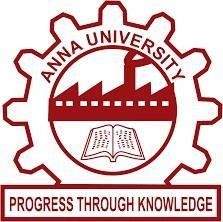
**A DATA CACHING METHODOLOGY IN EDGE COMPUTING**

# A PROJECT REPORT

***Submitted by***

|  |  |
| --- | --- |
| **NAVEEN KUMAR M** | **(1813047)** |
| **RAJESH KUMAR D** | **(1813056)** |
| **VERJIN SANTHIYA G** | **(1813078)** |

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**K.S.R. COLLEGE OF ENGINEERING**

**(An Autonomous Institution, Affiliated to Anna University Chennai and Approved by AICTE)**

**TIRUCHENGODE-637215**

# JUNE 2022

**K.S.R.COLLEGE OF ENGINEERING TIRUCHENGODE-637215**

**ANNA UNIVERSITY, CHENNAI**

**BONAFIDE CERTIFICATE**

Certified that this project report **“A DATA CACHING METHODOLOGY IN EDGE COMPUTING”** is the bonafide work of “**NAVEENKUMAR. M (1813047)”,** “**RAJESH KUMAR. D (1813056)”,** “**VERJIN SANTHIYA. G (1813078)”,** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**DR.A.RAJIVKANNAN M.E., Ph.D., MS.R.KEERTHANA M.E.,**

**HEAD OF THE DEPARTMENT SUPERVISOR**

**Professor & HoD Assistant Professor**

Department of CSE, Department of CSE,

K.S.R. College of Engineering K.S.R College of Engineering

Tiruchengode-637215. Tiruchengode-637215.

Submitted for Project viva-voce held on

Internal Examiner External Examiner

# ACKNOWLEDGEMENT

We feel highly honored to extend our sincere gratitude to our beloved Founder cum Chairman **Lion Dr. K. S. RANGASAMY MJF.,** K.S.R. Educational Institutions and our Chairman **Mr. R. SRINIVASAN BBM., MISTE.,** Aarthi Educational and charitable Trust for providing all facilities to complete this project work.

We would like to acknowledge the constant and kind support provided by our principal **Dr. P. SENTHILKUMAR M.E., Ph.D., (IITM),** who supported us in all the endeavors and been responsible for inculcating us all through our career.

We feel highly elated to thank our respectable Head of the Department

**Dr. A. RAJIV KANNAN M.E., Ph.D., MISTE., MCSI.,** who guided us and

was a pillar of support for the successful completion of the project.

We are thankful to our Project Coordinators **Dr. M. SOMU M.E., Ph.D.,** of our department for their valuable suggestions and guidance to our project.

We are the most fortunate in having the opportunity to work under the guide **MS. R. KEERTHANA M.E.,** express our sincere thanks to her. This project brought out the hidden talent within us.

It is a pleasure to express our gratefulness to our beloved parents for providing their support and confidence to us for the completion of the project and our heartfelt thanks to our entire department faculty members, beloved friends, directly and indirectly who helped us during the tenure of the project.

**ABSTRACT**

In the edge computing (EC) environment, edge servers are deployed at base stations to offer highly accessible computing and storage resources to nearby users. From the vendor's perspective, caching data on edge servers can ensure low latency in app users' retrieval of app data. However, an edge server normally owns limited resources due to its limited size. In this project, we investigate the collaborative caching problem in the EC environment with the aim to minimize the system cost including data caching cost, data migration cost, and quality-of-service (QoS) penalty. A constrained optimization problem and prove that it is caching complete. We propose a multi displinary algorithm such as linear algorithm and greedy algorithm, which will be used for collaborative data caching with the data cache on edge server. Data migration cost along with the data cache cost with the QOS caching can also be shown.

# TABLE OF CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |  |
|  |  |  |  |
|  | **ABSTRACT** | **Iv** |  |
|  | **LIST OF TABLES** | **X** |  |
|  | **LIST OF FIGURES** | **Xi** |  |
|  | **LIST OF ABBREVIATION** | **Xii** |  |
| **1** | **INTRODUCTION** | **1** |  |
|  | 1.1 EDGE COMPUTING | 1 |  |
|  | 1.2 APPLICATION | 2 |  |
|  | 1.3 USES OF EDGE COMPUTING | 3 |  |
|  | 1.4 HORIZONTALLY SCALABLE | 5 |  |
|  | 1.5 ONLINE ALGORITHM | 6 |  |
| **2** | **LITERATURE REVIEW** | **8** |  |

|  |  |  |
| --- | --- | --- |
|  | 2.1   THE FOUNDATION OF THE MOBILE AND WIRELESS COMMUNICATIONS SYSTEM FOR 2020 AND BEYOND | 8 |
|  | 2.2   OPTIMAL EDGE USER ALLOCATION IN EDGE COMPUTING WITH VARIABLE SIZED VECTOR BIN PACKING | 10 |
|  | 2.3    COMPUTATION PEER OFFLOADING FOR ENERGY-CONTRAINED MOBLIE EDGE COMPUTING IN SMALL-CELL NETWORKS LIXING | 12 |
|  | 2.4   COMPUTING OFFLOADING AND CELLULAR NETWORKS WITH MOBILE EDGE COMPUTING. | 14 |
|  | 2.5   ENERGY-EFFICIENT RESOURCE ALLOCATION FOR MOBILE-EDGE COMPUTATION OFFLOADING | 15 |
|  | 2.6   EFFICIENT MULTI-USER COMPUTATION OFFLOADING FOR MOBILE-EDGE CLOUD COMPUTING | 18 |
|  | 2.7    A GAME-THEORETICAL APPROACH FOR USER ALLOCATION IN EDGE COMPUTING ENVIRONMENT | 21 |
|  | 2.8   A SURVEY OF GENERAL-PURPOSE COMPUTATION ON GRAPHICS HARDWARE | 22 |
|  | 2.9    DISK CACHE PERFORMANCE FOR DISTRIBUTED SYSTEMS | 25 |
|  | 2.10   A SURVEY ON REPLACEMENT STRATEGIES IN CACHE MEMORY FOR EMBEDDED SYSTEMS | 28 |

1. **SOFTWARE DESCRIPTION** 32
   1. [FEATURES OF JAVA 32](#_TOC_250010)
   2. [SOCKET OVERVIEW 33](#_TOC_250009)
   3. [CLIENT/SERVER 33](#_TOC_250008)
   4. [RESERVED SOCKETS 34](#_TOC_250007)
   5. [JAVA AND THE NET 34](#_TOC_250006)
   6. [INETADDRESS 34](#_TOC_250005)
   7. [FACTORY METHODS 34](#_TOC_250004)
   8. [INSTANCE METHODS 35](#_TOC_250003)
   9. [TCP/IP CLIENT SOCKETS 36](#_TOC_250002)
   10. [TCP/IP SERVER SOCKETS 37](#_TOC_250001)
   11. [URL 38](#_TOC_250000)
2. MATERIALS AND METHODS 39
   1. EXISTING SYSTEM 39
   2. DRAWBACKS OF EXISTING SYSTEM 39
   3. PROPOSED METHOD 39
   4. ADVANTAGES OF PROPOSED MODEL 40
   5. INPUT AND OUTPUT 40
   6. MODULES DESCRIPTION 40
      1. Content Server 40
      2. User 40
      3. Caching And Multicasting 41
      4. Macro Cell Base Station 41
      5. Linear Relaxation And 41

Greedy Model

* 1. SYSTEM REQUIREMENTS 41
     1. Hardware Requirements 41
     2. Software Requirements 42
  2. FEASIBILITY STUDY 42
     1. Technical Feasibility 42
     2. Operational Feasibility 43
     3. Economic Feasibility 43
  3. TESTING AND IMPLEMENTATION 43
     1. System Testing 43
     2. Unit Testing 44
     3. Integration Testing 44
     4. Validation Testing 44
     5. Acceptance Testing 44

1. EXPERIMENTAL SETUP 46
2. RESULTS AND OUTPUT 47
3. CONCLUSION 54

# LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO.** | **TITLE** | **PAGE NO.** |
| **5.1** | EXPERIMENTAL SETUP | 46 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
| **5.1** | EXPERIMENTAL SETUP | 46 |
| **6.1** | CONTENT SERVER | 47 |
| **6.2** | USER | 48 |
| **6.3** | CACHING AND MULTICASTING | 49 |
| **6.4** | MACRO CELL BASE STATION | 50 |
| **6.5** | QOS PENALTY | 51 |
| **6.6** | DATA CACHE COST | 52 |
| **6.7** | DATA MIGRATION COST | 53 |

# LIST OF ABBREVATIONS

**ACRONYMS ABBREVIATIONS**

|  |  |
| --- | --- |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| UDP | User Datagram Protocol |
| CEDC | Collaborative Edge Data Caching |
| GUI | Graphical User Interface |
| API | Application Programming Interface |
| LAN | Local Area Network |
| IEEE | Institute Of Electrical And Electronics Engineers |
| EUA | Edge User Allocation |
| MEC | Mobile Edge Computing |
| ADMM | Alternating Direction Method Of Multipliers |
| OFDMA | Orthogonal Frequency-Division Multiple Access |
| GPU | Graphics Processing Unit |
| DBMS | Database Management System |

# CHAPTER 1

**INTRODUCTION**

# EDGE COMPUTING

Edge computing is a [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing) paradigm that brings [computation](https://en.wikipedia.org/wiki/Computation) and [data](https://en.wikipedia.org/wiki/Data_storage) [storage](https://en.wikipedia.org/wiki/Data_storage) closer to the sources of data. This is expected to improve response times and save [bandwidth.](https://en.wikipedia.org/wiki/Bandwidth_(computing)) "A common misconception is that edge and [IoT](https://en.wikipedia.org/wiki/Internet_of_things) are synonymous. Edge computing is a topology- and location-sensitive form of distributed computing, while IoT is a usecase instantiation of edge computing. The term refers to an architecture rather than a specific technology. The origins of edge computing lie in [content distributed networks](https://en.wikipedia.org/wiki/Content_delivery_network) that were created in the late 1990s to serve web and video content from edge [servers](https://en.wikipedia.org/wiki/Server_(computing)) that were deployed close to users. In the early 2000s, these networks evolved to host applications and application components at the edge servers, resulting in the first commercial edge computing services that hosted applications such as dealer locators, shopping carts, real-time data aggregators, and ad insertion engines.

One definition of edge computing is any type of [computer program](https://en.wikipedia.org/wiki/Computer_program) that delivers low latency [nearer to the requests](https://en.wikipedia.org/wiki/Locality_of_reference). Karim Arabi, in an IEEE DAC 2014 Keynote and subsequently in an invited talk at MIT's MTL Seminar in 2015, defined edge computing broadly as all computing outside the cloud happening at the edge of the network, and more specifically in applications where real-time processing of data is required. In his definition, [cloud](https://en.wikipedia.org/wiki/Cloud_computing) [computing](https://en.wikipedia.org/wiki/Cloud_computing) operates on [big data](https://en.wikipedia.org/wiki/Big_data) while edge computing operates on "instant data" that is real-time data generated by sensors or users. The term is often used synonymously with fog computing.

# APPLICATION

Edge nodes used for game streaming are known as gamelets, which are usually one or two hops away from the client. Per Anand and Edwin say "the edge node is mostly one or two hops away from the mobile client to meet the response time constraints for real-time games' in the [cloud gaming](https://en.wikipedia.org/wiki/Cloud_gaming) context."

Edge computing may employ [virtualization](https://en.wikipedia.org/wiki/Virtualization) technology to make it easier to deploy and run a wide range of applications on edge servers The world's data is expected to grow 61% to 175 zettabytes by 2025. According to research firm, Gartner, around 10% of enterprise-generated data is created and processed outside a traditional centralized data center or cloud. By 2025, the firm predicts that this figure will reach 75%. The increase of [IoT](https://en.wikipedia.org/wiki/Internet_of_things) devices at the edge of the network is producing a massive amount of data - storing and using all that data in cloud [data centers](https://en.wikipedia.org/wiki/Data_centers) pushes network bandwidth requirements to the limit. Despite the improvements of [network](https://en.wikipedia.org/wiki/Telecommunications_network) technology, data centers cannot guarantee acceptable transfer rates and response times, which, however, often is a critical requirement for many applications. Furthermore, devices at the edge constantly consume data coming from the cloud, forcing companies to decentralize data storage and service provisioning, leveraging physical proximity to the end user.

In a similar way, the aim of edge computing is to move the computation away from data centers towards the edge of the network, exploiting [smart](https://en.wikipedia.org/wiki/Smart_objects) [objects](https://en.wikipedia.org/wiki/Smart_objects), [mobile phones](https://en.wikipedia.org/wiki/Smartphone), or [network gateways](https://en.wikipedia.org/wiki/Gateway_(telecommunications)) to perform tasks and provide services on behalf of the cloud. By moving [services](https://en.wikipedia.org/wiki/Service_(systems_architecture)) to the edge, it is possible to provide content [caching](https://en.wikipedia.org/wiki/Cache_(computing)), service delivery, persistent data storage, and IoT management resulting in better response times and transfer rates. At the same time, distributing the logic to different network nodes introduces new issues and challenges.

# USES OF EDGE COMPUTING

Additionally, the usage of edge computing as an intermediate stage between client devices and the wider internet results in efficiency savings that can be demonstrated in the following example: A client device requires computationally intensive processing on video files to be performed on external servers. By using servers located on a local edge network to perform those computations, the video files only need to be transmitted in the local network. Avoiding transmission over the internet results in significant bandwidth savings and therefore increases efficiency. Another example is [voice recognition](https://en.wikipedia.org/wiki/Speech_recognition). If the recognition is performed locally, it is possible to send the recognized text to the cloud rather than audio recordings, significantly reducing the amount of required bandwidth.

Edge application services reduce the volumes of data that must be moved, the consequent traffic, and the distance that data must travel. That provides lower latency and reduces transmission costs. [Computation offloading](https://en.wikipedia.org/wiki/Computation_offloading) for real-time applications, such as facial recognition algorithms, showed considerable improvements in response times, as demonstrated in early research. Further research showed that using resource-rich machines called [cloudlets](https://en.wikipedia.org/wiki/Cloudlet) near mobile users, which offer services typically found in the cloud, provided improvements in execution time when some of the tasks are offloaded to the edge node. On the other hand, offloading every task may result in a slowdown due to transfer times between device and nodes, so depending on the workload, an optimal configuration can be defined.

Another use of the architecture is cloud gaming, where some aspects of a game could run in the cloud, while the rendered video is transferred to lightweight clients running on devices such as mobile phones, VR glasses, etc. This type of streaming is also known as pixel streaming.

Other notable applications include [connected cars](https://en.wikipedia.org/wiki/Connected_car), [autonomous cars](https://en.wikipedia.org/wiki/Autonomous_car), [smart](https://en.wikipedia.org/wiki/Smart_city) [cities](https://en.wikipedia.org/wiki/Smart_city), [Industry 4.0](https://en.wikipedia.org/wiki/Industry_4.0) (smart industry), and [home automation](https://en.wikipedia.org/wiki/Home_automation) systems. Database caching is a process included in the design of computer applications which generate web pages on-demand (dynamically) by accessing backend databases. When these applications are deployed on multi-tier environments that involve browser-based clients, web application servers and backend databases, middle-tier database caching is used to achieve high scalability and performance.

In a [three tier architecture](https://en.wikipedia.org/wiki/Three_tier_architecture), the [application software](https://en.wikipedia.org/wiki/Application_software) tier and [data](https://en.wikipedia.org/wiki/Computer_data_storage) [storage](https://en.wikipedia.org/wiki/Computer_data_storage) tier can be in different hosts. Throughput of an application can be limited by the [network](https://en.wikipedia.org/wiki/Computer_network) speed. This limitation can be minimized by having the [database](https://en.wikipedia.org/wiki/Database) at the application tier. Because commercial database software makes extensive use of system resources, it is not always practical to have the application and the [database](https://en.wikipedia.org/wiki/Database) at the same host. In this case, a more light-weight database application can be used to cache data from the commercial [database management system](https://en.wikipedia.org/wiki/Database_management_system). Database caching improves scalability by distributing query workload from backend to multiple cheap front-end systems. It allows flexibility in the processing of data; for example, the data of Platinum customers can be cached while that of ordinary customers are not. Caching can improve availability of data, by providing continued service for applications that depend only on cached tables even if the backend server is unavailable. Another benefit is improved data access speeds brought about by locality of data and smoothing out load peaks by avoiding round-trips between middle-tier and data-tier. Updateable cache tables: Many cache systems are read- only which limits their usage to small segment of the applications, non-real time applications. Bi-Directional updates: For updateable caches, updates, which happen in cache, should be propagated to the target database and any updates that happen directly on the target database should come to cache automatically. Synchronous and asynchronous update propagation: The updates on cache table shall be propagated to target database in two modes. Synchronous mode makes sure that after the database operation completes the updates are applied at the target database as well. In case of Asynchronous mode the updates are delayed to the target database. Synchronous mode gives high cache consistency and is suited for real time applications. Asynchronous mode gives high throughput and is suited for near real time applications.

Edge application services reduce the volumes of data that must be moved, the consequent traffic, and the distance that data must travel. That provides lower latency and reduces transmission costs. [Computation offloading](https://en.wikipedia.org/wiki/Computation_offloading) for real-time applications, such as facial recognition algorithms, showed considerable improvements in response times, as demonstrated in early research. Further research showed that using resource-rich machines called [cloudlets](https://en.wikipedia.org/wiki/Cloudlet) near mobile users, which offer services typically found in the cloud, provided improvements in execution time when some of the tasks are offloaded to the edge node. On the other hand, offloading every task may result in a slowdown due to transfer times between device and nodes, so depending on the workload, an optimal configuration can be defined.

Multiple cache granularity - Database level, Table level and Result-set caching: Major portions of corporate databases are historical and infrequently accessed. But, there is some information that should be instantly Recovery for cached tables: In case of system or power failure, during the restart of caching platform all the committed transactions on the cached tables should be recovered.

Tools to validate the coherence of cache: In case of asynchronous mode of update propagation, cache at different cache nodes and target database may diverge. This needs to be resolved manually, with mismatches identified and corrective measures taken if required.

# HORIZONTALLY SCALABLE

[Cluster computing](https://en.wikipedia.org/wiki/Cluster_computing) may increase availability and achieve load balancing. Caching in a clustered environment spans multiple nodes, keeping the cached data coherent across nodes. Transparent access to non-cached tables reside in target database: Database cache should keep track of queries and should be able to intelligently route to the database cache or to the origin database based on the data locality without any [application code](https://en.wikipedia.org/wiki/Application_code) modification.

Transparent Fail over: There should not be any service outages in case of caching platform failure. Client connections should be routed to the target database. No or very few changes to application: Support for standard interfaces JDBC, ODB etc. that will make the application to work seamlessly without any application code changes. It should route all stored procedure calls to target database so that they don't need to be migrated.

Pros: Horizontal scaling is much easier to accomplish without downtime. Horizontal scaling is also easier than vertical scaling to manage automatically. Limiting the number of requests any instance gets at one time is good for performance, no matter how large the instance. Provisioning additional instances also means having greater redundancy in the rare event of an outage.

Cons: Depending on the number of instances you need, your costs may be higher. Additionally, without a load balancer in place, your machines run the risk of being over-utilized, which could lead to an outage. However, with public cloud platforms, you can pay attention to discounts for Reserved Instances (RIs) if you’re able to predict when you require more compute power. Following cloud cost management best practices can help you efficiently scale in or out.

# ONLINE ALGORITHM

In [computer science](https://en.wikipedia.org/wiki/Computer_science), an online algorithm is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the [algorithm](https://en.wikipedia.org/wiki/Algorithm). In contrast, an offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand. In [operations research](https://en.wikipedia.org/wiki/Operations_research), the area in which online algorithms are developed is called [online optimization](https://en.wikipedia.org/wiki/Online_optimization). As an example, consider the [sorting algorithms selection](https://en.wikipedia.org/wiki/Sorting_algorithms) [sort](https://en.wikipedia.org/wiki/Selection_sort) and [insertion sort](https://en.wikipedia.org/wiki/Insertion_sort): selection sort repeatedly selects the minimum element from the unsorted remainder and places it at the front, which requires access to the entire input; it is thus an offline algorithm. On the other hand, insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus insertion sort is an online algorithm.

Note that the final result of an insertion sort is optimum, i.e., a correctly sorted list. For many problems, online algorithms cannot match the performance of offline algorithms. If the ratio between the performance of an online algorithm and an optimal offline algorithm is bounded, the online algorithm is called [competitive](https://en.wikipedia.org/wiki/Competitive_analysis_(online_algorithm)). Not every offline algorithm has an efficient online counterpart. Because it does not know the whole input, an online algorithm is forced to make decisions that may later turn out not to be optimal, and the study of online algorithms has focused on the quality of decision-making that is possible in this setting. [Competitive](https://en.wikipedia.org/wiki/Competitive_analysis_(online_algorithm)) [analysis](https://en.wikipedia.org/wiki/Competitive_analysis_(online_algorithm)) formalizes this idea by comparing the relative performance of an online and offline algorithm for the same problem instance. Specifically, the competitive ratio of an algorithm, is defined as the worst-case ratio of its cost divided by the optimal cost, over all possible inputs. The competitive ratio of an online problem is the best competitive ratio achieved by an online algorithm. Intuitively, the competitive ratio of an algorithm gives a measure on the quality of solutions produced by this algorithm, while the competitive ratio of a problem shows the importance of knowing the future for this problem. A problem exemplifying the concepts of online algorithms is the [Canadian Traveler Problem](https://en.wikipedia.org/wiki/Canadian_Traveller_Problem). The goal of this problem is to minimize the cost of reaching a target in a weighted graph where some of the edges are unreliable and may have been removed from the graph. However, that an edge has been removed (failed) is only revealed to the traveler when she/he reaches one of the edge's endpoints. The worst case for this problem is simply that all of the unreliable edges fail and the problem reduces to the usual [Shortest Path Problem](https://en.wikipedia.org/wiki/Shortest_path_problem). An alternative analysis of the problem can be made with the help of competitive analysis. For this method of analysis, the offline algorithm knows in advance which edges will fail and the goal is to minimize the ratio between the online and offline algorithms' performance. This problem is [PSPACE-complete](https://en.wikipedia.org/wiki/PSPACE-complete). An online algorithm is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the beginning.

In contrast, an offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand. As an example, consider the sorting algorithms [selection sor](https://www.geeksforgeeks.org/selection-sort/)t and [insertion sort](https://www.geeksforgeeks.org/insertion-sort/):

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. which requires access to the entire input; it is thus an offline algorithm. On the other hand, insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus insertion sort is an online algorithm.

# CHAPTER 2 LITERATURE REVIEW

# THE FOUNDATION OF THE MOBILE AND WIRELESS COMMUNICATIONS SYSTEM FOR 2020 AND BEYOND

**Afif Osseiran ET.AL.,** has proposed. In this paper In 2020, mobile and wireless traffic volume is expected to increase thousand-fold over 2010 figures. Moreover, an increase in the number of wirelessly-connected devices to counts in the tens of billions will have a profound impact on society. Massive machine communication, forming the basis for the Internet of Things, will make our everyday life more efficient, comfortable and safer, through a wide range of applications including traffic safety and medical services. The variety of applications and data traffic types will be significantly larger than today, and will result in more diverse requirements on services, devices and networks. METIS is set up by leading global players to prepare the migration towards tomorrow's multipurpose global communication infrastructure, serving humans and things. The main objective of METIS is to lay the foundation for this future global mobile and wireless communications system, and to generate a global consensus here. In particular, METIS will provide new solutions which fit the needs beyond 2020. The coexistenceof human-centric and machine-type applications will lead to a large diversity of communication characteristics imposing different requirements on mobile and wireless communication systems, e.g. in terms of cost, complexity, energy dissipation, and service requirements. As a particular example, it is expected that the uptake of machine communications in many fields such as healthcare, security, logistics, automotive applications etc. will pose far higher reliability requirements on connectivity than communication does today. In terms of quality of service, a future mobile and wireless communication system must support a wide range of user data rates. For example, it must provide multi-Gbps data rates in specific scenarios, be able to guarantee tens of Mbps data rates with very high availability and reliability, and also be able to provide low data rates for machine-type communication in a cost-efficient and energy-efficient manner. Societal development will lead to changes in the way mobile and wireless communication systems are used. Essential services such as e-banking, e-learning and e-health will continue to proliferate and become more mobile. On demand information and entertainment will be delivered over mobile and wireless communication systems. The wirelessly connected Internet of Things will make our everyday life more efficient, comfortable and safer. As shown in, the societal development will lead to an avalanche of mobile and wireless traffic volume, predicted to increase a thousand- fold over the next decade. More precisely, in the already highly developed communication societies of Western Europe there are predictions that the traffic in cellular systems will increase by roughly a factor of 70 until 2020 . Other forecasts by telecom players predict even more massive traffic growth, with forecasts in the order of 1000 times larger global wireless-traffic volumes in 2020 than seen in 2010, and primarily driven by increased usage of mobile multi-media services. Although the existing wide area mobile and wireless communication technologies are evolving, they have been primarily designed having human-centric mobile- broadband services and applications in mind. As a result of this, they are not able to respond sufficiently well to the diversity of application requirements and use cases foreseen for timeframes beyond 2020. They do also not provide the efficiency and scalability needed to respond to the expected explosion of traffic and number of connected devices. Mobile and wireless communication systems beyond 2020 will have to respond to the increase in traffic volume, by increasing capacity and by improving efficiency in energy, cost and spectrum utilization. Further, the numbers of devices and the variety of use cases and requirements will necessitate mobile and wireless communication solutions with significantly increased versatility and improved scalability. METIS will lay the foundation for the next generation mobile and wireless system. METIS will develop a system concept that delivers the necessary efficiency, versatility and scalability. The project will research technology components such as network topologies, radio links, multi-node, and spectrum usage techniques. Horizontal topics such as Device-to-Device, Massive Machine Communications, Moving Networks and Ultra-dense Networks will be used to integrate the research results into a system concept that provides the necessary flexibility, versatility and scalability at a low cost.

Mobile and wireless traffic volume is expected to increase thousand-fold over 2010 figures. Moreover, an increase in the number of wirelessly-connected devices to counts in the tens of billions will have a profound impact on society. Massive machine communication, forming the basis for the Internet of Things, will make our everyday life more efficient, comfortable and safer, through a wide range of applications including traffic safety and medical services. The variety of applications and data traffic types will be significantly larger than today, and will result in more diverse requirements on services, devices and networks.

METIS is set up by leading global players to prepare the migration towards tomorrow's multipurpose global communication infrastructure, serving humans and things. The main objective of METIS is to lay the foundation for this future global mobile and wireless communications system, and to generate a global consensus here. In particular, METIS will provide new solutions which fit the needs beyond 2020.

# OPTIMAL EDGE USER ALLOCATION IN EDGE COMPUTING WITH VARIABLE SIZED VECTOR BIN PACKING

**Phu Lai** et.al., has proposed. In this paper In mobile edge computing, edge servers are geographically distributed around base stations placed near end- users to provide highly accessible and efficient computing capacities and services. In the mobile edge computing environment, a service provider can deploy its service on hired edge servers to reduce end-to-end service delays experienced by its end- users allocated to those edge servers. An optimal deployment must maximize the number of allocated end-users and minimize the number of hired edge servers while ensuring the required quality of service for endusers. In this paper, we model the edge user allocation (EUA) problem as a bin packing problem, and introduce a novel, optimal approach to solving the EUA problem based on the Lexicographic Goal Programming technique. We have conducted three series of experiments to evaluate the proposed approach against two representative baseline approaches. Experimental results show that our approach significantly outperforms the other two approaches. Edge computing is a promising new computing architecture, especially for high volume, data processing-intensive, latency-sensitive applications and services. However, when an edge computing scenario scales up, an ineffective edge user allocation solution will greatly increase the operational costs for service providers. To address this problem, we formulated the edge user allocation (EUA) problem as a variant of the bin packing problem named variable sized vector bin packing, an N P-hard problem. We solved this problem using a Lexicographic Goal Programming technique with two optimization objectives, i.e., to maximize the number of users allocated and minimize the number of edge servers hired. We then conducted extensive experiments in scenarios with various service deployment requirements. Our experimental results show that our approach significantly outperforms two baseline approaches, greedy and random. 244 P. Lai et al. It is capable of allocating the most end-users with significantly fewer edge servers – nearly three times less than the greedy method – as the EUA problem scales up. This research has established a basic foundation for the EUA problem and opened up a number of research directions. In our future work, we will take into account the users’ mobility as well as the dynamics of users’ computation tasks. In addition, apart from the proximity and capacity constraints, there are several elements that also play an important role such as network latency, service availability, pricing, and security. To the best of our knowledge, our work is the first to tackle the EUA problem in scenarios with multiple edge servers and end-users that possess and require multi- dimensional computing capacities. We also realistically and innovatively address this problem with respect to proximity constraints with the aims to maximize the number of allocated users and minimize the number of hired servers. Threats to Conclusion Validity.

The lack of statistical tests is the biggest threat to our conclusion validity. Statistical tests will be included in our future work to prove a statistically significant relationship between the experiment settings and the results. In this paper, we have compensated for this with meaningful comparison baselines and extensive experiments that cover many different scenarios, varying in both size and complexity. When an experimental parameter changes, the results are averaged over 100 runs of the experiment.

In the mobile edge computing environment, a service provider can deploy its service on hired edge servers to reduce end-to-end service delays experienced by its end-users allocated to those edge servers. An optimal deployment must maximize the number of allocated end-users and minimize the number of hired edge servers while ensuring the required quality of service for end-users.

# COMPUTATION PEER OFFLOADING FOR ENERGY- CONSTRAINED MOBILE EDGE COMPUTING IN SMALL-CELL NETWORKS LIXING

**Chen et.al.,** has proposed. In this paper The (ultra-)dense deployment of small- cell base stations (SBSs) endowed with cloud-like computing functionalities paves the way for pervasive mobile edge computing (MEC), enabling ultra-low latency and location-awareness for a variety of emerging mobile applications and the Internet of Things. To handle spatially uneven computation workloads in the network, cooperation among SBSs via workload peer offloading is essential to avoid large computation latency at overloaded SBSs and provide high quality of service to end users. However, performing effective peer offloading faces many unique challenges due to limited energy resources committed by self-interested SBS owners, uncertainties in the system dynamics and co-provisioning of radio access and computing services. This paper develops a novel online SBS peer offloading framework, called OPEN, by leveraging the Lyapunov technique, in order to maximize the long-term system performance while keeping the energy consumption of SBSs below individual long-term constraints. OPEN works online without requiring information about future system dynamics, yet provides provably near-optimal performance compared to the oracle solution that has the complete future information. In addition, this paper formulates a peer offloading game among SBSs, analyzes its equilibrium and efficiency loss in terms of the price of anarchy to thoroughly understand SBSs’ strategic behaviors, thereby enabling decentralized and autonomous peer offloading decision making. Extensive simulations are carried out and show that peer offloading among SBSs dramatically improves the edge computing performance. In this paper, we investigated peer offloading schemes in MEC-enabled small cell networks where heterogeneous task arrival pattern in both spatial and temporal domains is considered. We developed OPEN, a novel online peer offloading framework

to optimize the edge computing performance under limited energy resources committed by individual SBSs without requiring information on future system dynamics. The proposed framework allows both centralized and autonomous decision making, and provide provable performance guarantee. We also showed that OPEN incurs acceptable overhead in practice. In the current system model, we considered a simple structure of LAN to model the congestion delay during peer offloading. How to extend our analysis to more sophisticated and practical congestion scenarios needs further investigation. Moreover, the performance guarantee of OPEN rests on the assumption that the observations of task arrival rates in the current slot 36 are precise, which may not hold for all network systems. Therefore, future efforts are needed to take into consideration of imprecise estimation of task arrival. Notice that the task arrival pattern in the real- world system may not follow the assumed Poisson process. To analyze the practicality of OPEN, we implement it with different task arrival realizations. the performances achieved by OPEN and NoP with two task arrival realizations: bursty arrival and non-

i.i.d. arrival. We can see from that OPEN and NoP both have a higher delay cost in the bursty arrival case compared to that in the Poisson 35 arrival case. This is because the tasks are more likely to queue up at edge servers when bursts occur. However, we see that the proposed algorithm still provides a 55.0% delay reduction with bursty arrival which is similar to that of Poisson arrival. In the non-i.i.d. case, we use a Markov process to model a task arrival pattern where the intervals of task arrivals are determined by certain transition probabilities. We see from Fig. 14(b) that the delay reduction achieved by OPEN in the non-i.i.d case slightly decreases compared to that in the Poisson arrival case. However, applying OPEN still offers an obvious delay reduction, 37.5%, for the edge system. These two examples indicate that the proposed algorithm can offer considerable performance improvement for the edge system even if the real task arrival does not closely follow the Poisson process.[3]

# COMPUTATION OFFLOADING AND RESOURCE ALLOCATION IN WIRELESS CELLULAR NETWORKS WITH MOBILE EDGE COMPUTING

Chenmeng Wang et.al., has proposed. In this paper Mobile edge computing (MEC) has risen as a promising technology to augment computational capabilities of mobile devices. Meanwhile, in-network caching has become a natural trend of the solution of handling exponentially increasing Internet traffic. The important issues in these two networking paradigms are computation offloading and content caching strategies, respectively. In order to jointly tackle these issues in wireless cellular networks with mobile edge computing, we formulate computation offloading decision, resource allocation and content caching strategy as an optimization problem, considering the total revenue of the network. Furthermore, we transform the original problem into a convex problem and then decompose it in order to solve it in a distributed and efficient way. Finally, with recent advances in distributed convex optimization, we develop an alternating direction method of multipliers (ADMM) based algorithm to solve the optimization problem. The effectiveness of the proposed scheme is demonstrated by simulation results with different system parameters. The main reason is that when there are too many small cells in the system, all those UEs will cause severe interference to each other during the computation tasks offloading process, so the algorithm automatically declined some of the offloading requests generated by the UEs. Besides, the computation resource of the MEC server cannot be reused at the same time, so if there are too many offloading UEs, the amount of computation resource assigned to each UE will decrease, which implies that the MSO will gain less from a single UE. The centralized algorithm and our proposed ADMM based algorithm achieve relatively high revenue among all the six solutions. Again, because of the uniform allocation of spectrum and computation resources, the other three solutions in which uniform resource allocation strategies are adopted can just achieve much lower revenue. Similarly, due to lack of revenue from alleviated backhaul bandwidth, the revenue of ADMM solution without caching is also low. In Fig. 9, the revenue of ADMM is compared with those of the centralized algorithm and the other four baseline solutions. In this figure, the number of small cells is set as 20. It can be seen from Fig. 9 that the revenue of our proposed ADMM algorithm is close to the revenue achieved by the centralized algorithm under various spectrum bandwidth conditions. The solution of ADMM with uniform computation resource allocation but optimal spectrum allocation can achieve relatively higher revenue compared with the other two uniform solutions (ADMM with uniform spectrum allocation and uniform spectrum and computation resource allocation). This is mainly because of the fact that spectrum allocation usually plays a more important role in earning interest of MSO, and this in turn is due to the fact that unlike computation resource of MEC server, the spectrum resource could be reused simultaneously among UEs in different small cells. In this paper, we presented an ADMM-based decentralized algorithm for computation offloading, resource allocation and Internet content caching optimization in heterogeneous wireless cellular networks with mobile edge computing. We formulated the computation offloading decision, spectrum resource allocation, MEC allocation, and content caching issues as an optimization problem. Then, in order to tackle this problem in an efficient way, we presented an ADMM-based distributed solution, followed by a discussion about the feasibility and complexity of the algorithm. Finally, the performance evaluation of the proposed scheme was presented in comparison with the centralized solution and several baseline solutions. Simulation results demonstrated that the proposed scheme can achieve better performance than other baseline solutions under various system parameters. Future work is in progress to consider wireless network virtualization in the proposed framework.[4]

# ENERGY-EFFICIENT RESOURCE ALLOCATION FOR MOBILE-EDGE COMPUTATION OFFLOADING

Changsheng You et.al., has proposed. In this paper Mobile-edge computation offloading (MECO) offloads intensive mobile computation to clouds located at the edges of cellular networks. Thereby, MECO is envisioned as a promising technique for prolonging the battery lives and enhancing the computation capacities of mobiles. In this paper, we study resource allocation for a multiuser MECO system based on time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA). First, for the TDMA MECO system with infinite or finite cloud computation capacity, the optimal resource allocation is formulated as a convex optimization problem for minimizing the weighted sum mobile energy consumption under the constraint on computation latency. The optimal policy is proved to have a threshold-based structure with respect to a derived offloading priority function, which yields priorities for users according to their channel gains and local computing energy consumption. As a result, users with priorities above and below a given threshold perform complete and minimum offloading, respectively. Moreover, for the cloud with finite capacity, a sub-optimal resource-allocation algorithm is proposed to reduce the computation complexity for computing the threshold. Next, we consider the OFDMA MECO system, for which the optimal resource allocation is formulated as a mixed-integer problem. To solve this challenging problem and characterize its policy structure, a low-complexity sub-optimal algorithm is proposed by transforming the OFDMA problem to its TDMA counterpart. The corresponding resource allocation is derived by defining an average offloading priority function and shown to have close- to-optimal performance in simulation. Both the TDMA and OFDMA systems are considered as follows. For the TDMA system, time is divided into slots each with a duration of T seconds where T is chosen to meet the user-latency requirement. As shown in Fig. 1, each time slot comprises two sequential phases for 1) mobile offloading or local computing and 2) cloud computing and downloading of computation results from the edge cloud to mobiles. Cloud computing has small latency; the downloading consumes negligible mobile energy and furthermore is much faster than offloading due to relative smaller sizes of computation results. For these reasons, the second phase is assumed to have a negligible duration compared to the first phase and not considered in resource allocation. For the OFDMA system, the total bandwidth is divided into multiple orthogonal subchannels and each sub-channel can be assigned to at most one user. The offloading mobiles will be allocated with one or more sub-channels. Considering an arbitrary slot in TDMA/OFDMA, the BS schedules a subset of users for complete/partial offloading. The user with partial or no offloading computes a fraction of or all input data, respectively, using a local CPU. This work studies resource allocation for a multiuser MECO system based on TDMA/OFDMA, accounting for both the cases of infinite and finite cloud computation capacities. For the TDMA MECO system, it shows that to minimize weighted sum mobile energy consumption, the optimal resource allocation policy should have a threshold-based structure. Specifically, we derive an offloading priority function that depends on the local computing energy and channel gains. Based on this, the BS makes a binary offloading decision for each mobile, where users with priorities above and below a given threshold will perform complete and minimum offloading. Then, we extend this threshold-based policy structure to the OFDMA system and design a low-complexity algorithm to solve the formulated mixed-integer optimization problem, which has close-to-optimal performance in simulation. This paper considers resource allocation in a multiuser MECO system based on TDMA and OFDMA. Multiple mobiles are required to compute different computation loads with the same latency constraint. Assuming that computation data can be split for separate computing, each mobile can simultaneously perform local computing and offloading. Moreover, the edge cloud is assumed to have perfect knowledge of local computing energy consumption, channel gains and fairness factors at all users, which is used for designing centralized resource allocation to achieve the minimum weighted sum mobile energy consumption. In the TDMA MECO system, the optimal threshold-based policy is derived for both the cases of infinite and finite cloud capacities. For the OFDMA MECO system, a low-complexity sub-optimal algorithm is proposed to solve the mixed-integer resource allocation problem. The contributions of current work are as follows. • TDMA MECO with infinite cloud capacity: For TDMA MECO with infinite (computation) capacity, a convex optimization problem is formulated to minimize the weighted sum mobile energy consumption under the time-sharing constraint. To solve it, an offloading priority function is derived that yields priorities for users and depends on their channel gains and local computing energy consumption. Based on this, the optimal policy is proved to have a threshold-based structure that determines complete and minimum offloading for users with priorities above and below a given threshold, respectively. • TDMA MECO with finite cloud capacity: The above results are extended to the case of finite capacity. Specifically, the optimal resource allocation policy is derived by defining an effective offloading priority function and modifying the threshold-based policy as derived for the infinite-capacity cloud. To reduce the complexity arising from a two-dimension search for Lagrange multipliers, a simple and low-complexity algorithm is proposed based on the approximated offloading priority order. This reduces the said search to a one- dimension search, shown by simulation to have close-to-optimal performance.[5]

# EFFICIENT MULTI-USER COMPUTATION OFFLOADING FOR MOBILE-EDGECLOUD COMPUTING

**Xu Chen et.al.,** has proposed. In this paper Mobile-edge cloud computing is a new paradigm to provide cloud computing capabilities at the edge of pervasive radio access networks in close proximity to mobile users. In this paper, we first study the multi-user computation offloading problem for mobile-edge cloud computing in a multi-channel wireless interference environment. We show that it is NP-hard to compute a centralized optimal solution, and hence adopt a game theoretic approach for achieving efficient computation offloading in a distributed manner. We formulate the distributed computation offloading decision making problem among mobil e device users as a multi-user computation offloading game. We analyze the structural property of the game and show that the game admits a Nash equilibrium and possesses the finite improvement property. We then design a distributed computation offloading algorithm that can achieve a Nash equilibrium, derive the upper bound of the convergence time, and quantify its efficiency ratio over the centralized optimal solutions in terms of two important performance metrics. We further extend our study to the scenario of multi-user computation offloading in the multi-channel wireless contention environment. Numerical results corroborate that the proposed algorithm can achieve superior computation offloading performance and scale well as the use r size increases. The concept of beneficial cloud computing plays an important role in the mobile-edge cloud computing. On the one hand, from the user’s perspective, beneficial cloud computing ensures the individual rationality, i.e., a mobile device user would not suffer performance loss by adopting the cloud computing approach. On the other hand, from the telecom operator’s point of view, the larger number of users achieving beneficial cloud computing implies a higher utilization ratio of the cloud resources and a higher revenue of providing mobile edge cloud computing service. Thus, different from traditional multi-user traffic scheduling problem, when determining the wireless access schedule for computation offloading, we need to ensure that for a user choosing cloud computing, that user must be a beneficial cloud computing user. Otherwise, the user will not follow the computation offloading schedule, since it can switch to the local computing approach to reduce the computation overhead. In this paper, we propose a game theoretic approach for the computation offloading decision making problem among multiple mobile device users for mobile-edge cloud computing. We formulate the problem as as a multi-user computation offloading game and show that the game always admits a Nash equilibrium. We also design a distributed computation offloading algorithm that can achieve a Nash equilibrium, derive the upper bound of convergence time, and quantify its price of anarchy. Numerical results demonstrate that the proposed algorithm achieves superior computation offloading performance and scales well as the user size increases. For the future work, we are going to consider the more general case that mobile users may depart and leave dynamically within a computation offloading period. In this case, the user mobility patterns will play an important role in the problem formulation. Another direction is to study the joint power control and offloading decision-making problem, which would be very interesting and technically challenging. Given the fact that base-stations in most wireless networks are operating in the multi-channel wireless environment, in this paper we study the generalized multi-user computation offloading problem in a multi-channel setting, which results in significant differences in analysis. For example, we show the generalized problem is NP-hard, which is not true for the single-channel case. We also investigate the price of anarchy in terms of two performance metrics and show that the number of available channels can also impact the price of anarchy (e.g., Theorem 5). We further derive the upper bound of the convergence time of the computation offloading algorithm in the multi-channel environment. Barbarossa et al. in [9] studied the multi-user computation offloading problem in a multi-channel wireless environment, by assuming that the number of wireless access channels is greater than the number of users such that each mobile user can offload the computation via a single orthogonal channel independently without experiencing any interference from other users. In this paper we consider the more practical case that the number of wireless access channels is limited and each user mobile may experience interference from other users for computation offloading.[6]

In mobile edge computing, edge servers are geographically distributed around base stations placed near end-users to provide highly accessible and efficient computing capacities and services. In the mobile edge computing environment, a service provider can deploy its service on hired edge servers to reduce end-to-end service delays experienced by its end-users allocated to those edge servers. An optimal deployment must maximize the number of allocated end-users and minimize the number of hired edge servers while ensuring the required quality of service for end-users.

In this paper, we model the edge user allocation (EUA) problem as a bin packing problem, and introduce a novel, optimal approach to solving the EUA problem based on the Lexicographic Goal Programming technique. We have conducted three series of experiments to evaluate the proposed approach against two representative baseline approaches. Experimental results show that our approach significantly outperforms the other two approaches.

Then, we propose an optimal approach named EAD- opt to find the optimal solution of EAD based on integer programming, and an approximation approach named EAD- apx to find approximate solutions in large-scale EAD scenarios efficiently. We evaluate our approaches by conducting experiments on a widely used real-world data set and a synthetic data set with comparison against two baseline approaches. The experimental results demonstrate that our approaches can solve the EAD problem effectively and efficiently.

# A GAME-THEORETICAL APPROACH FOR USER ALLOCATION IN EDGECOMPUTING ENVIRONMENT

**Qiang He et.al.,** has proposed. In this paper Edge Computing provides mobile and Internet-of-Things (IoT) app vendors with a new distributed computing paradigm which allows an app vendor to deploy its app at hired edge servers distributed near app users at the edge of the cloud. This way, app users can be allocated to hired edge servers nearby to minimize network latency and energy consumption. A cost-effective edge user allocation (EUA) requires maximum app users to be served with minimum overall system cost. Finding a centralized optimal solution to this EUA problem is NP-hard. Thus, we propose EUAGame, a game- theoretic approach that formulates the EUA problem as a potential game. We analyze the game and show that it admits a Nash equilibrium. Then, we design a novel decentralized algorithm for finding a Nash equilibrium in the game as a solution to the EUA problem. The performance of this algorithm is theoretically analyzed and experimentally evaluated. The results show that the EUA problem can be solved effectively and efficiently. This section presents EUAGame, our game- theoretic approach for solving the EUA problem. The reasons for the adoption of a game-theoretic approach are threefold. First, app users may have differentiated needs and pursue different interests. Game theory has been successfully employed in many fields as a powerful tool for analyzing the interactions among multiple players who act in their own interests. In the edge computing environment, it can be employed to devise incentive compatible EUA mechanisms for finding collectively satisfactory EUA solutions such that no app users have the incentive to deviate unilaterally. Second, by leveraging the intelligence of each individual app user, EUAGame attempts to solve the EUA problem in a distributed manner. It can ease the heavy burden of finding the centralized optimal solution. It will also scale with the size of the EUA problem, e.g., the number app users to allocate and the number

of available edge servers. Finally, compared with a centralized approach, a decentralized game-theoretic approach can find EUA solutions quickly. This fulfills app users as well as app vendors’ needs for low latency in the edge computing environment. This section theoretically and experimentally evaluates the performance of EUAGame in achieving app vendors’ two optimization objectives as discussed in Section 3.4: 1) maximize the number of alloc7 RELATED WORK One of the critical limitations of mobile and IoT devices is their limited computing capacities and energy capacities. This is also one of the key drivers that promote the advances in distributed computing paradigm in recent years, including cloud computing and edge computing. Both cloud computing and edge computing allow computation tasks to be offloaded from mobile and IoT devices to external servers deployed in the cloud or at the edge of the cloud. This way, the limitation of mobile and IoT devices’ computing capacities can be tackled. The research problem of how to effectively and efficiently offload computation tasks is referred to as computation offloading. In this paper, we proposed EUAGame, a novel gametheoretical approach for solving the Edge User Allocation (EUA) problem from app vendors’ perspective in the edge computing environment. We show that the EUA problem is NP-hard and model it as a potential game in which app users make their own allocation decisions. This way, the EUA problem can be solved in a distributed manner. We prove that the EUA game admits at least one Nash equilibrium and propose a decentralized allocation mechanism to achieve it. Its performance is evaluated both theoretically and experimentally. This research has established the basic foundation for the EUA problem and opened up many research directions. In our future work, we will first investigate the impacts of app users’ mobility and trajectories on EUAGame. Then, we will consider app users’ dynamic participation in the EUA game, including the arrivals of new app users and the departures of existing app users. We will also improve EUA Game to accommodate app users’ diverse capacities needs and

investigate the impact on user experience.[7]

# A SURVEY OF GENERAL-PURPOSE COMPUTATION ON GRAPHICS HARDWARE

**John D. Owens et.al.,** has proposed. In this system The rapid increase in the performance of graphics hardware, coupled with recent improvements in its program ability, have made graphics hardware a compelling platform for computationally demanding tasks in a wide variety of application domains. In this report, we describe, summarize, and analyze the latest research in mapping general- purpose computation to graphics hardware. We begin with the technical motivations that underlie general-purpose computation on graphics processors (GPGPU) and describe the hardware and software developments that have led to the recent interest in this field. We then aim the main body of this report at two separate audiences. First, we describe the techniques used in mapping general-purpose computation to graphics hardware. We believe these techniques will be generally useful for researchers who plan to develop the next generation of GPGPU algorithms and techniques. Second, we survey and categorize the latest developments in general- purpose application development on graphics hardware. This survey should be of particular interest to researchers who are interested in using the latest GPGPU applications in their systems of interest Commodity computer graphics chips are probably today’s most powerful computational hardware for the dollar. These chips, known generically as Graphics Processing Units or GPUs, have gone from afterthought peripherals to modern, powerful, and programmable processors in their own right. Many researchers and developers have become interested in harnessing the power of commodity graphics hardware for general-purpose computing. Recent years have seen an explosion in interest in such research efforts, known collectively as GPGPU (for “General Purpose GPU”) computing. In this State of the Art Report we summarize the motivation and essential developments in the hardware and software behind GPGPU. We give an overview of the techniques and computational

building blocks used to map general purpose computation to graphics hardware and provide a survey of the various general-purpose computing applications to which GPUs have been applied. The GPU is hardly a computational panacea. The arithmetic power of the GPU is a result of its highly specialized architecture, evolved over the years to extract the maximum performance on the highly parallel tasks of traditional computer graphics. The rapidly increasing flexibility of the graphics pipeline, coupled with some ingenious uses of that flexibility by GPGPU developers, has enabled a great many applications outside the original narrow tasks for which GPUs were originally designed, but many applications still exist for which GPUs are not (and likely never will be) well suited. Word processing, for example, is a classic example of a “pointer chasing” application, which is dominated by memory communication and difficult to parallelize. Today’s GPUs also lack some fundamental computing constructs, such as integer data operands. The lack of integers and associated operations such as bit-shifts and bitwise logical operations (AND, OR, XOR, NOT) makes GPUs illsuited for many computationally intense tasks such as cryptography. Finally, while the recent increase in precision to 32-bit floating point has enabled a host of GPGPU applications, 64-bit double precision arithmetic appears to be on the distant horizon at best. The lack of double precision hampers or prevents GPUs from being applicable to many very largescale computational science problems. The application domain of interactive 3D graphics has several characteristics that differentiate it from more general computation domains. In particular, interactive 3D graphics applications require high computation rates and exhibit substantial parallelism. Building custom hardware that takes advantage of the native parallelism in the application, then, allows higher performance on graphics applications than can be obtained on more traditional microprocessors. All of today’s commodity GPUs structure their graphics computation in a similar organization called the graphics pipeline. This pipeline is

designed to allow hardware implementations to maintain high computation rates through parallel execution. The pipeline is divided into several stages; all geometric primitives pass through every stage. In hardware, each stage is implemented as a separate piece of hardware on the GPU in what is termed a task-parallel machine organization. Figure 2 shows the pipeline stages in current GPUs. At a high level, radiosity works much like photon mapping when computing global illumination for diffuse surfaces. In a radiosity-based algorithm, energy is transferred around the scene much like photons are. Unlike photon mapping, the energy is not stored in a separate data structure that can be queried at a later time. Instead, the geometry in the scene is subdivided into patches or elements, and each patch stores the energy arriving on that patch. Database Management Systems (DBMSs) and data mining algorithms are an integral part of a wide variety of commercial applications such as online stock marketing and intrusion detection systems. Many of these applications analyze large volumes of online data and are highly computation and memory- intensive. As a result, researchers have been actively seeking new techniques and architectures to improve the query execution time. The high memory bandwidth and the parallel processing capabilities of the GPU can significantly accelerate the performance of many essential database queries such as conjunctive selections, aggregations, semilinear queries and join queries. These queries are described in Section 4.5. Govindaraju et al. compared the performance of SQL queries on an NVIDIA GeForce 6800 against a 2.8 GHz Intel Xeon processor. Preliminary comparisons indicate up to an order of magnitude improvement for the GPU over a SIMD-optimized CPU implementation.

* 1. **DISK CACHE PERFORMANCE FOR DISTRIBUTED SYSTEMS**

**Dwight J. Makaroff et.al.,** has proposed. In this paper Disk caching in fie-

server based distributed computer systems involves additional design decisions due to the presence of both workstation and file server caches. Different costs are involved in placing a given total amount of cache at the workstation clients or at the server, but much different benefits may be achieved. The influences on caching performance of client and server cache sizes, and the number of clients, are studied here through trace-driven simulation. The results indicate that the locality of reference in disk block reference panems allows relatively small caches to significantly reduce the number of disk accesses required. File server cache performance is significantly different than client cache performance due to the capture of disk block references by the client caches. The major factor influencing overall miss ratio statistics (actual disk reference frequencies) was found to be the maximum of the sewer cache size and the size of client caches. Two specific metrics that are commonly used to evaluate the effectiveness of a disk cache are the hit ratio and the miss ratio. The "hit ratio" is defined as the fraction of total disk block references which are satisfied by searching the cache. A reference that is satisfied in this manner is termed a cache hit. The "miss ratio" of a cache is defined as the fraction of total disk block references not satisfied by the cache (cache misses) and ultimately resulting in further search, either in another cache, or on the disk itself. In this paper, we consider miss ratios of client and server caches individually as well as "overall" miss ratios. Evaluating the effect of disk caching on the overall performance of a distributed system is complicated by the fact that a significant portion of the cost of disk U0 comes from the transmission of data across the network. Thus, cache hits at the client yield much greater performance improvements than cache hits at the server. Quantifying the influence of these costs requires data on average disk service times, cache access times, CPU and network costs for data transmission, and system load. This highly system dependent data can be used in conjunction with the results that will be presented here concerning miss ratios to evaluate disk caching performance in any particular system of interest. The source of measurement data used in this study was an essentially stand-alone SUN 2-120 workstation equipped with a 130 MByte disk and 5 M Bytes of main memory, running the SUN UNE' operating system. Since only data on the pattem of disk block read and write operations in a workstation environment was required, the fact that the disk was local rather than remotely located was largely irrelevant as far as the characteristics of the data collected were concemed. The use of a disked workstation to obtain data on the disk block reference pattems likely to be generated by a diskless workstation has the advantage that each 110 request can be easily obtained by appropriate "probes" inserted into the low level operating system routines responsible for disk block U0 reads and writes. If data was collected at a file server instead, the disk block requests that are satisfied by the client's cache would not be observed. Alliteratively, attempting to capture disk block requests at a diskless client would be very difficult, since operating system "probes" would have to be inserted at a variety of "higher-level" locations and since storage of large volumes of data would be difficult. The "real" workload represented by the traces contains extreme variations in the disk block reference activity. Thus, a number of simulation runs were performed with different selections from the available traces for each configuration simulated. The following performance measures were obtained from each set of runs, in addition to the individual client miss ratios: Combined Client Miss Ratio: the total number of cache misses at the clients (summed over all simulation runs for a given configuration), divided by the total number of disk block references generated by those clients (summed over all simulation runs for the configuration). Combined Server Miss Ratio: the total number of cache misses at the server (summed over all simulation runs for a given configuration), divided by the total number of disk block references seen by the server (summed over all simulation runs for the configuration). In a file-server based distributed system, the existence of both client workstation and file server disk caches complicates both design decisions (appropriate sizes of both client and server caches must be determined), and the goal of understanding disk caching behavior. In this work, trace-driven simulation was employed to study distributed system disk caching, and in particular the influence of client and server disk cache sizes. The results show clearly that client caching results in much degraded cache performance at the file server. In fact, in simulations with a single client and a single file server, the “overall miss ratio” (fraction of disk block references that miss in both client and scrver caches, and must therefore be satisfied by accessing a file server disk) may actually increase slightly when the client cache size is increased. This effect is somewhat counter-intuitive, but can explained by the degraded locality in references seen by the server when these references are filtered through the client cache. The degradation of file server cache performance with increasing client cache size appears to result in a strong correlation between the overall miss ratio and the maximum of the size of the server cache and the size of the client caches. Thus, if substantial client caching is done, server caching may be of limited usefulness. Alternatively, from the perspective of optimizing miss ratio, the placement of memory for disk caching at the server is much more effective than dividing the same amount of memory between the clients, for client disk caches. Of course, to obtain the best user response time and throughput capacity in any particular system of interest, the costs of network access and server congestion in that system must also be considered.

# A SURVEY ON REPLACEMENT STRATEGIES IN CACHE MEMORY FOREMBEDDED SYSTEMS

**Parag Panda et.al.,** has proposed. In this system Cache is one of the most power- consuming components in computer architecture. Power reduction in cache can be achieved by reducing miss rate, miss penalty, latency per access, and power consumption per access. The power reduction can also be achieved by shutting down unused part of the cache, by allowing not so recently used cache banks to sleep, reconfiguring the cache for specific application and various combinations of one or more of these. The cache hit depends on the cache size, associativity and the cache line size. Replacement strategies in associative mapping schemes play an important role in cache hit rate performance. This survey paper proposes a classification of these strategies with detailed discussion on their advantages and disadvantages. The performance, cost, size and power consumption are some of the major concerns in processor design. Reduction in power consumption is one of the most important and high priority design constraint in embedded processor architectures. The power issues of a processor has been addressed at various levels – technology level, architecture level, operating system level, compiler level and system and application program level. The main sources of power consumption are at application and system programs level, operating system level and architecture level. The majority of architecture level power consumption is due to memory sub-system which is because of the cache memory activities. Reduction in cache power consumption can be achieved by reducing miss rate, miss penalty, latency per access, and power consumption per access. By improving the cache-hit performance, the system can reduce the effective cache access time and overall power consumption. The cachehit performance in associative mapping schemes depends on the cache replacement strategies irrespective of the cache parameters like cache size, cache line size and associativity. The replacement strategy helps in reducing the number of cache misses and hence, the power consumption and effective cache access time. An ideal cache replacement strategy identifies a cache line which will not be accessed in near future as victim cache line for replacement. This is impractical as the future references are unknown. So the performance of the cache replacement strategy mainly depends on how accurately strategy can predict the future reference pattern based on the past references. The future reference pattern may depend on the past reference pattern and input data. The selection of a suitable replacement strategy for associative caches can have significant impact on the overall system performance. The choice of a replacement strategy is one of the most critical cache design issues. Though the amount of literature and information available for various cache replacement schemes is huge, there exist no recent in-depth survey articles of cache replacement strategies for embedded system memory. In embedded systems, the suitability of a cache replacement circuit is based on its hardware complexity and its worst case cache miss rate for a specific range of configuration and target workloads. It has to satisfy the WCET with minimum hardware overhead. This work analyses the cache miss rate and hardware complexity of the replacement circuits. Following section is a comprehensive survey on cache replacement strategies. Arrival based replacement strategy guarantees a predictable cache behavior with minimum hardware complexity. The widely used replacement strategy under arrival heuristic is First In, First Out replacement strategy (FIFO). FIFO chooses the oldest cache line in the set as the victim cache line for replacement. This scheme requires an Nbit register per cache line as additional hardware to store the arrival time information. On cache miss, the arrival time registers of all the cache lines in the set are compared to choose the victim cache line. Alternate implementation possible is to maintain a FIFO queue based on the arrival time of the blocks into the cache. When a replacement is necessary, the block at the head of the queue is removed. The replacement circuitry is activated and modified only on a cache miss. The cache access time is not affected by this replacement strategy as it is not in critical path. This strategy is useful in some of the embedded applications where the memory reference is in FIFO order. FIFO can replace most recently or most frequently used cache line/block because of its least arrival time, hence the cache hit performance of FIFO is poor. The FIFO strategy is not suitable for high performance system where cache hit rate required is very high. The cache memory performance has huge impact in overall system performance. One way to improve the cache memory performance is to improve the cache hit rate. This can be achieved with the help of an efficient replacement strategy. This paper has given an exhaustive survey of cache replacement strategies. A simple classification scheme for these replacement strategies is given and is used for the description and general critique of the described replacement strategies. Software assisted cache replacement strategies achieve the least cache miss rate as compare to hardware cache replacement strategies. However intelligent compiler, operating system and cache controller communication is required by software assisted cache replacement strategies.[10]

Multicore processors are being extensively used by real-time systems, mainly because of their demand for increased computing power. However, multicore processors have shared resources that affect the predictability of real-time systems, which is the key to correctly estimate the worst-case execution time of tasks. One of the main factors for unpredictability in a multicore processor is the cache memory hierarchy. Recently, many research works have proposed different techniques to deal with caches in multicore processors in the context of real-time systems. Nevertheless, a review and categorization of these techniques is still an open topic and would be very useful for the real-time community. In this article, we present a survey of cache management techniques for real-time embedded systems, from the first studies of the field in 1990 up to the latest research published in 2014.

**CHAPTER 3**

**SOFTWARE DESCRIPTION**

The software requirement specification is created at the end of the analysis task. The function and performance allocated to software as part of system engineering are developed by establishing a complete information report as functional representation, a representation of system behavior, an indication of performance requirements and design constraints, appropriate validation criteria.

# FEATURES OF JAVA

Java platform has two components:

* + - The Java Virtual Machine (Java VM)
    - The Java Application Programming Interface (Java API)

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

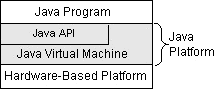


Figure No 3.1 Depicts of Java Program

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring Java's performance close to that of native code without threatening portability.

# SOCKET OVERVIEW:

A network socket is a lot like an electrical socket. Various plugs around the network have a standard way of delivering their payload. Anything that understands the standard protocol can “plug in” to the socket and communicate.

Internet protocol (IP) is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.

Transmission Control Protocol (TCP) is a higher-level protocol that manages to reliably transmit data. A third protocol, User Datagram Protocol (UDP), sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets.

# CLIENT/SERVER:

A server is anything that has some resource that can be shared. There are compute servers, which provide computing power; print servers, which manage a collection of printers; disk servers, which provide networked disk space; and web servers, which store web pages. A client is simply any other entity that wants to gain access to a particular server.

A server process is said to “listen” to a port until a client connects to it. A server is allowed to accept multiple clients connected to the same port number, although each session is unique. To mange multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

# RESERVED SOCKETS:

Once connected, a higher-level protocol ensues, which is dependent on which port user are using. TCP/IP reserves the lower, 1,024 ports for specific protocols. Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP, 119 is for Netnews- and the list goes on. It is up to each protocol to determine how a client should interact with the port.

# JAVA AND THE NET:

Java supports TCP/IP both by extending the already established stream I/O interface. Java supports both the TCP and UDP protocol families. TCP is used for reliable stream-based I/O across the network. UDP supports a simpler, hence faster, point-to-point datagram-oriented model.

# INETADDRESS:

The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address. User interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddress class hides the number inside. As of Java 2, version 1.4, InetAddress can handle both IPv4 and IPv6 addresses.

# FACTORY METHODS:

The InetAddress class has no visible constructors. To create an InetAddress object, user use one of the available factory methods. Factory methods are merely a convention whereby static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer. Three commonly used InetAddress factory methods are:

1. Static InetAddress getLocalHost ( ) throws UnknownHostException
2. Static InetAddress getByName (String hostName) throws UnknowsHostException
3. Static InetAddress [ ] getAllByName (String hostName) throws UnknownHostException

The getLocalHost ( ) method simply returns the InetAddress object that represents the local host. The getByName ( ) method returns an InetAddress for a host name passed to it. If these methods are unable to resolve the host name, they throw an UnknownHostException.

On the internet, it is common for a single name to be used to represent several machines. In the world of web servers, this is one way to provide some degree of scaling. The getAllByName ( ) factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to. It will also throw an UnknownHostException if it can’t resolve the name to at least one address. Java 2, version 1.4 also includes the factory method getByAddress ( ), which takes an IP address and returns an InetAddress object. Either an IPv4 or an IPv6 address can be used.

# INSTANCE METHODS:

The InetAddress class also has several other methods, which can be used on the objects returned by the methods just discussed. Here are some of the most commonly used. Boolean equals (Object other) - Returns true if this object has the same Internet address as other.

1. byte [ ] get Address ( ) - Returns a byte array that represents the object’s Internet address in network byte order.
2. String getHostAddress ( ) - Returns a string that represents the host address associated with the InetAddress object.
3. String get Hostname ( ) - Returns a string that represents the host name associated with the InetAddress object.
4. boolean isMulticastAddress ( ) - Returns true if this Internet address is a multicast address. Otherwise, it returns false.
5. String toString ( ) - Returns a string that lists the host name and the IP address for convenience.

# TCP/IP CLIENT SOCKETS:

TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point and stream-based connections between hosts on the Internet. A socket can be used to connect Java’s I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.

There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The Server Socket class is designed to be a “listener,” which waits for clients to connect before doing anything. The Socket class is designed to connect to server sockets and initiate protocol exchanges.

The creation of a Socket object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

* Socket (String hostName, int port) - Creates a socket connecting the local host to the named host and port; can throw an UnknownHostException or anIOException.
* Socket (InetAddress ipAddress, int port) - Creates a socket using a preexisting InetAddress object and a port; can throw an IOException.

A socket can be examined at any time for the address and port information associated with it, by use of the following methods:

* InetAddress getInetAddress ( ) - Returns the InetAddress associated with Socket object.
* Int getPort ( ) - Returns the remote port to which this Socket object is connected.
* Int getLocalPort ( ) - Returns the local port to which this Socket object is connected.

Once the Socket object has been created, it can also be examined to gain access to the input and output streams associated with it. Each of these methods can throw an IOException if the sockets have been invalidated by a loss of connection on the Net.

* InputStream getInputStream ( ) - Returns the InputStream associated with the invoking socket.
* OutputStream getOutputStream ( ) - Returns the OutputStream associated with the invoking socket0.

Several other methods are available, including connect( ), which allows you to specify a new connection; isConnected( ), which returns true if the socket is connected to a server; isBound( ), which returns true if the socket is bound to an address; and isClosed( ), which returns true if the socket is closed. To close a socket, call close( ). Closing a socket also closes the I/O streams associated with the socket. Beginning with JDK 7, Socket also implements AutoCloseable, which means that you can use a try-with-resources block.

# TCP/IP SERVER SOCKETS:

Java has a different socket class that must be used for creating server applications. The ServerSocket class is used to create servers that listen for either local or remote client programs to connect to them on published ports. ServerSockets are quite different form normal Sockets. When the user create a ServerSocket, it will register itself with the system as having an interest in client connections.

* ServerSocket(int port) - Creates server socket on the specified port with a queue length of 50.
* Serversocket(int port, int maxQueue) - Creates a server socket on the specified port with a maximum queue length of maxQueue.
* ServerSocket(int port, int maxQueue, InetAddress localAddress)-Creates a server socket on the specified port with a maximum queue length of maxQueue. On a multihomed host, localAddress specifies the IP address to which this socket binds.
* ServerSocket has a method called accept( ) - which is a blocking call that will wait for a client to initiate communications, and then return with a normal Socket that is then used for communication with the client.

# URL:

The Web is a loose collection of higher-level protocols and file formats, all unified in a web browser. One of the most important aspects of the Web is that Tim Berners-Lee devised a scaleable way to locate all of the resources of the Net. The Uniform Resource Locator (URL) is used to name anything and everything reliably. The URL provides a reasonably intelligible form to uniquely identify or address information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web.

# CHAPTER 4

**MATERIALS AND METHODS 4.1 EXISTING SYSTEM**

Online algorithm, called CEDC-O (collaborative edge data caching problem), is used in the existing system CEDC-O based on Lyapunov optimization to solve the CEDC problem cross multiple time slots without requiring future information, and prove the performance bounds of this algorithm. a particular area can constitute a graph, namely edge server network, where a node represents an edge server and an edge represents the link between two edge servers Mobile devices connect to edge servers to retrieve data. Compared to cloud servers, the storage resources on an edge server are usually limited due to their limited sizes.

# DRAWBACKS OF EXISTING SYSTEM

* + edge server are usually limited due to their limited sizes low latency
  + delay ratio is high
  + probabilistic routing protocol is not sufficient.
  + The performance is less when compared with the proposed method.
  + a real-world data set, and the results demonstrate that it insignificantly outperforms two representative approaches.
  + But the time taken to execute the program takes large amount of time.

# PROPOSED METHOD

Linear relaxation algorithm and the greedy algorithm method is used as the proposed method. The text file is given as the input then the caching limit is settled so that the file size and the remaining cache in kb can be obtained. First the connection the file list and the file request can be utilized in our project. relaxation algorithm is presented for finding a globally optimal solution for problem (P). The

algorithm finds an exact optimal solution to the problem after a finite number of iterations. A detailed discussion is included of how to implement the algorithm using only linear programming methods.

First the caching and multicasting method the total number of base station and the total number of users are created. In the caching model the linear mac function and the greedy mac function is used in the system. In the content server the file can be used in the base station and the caching method are executed.

# ADVANTAGES OF PROPOSED MODEL

* + Higher level of caching.
  + Data can be easily fetched along with the energy cost.
  + Faster computational efficiency.

# INPUT AND OUTPUT

Input as the sample text file as the total number of cache size can also be given as the input. Fetching of file according to the file size is considered as the output.

# MODULES DESCRIPTION

* + CONTENT SERVER
  + USER
  + CACHING AND MULTICASTING
  + MACRO CELL BASE STATION
  + LINEAR RELAXATION AND GREEDY MODEL

# CONTENT SERVER

The content server is the main part of the program that the selected tile list is added with the data and the particular file can be added, viewed, and the responsible base station can be added. The sbs id can be viewed and the cache size also be noted.

# USER

In the user module the number of files can be transferred to the nearest SBS and

then it can be connected. The user is the one where the total number of user in simulation can be created. In the user frame the total number of file list as well as the request can be viewed.

# CACHING AND MULTICASTING

In caching and multicasting the total number of SBS server base station can be created and the total number of users can be created in the caching and multicasting. Caching and multicasting at macro base stations (BSs) are two promising methods to support massive content delivery and reduce backhaul load.

# MACRO CELL BASE STATION

In the macro cell base station the SBS id along with the user file list can be allotted with the sbs id and the cache size. As the file is being fetched form every base station the selected dataset is being for every base station the number of files in the each entity is derived and the requested file from the each and every sbs is located and identified. The dataset from the user is located and the requested.

# LINEAR RELAXATION AND GREEDY MODEL

In the linear relaxation and greedy model the individual files according to area for the data is loaded in the form of linear relaxation and the greedy algorithm is used as the model for the energy cost. The energy cost is noted for the every transaction.as well for the greedy model the energy cost is implemented.

# SYSTEM REQUIREMENTS

* + 1. **HARDWARE REQUIREMENTS**:

|  |  |  |
| --- | --- | --- |
| * Processor Type | ` : | Pentium i3 |
| * Speed | : | 3.40GHZ |
| * RAM | : | 4GB DD2 RAM |
| * Hard disk | : | 500 GB |

|  |  |  |
| --- | --- | --- |
| * Keyboard | : | 101/102 Standard Keys |
| * Mouse | : | Optical Mouse |

# SOFTWARE REQUIREMENTS:

* + - * Operating System : Windows 8
      * IDE : Net Beans IDE 8.3
      * Coding Language : JDK 1.8 /JAVA

# FEASIBILITY STUDY

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

* Technical Feasibility
* Operation Feasibility
* Economical Feasibility

# TECHNICAL FEASIBILITY

The technical issue usually raised during the feasibility stage of the investigation includes the following:

Therefore, it provides the technical guarantee of accuracy, reliability and security. The software and hard requirements for the development of this project are not many and are already available in-house at NIC or are available as free as open source. The work for the project is done with the current equipment and existing software technology. Necessary bandwidth exists for providing a fast feedback to the users irrespective of the number of users using the system.

# OPERATIONAL FEASIBILITY

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following:

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

# ECONOMIC FEASIBILITY

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs.

The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

# TESTING AND IMPLEMENTATION

* + 1. **SYSTEM TESTING**

Testing objectives include

* + Testing is a process of executing a program with the intent of finding an error.
  + A good test case is one that has a high probability of finding an as yet undiscovered error.
  + A successful test is one that uncovers an as yet undiscovered error.

Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications. The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect -- it can only show that software defects are present.

# UNIT TESTING

Unit testing verification efforts on the smallest unit of software design, module. This is known as “Module Testing”. The modules are tested separately. This testing is carried out during programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

# INTEGRATION TESTING

Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole. In the integration-testing step, all the error uncovered is corrected for the next testing steps.

# VALIDATION TESTING

To uncover functional errors, that is, to check whether functional characteristics confirm to specification or not

**Several other methods are available, including connect( ), which allows you to specify a new connection; isConnected( ), which returns true if the socket is connected to a server; isBound( ), which returns true if the socket is bound to an address; and isClosed( ), which returns true if the socket is closed. To close a socket, call close( ). Closing a socket also closes the I/O streams associated with the socket. Beginning with JDK 7, Socket also implements AutoCloseable, which means that you can use a try-with-resources block** Assuming that the users find no major problems with its accuracy the system passes through a final acceptance test. This last test confirms that the system meets the original goals, objectives and requirements established during design.

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes where ever required.

The new system developed was tested by the acceptance testing method. Acceptance test incorporates both unit testing and integration testing. The user provided test area. Thus the system was successfully tested and it satisfies the user requirements. Afterwards it was implemented successfully.

# CHAPTER 5 EXPERIMENTAL SETUP

**ENERGY EFFICIENCY**

98.5

98

97.5

97

96.5

96

95.5

95

94.5

94

93.5

ENERGY EFFICIENCY

LINEAR RELAXATION GREEDY MODEL

Figure no 5.1 Energy Efficiency

|  |  |
| --- | --- |
| **ALGORITHM** | **ENERGY EFFICIENCY** |
| **LINEAR RELAXATION** | **98** |
| **GREEDY MODEL** | **95** |

Table 5.1

The energy efficiency for the each model is used as the linear relaxation model and the greedy model which are utilizing the various procedures for the energy efficiency for the training and testing model

# CHAPTER 6 RESULTS AND OUTPUT

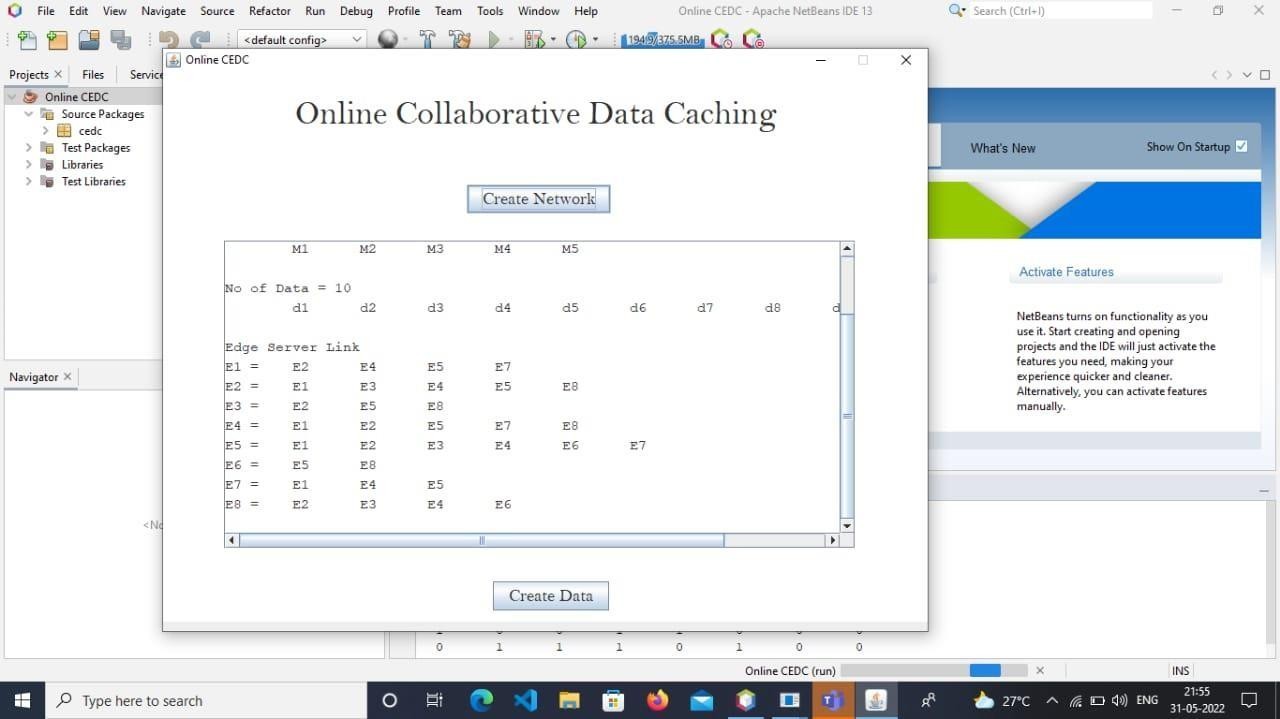


Figure 6.1 Content Server

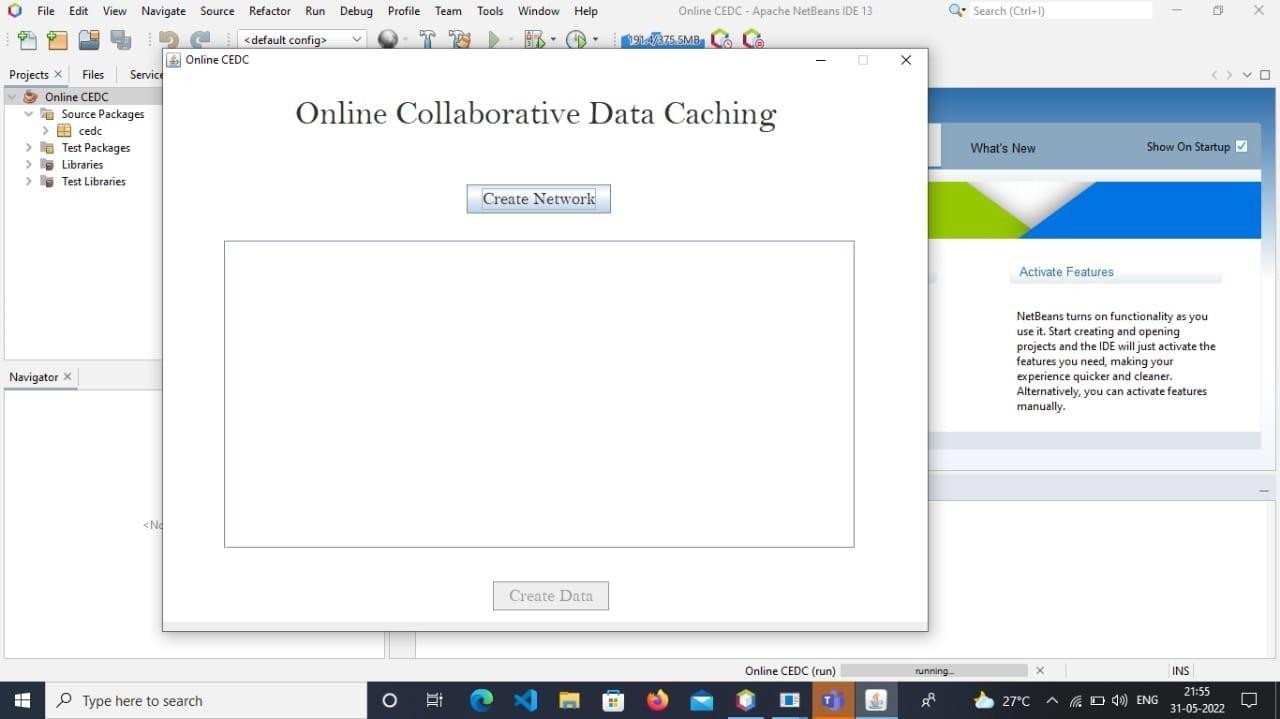


Figure 6.2 User

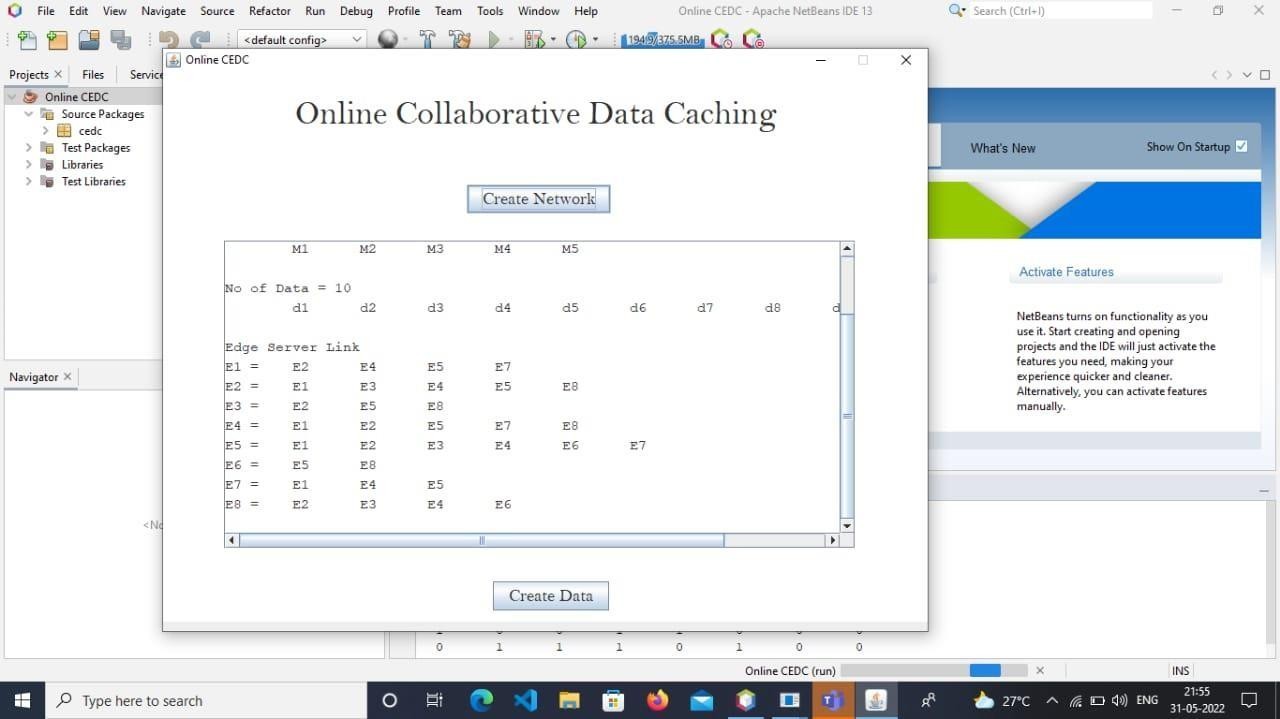


Figure 6.3 Caching and multitasking

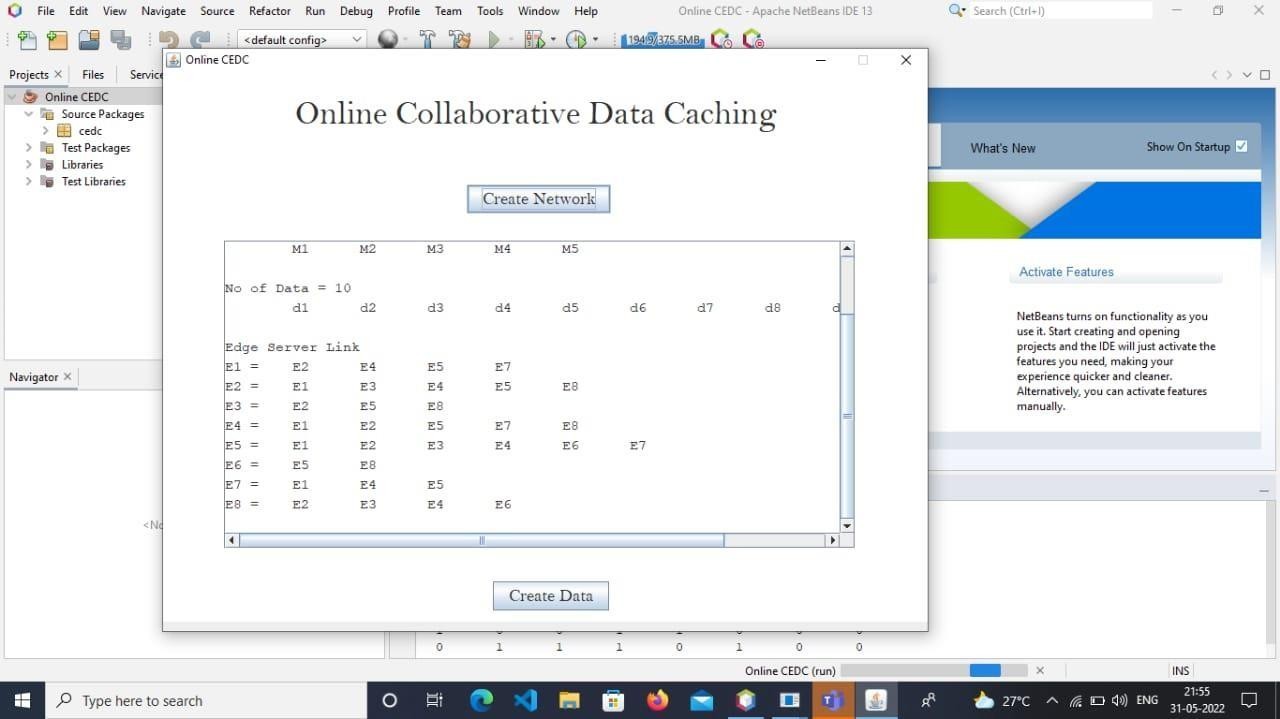


Figure 6.4 Macro cell base station

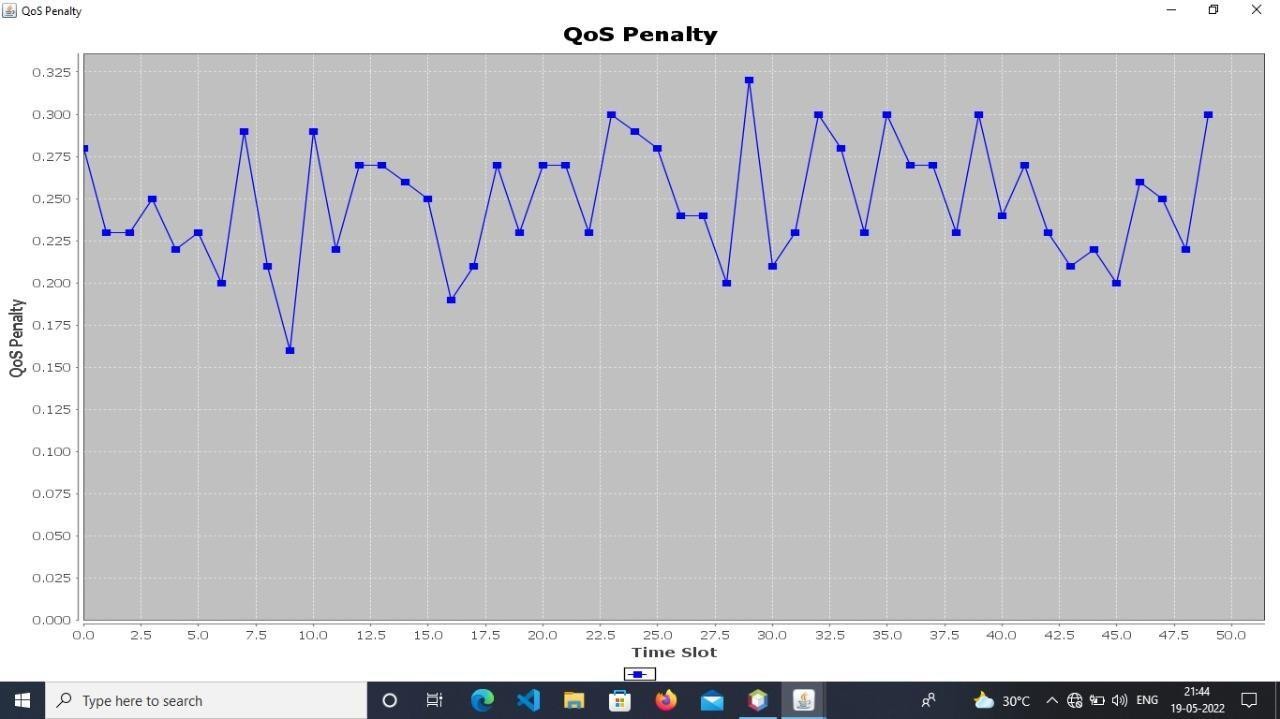


Figure 6.5 Qos Penalty

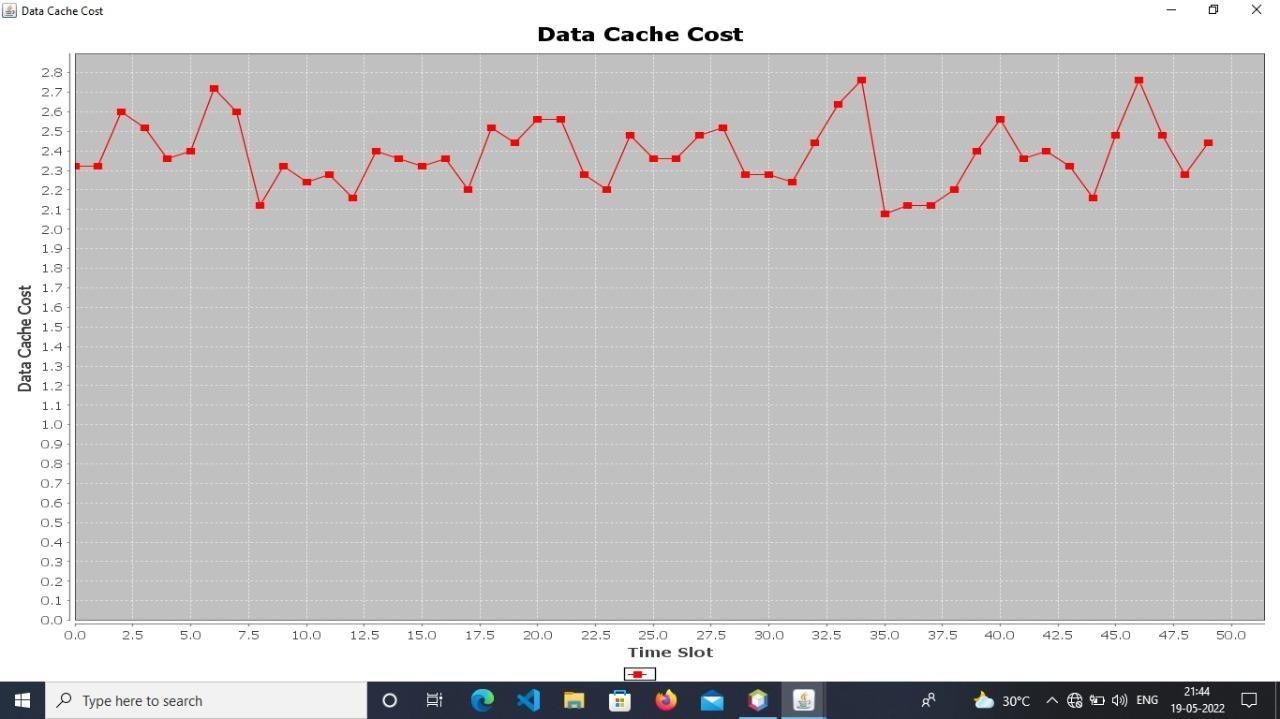


Figure 6.6 Data Cache Cost

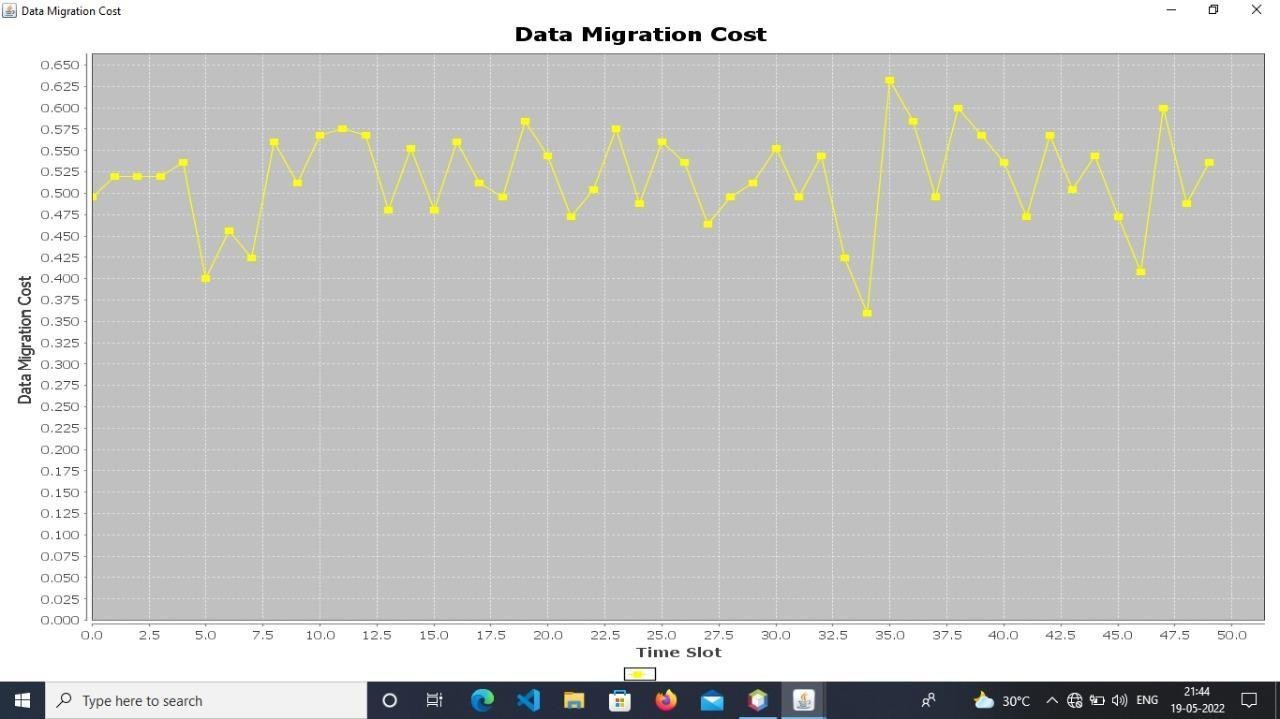


Figure 6.7 Data Migration Cost

**CHAPTER 7 CONCLUSION**

We studied the collaborative edge data caching (CEDC) problem. We first identified the major challenges and proposed a comprehensive cost model for this problem, where system cost is composed of data caching cost, data migration cost and QoS penalty. We also proved the NP completeness of the CEDC problem. We proposed, an online algorithm with provable performance guarantee, and evaluated its performance with extensive simulations. This research has established the foundation for the linear relaxation and greedy algorithm is used and opened up a number of future research directions. In our future work, we will consider dynamics on available edge server caches, user mobility and security policies.

# REFERENCES

1. A. Osseiran, V. Braun, T. Hidekazu, P. Marsch, H. Schotten, H. Tullberg, M. A. Uusitalo, and M. Schellman, “The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions,” in IEEE 77th Vehicular Technology Conference (VTC2013-Spring), 2013, pp. 1– 5.
2. P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, “Optimal edge user allocation in edge computing with variable sized vector bin packing,” in International Conference on Service-Oriented Computing, 2018, pp. 230–245.
3. L. Chen, S. Zhou, and J. Xu, “Computation peer offloading for energy-constrained mobile edge computing in small-cell networks,” IEEE/ACM Transactions on Networking, vol. 26, no. 4, pp. 1619–1632, 2018.
4. C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, “Computation offloading and resource allocation in wireless cellular networks with mobile edge computing,” IEEE Transactions on Wireless Communications, vol. 16, no. 8, pp. 4924–4938, 2017.
5. C. You, K. Huang, H. Chae, and B.-H. Kim, “Energy-efficient resource allocation for mobile-edge computation offloading,” IEEE Transactions on Wireless Communications, vol. 16, no. 3, pp. 1397– 1411, 2017.
6. X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2795–2808, 2016
7. Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, “A game- theoretical approach for user allocation in edge computing environment,” IEEE Transactions on Parallel and Distributed Systems, 2019.
8. P. Stenstrom, “A survey of cache coherence schemes for multiprocessors,” Computer, vol. 23, no. 6, pp. 12–24, 1990.
9. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, ¨ A. E. Lefohn, and T.

J. Purcell, “A survey of general-purpose computation on graphics hardware,” in Computer graphics forum, vol. 26, no. 1. Wiley Online Library, 2007, pp. 80–113.

1. S. Podlipnig and L. Bosz ¨ ormenyi, “A survey of web cache replace- ¨ ment strategies,” ACM Computing Surveys (CSUR), vol. 35, no. 4, pp. 374–398, 2019.

**PUBLICATION**

1. Ms. R. Keerthana, M. Naveen Kumar, D. Rajesh Kumar and G. Verjin Santhiya. “DATA CACHING METHODOLOGY IN EDGE COMPUTING” International journal of Health Sciences (IJHS), Acceptance received.

72