	13.06	10.21	9.52	10.47
2048	11.21	3.63	3.38	3.74
4096	10.28	19.91	19.03	13.56
8192	21.39	21.33	20.23	13.82
16384	17.6	16.68	15.95	10.33
32768	32.45	18.95	17.67	9.96

Table 27: Child 1 HR Cache Sim = 1 Gb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	11.11	14.29	12.7	14.29
2	13.24	14.71	13.24	14.71
4	12.68	14.08	12.68	14.08
8	15.28	18.06	16.67	18.06
16	8.96	11.94	10.45	11.94
32	18.75	18.75	18.75	18.75
64	12.33	13.7	12.33	13.7
128	15.38	17.95	16.67	17.95
256	17.11	19.74	18.42	19.74
512	3.33	5	3.33	5
1024	16	17.33	16	17.33
2048	16	18.67	17.33	18.67
4096	11.59	11.59	11.59	11.59
8192	20.24	21.43	20.24	21.43
16384	18.99	21.52	20.25	21.52
32768	26.76	28.17	26.76	28.17

Table 28: Child1 BHR Cache Sim = 1Gb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	10.2	10.89	10.32	11.42
2	10.88	11.41	10.95	11.85
4	11.49	11.91	11.35	12.5
8	17.65	18.26	17.75	19.07
16	9.27	9.87	9.39	10.27
32	17.49	17.28	17.3	17.96
64	12.15	12.36	11.93	12.82
128	16.15	16.88	16.43	17.55
256	15.05	15.69	15.18	16.39
512	5.53	1.28	0.68	1.35
1024	14.16	14.15	13.67	14.74
2048	5.6	6.31	5.76	6.61
4096	6.73	6.56	6.57	6.89
8192	16.36	16.36	15.96	16.94
16384	22.74	23.35	22.88	24.32
32768	26.48	24.54	24.02	25.69

Table 29: Child 2 HR Cache Sim = 1Gb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	28.7	11.11	11.11	11.11
2	31.93	15.13	14.29	15.13
4	21.57	7.84	7.84	6.86
8	25.56	0	0	0
16	20.79	4.95	4.95	4.95
32	20.45	0	0	0
64	25.49	7.84	7.84	7.84
128	23.66	1.08	1.08	1.08
256	21.52	0	0	0
512	28.44	10.09	9.17	9.17
1024	22.34	3.19	3.19	3.19
2048	23.53	7.84	6.86	6.86
4096	29.17	4.17	3.12	4.17
8192	23.38	0	0	0
16384	22.89	0	0	0
32768	32.71	14.95	14.02	14.02

Table 30: Child 2 BHR Cache Sim = 1Gb

Cache Size (k)	PCCIA	LRU	LFU	Size
1	30.99	12.15	12.35	13.06
2	36.72	11.99	9.5	12.85
4	23.75	9.72	9.88	1.57
8	32.85	0	0	0
16	29.9	2.08	2.12	2.26
32	19.3	0	0	0
64	26.07	6.82	6.94	7.36
128	20.55	0.37	0.37	0.4
256	28.11	0	0	0
512	29.83	16.54	14.2	9.63
1024	20.2	1.09	1.11	1.19
2048	27.69	14.39	11.89	6.97
4096	36.13	4.14	1.08	4.5
8192	27.8	0	0	0
16384	33.92	0	0	0
32768	23.46	19.69	17.37	12.93

Table 31: Hit Ratio and Byte Hit Ratio in Best Results

Cache Size	Parent Cache		Child1 Cache		Child2 Cache	
	Hit Ratio	Byte Hit Ratio	Hit Ratio	Byte Hit Ratio	Hit Ratio	Byte Hit Ratio
1 Mb	73.33	73	73.33	77.18	80.00	73.05
6 Mb	47.67	68.34	44.71	57.51	55.47	60.67
500 Mb	22.35	27.35	15.49	28.36	25.00	33.11
800 Mb	24.64	36.02	20.24	32.52	25.23	25.88
1 Gb	18.06	32.45	26.76	26.48	32.71	36.72

Table 32: Hit Ratio and Byte Hit Ratio in Worse Results

Cache Size	Parent Cache		Child1 Cache		Child2 Cache	
	Hit Ratio	Byte Hit Ratio	Hit Ratio	Byte Hit Ratio	Hit Ratio	Byte Hit Ratio
1 Mb	25.00	48.36	42.86	24.61	66.67	56.12
6 Mb	36.14	23.22	31.88	35.34	36.96	36.20
500 Mb	7.55	5.61	0.00	0.00	11.39	15.34
800 Mb	7.55	5.14	3.33	3.01	0.00	0.00
1 Gb	7.25	5.12	3.33	5.60	20.15	19.30