COLLABORATIVE DATA CACHING IN EDGE COMPUTING

BANDI KOMALI, CHINTHAKAYALA KRISHNAVENI, TALLAPAKA DIVYA

GUNREDDY HAARIKA, SUPERVISOR ,N MOWNIKA
Assistant Professor

ANURAG ENGINEERING COLLEGE
AUTONOMOUS
(Affiliated to JNTU-Hyderabad,Aprroved by AICTE-New Delhi)
ANANTHAGIRI (V) (M), SURYAPETA (D), TELANGANA-508206

Abstract*: Technology, especially mobile devices like smartphones, wearables, tablets, smart cars, and IoT gadgets, has been expanding at a rapid clip during the last decade. Congestion and increased latency are common results of such heavy usage of the network. Our solution included edge computing (EC) to deal with this problem. Edge computing has evolved as a means to redistribute processing power away from the cloud and onto edge computers. Our work is a web-based programme. By storing frequently used data on edge devices that are closer to end-users, collaborative data caching employing edge computing may boost the speed of web applications. In addition to a data-storing cloud server, edge servers are set up as well. Data stored on edge servers is automatically deleted after one day to save memory. The data is sent from the edge server or the cloud server to the mobile destination, where it is received, but the files can only be downloaded using a key. The created key and extremely secure data protection and privacy key are sent to the user's specified email address. There were a few key safeguards included in our collaborative data caching project to prevent unwanted eyes from spying on or intercepting private information.*

I. INTRODUCTION

The use of smartphones and other mobile devices has skyrocketed in recent years. Congestion and delays in the network are common results of the massive amount of traffic. In response, a new computing paradigm known as edge computing (EC) has arisen to shift the focus of computing resources away from the cloud and onto decentralised edge servers. Each edge server receives its power from one or more hardware devices and is connected to a nearby base station or access point used by mobile app users. By leasing processing power and data storage space from edge servers, providers of mobile and IoT applications may guarantee their consumers experience low latency and high-quality services. The computational burden and power consumption of mobile devices may be lowered by offloading compute workloads to neighbouring edge

servers. More and more information will be sent from the cloud to users' mobile devices by way of edge servers as the number of people using these applications grows exponentially. In order to minimise network latency while retrieving app data, app vendors should cache certain data, particularly popular ones like viral videos and postings from Facebook and Twitter. If data have been cached on the app's edge servers, users may access them without having to send a request to the cloud servers located far away. The app vendor's data transfer costs under the pay-as-you-go pricing model may be reduced by caching data on edge servers, which reduces the quantity of data moved between the cloud and the mobile devices. Hardware cache, such as central processing unit, graphics processing unit, memory, and discs; and software cache, such as the world wide web, database, and so on; are just a few examples of where data caching methods have been extensively employed. Data caching's benefits in lowering bandwidth use, decreasing network latency, and decreasing access prices have also been extensively explored in the network sector. Different types of network cache have been studied by academics in recent years. These types of studies include cache allocation and replacement techniques, coded caching, request routing, and information-theoretic caching.

How Edge computing works?

Fig 1.1 Working of edge computing

In edge computing's collaborative data caching, many edge devices store and exchange information. Typically, it goes like this: Edge devices, which are often positioned in close proximity to the source of the data created, gather the data. The data is processed and cached on the edge devices, where it may be accessed by local applications without being sent over the network. If two edge devices need to access the same information at the same time, one might ask the other to get it from its cache instead of going back to the source. As a result, there will be less strain on the network and faster retrieval of data. If an edge device doesn't have the necessary information stored locally, it may send a request to other edge devices in the network for that information. This establishes a cooperative caching system in which edge devices cooperate to speed up and improve the efficiency of data retrieval.

By lowering the quantity of data transported between the cloud and app users' mobile devices, caching on edge

servers may help ease the traffic stress on the Internet backbone.

One simple way for an app provider to reduce wait times for their consumers in a given region is to store copies of the most frequently requested data on all of the edge servers in that region. The app provider must take into account not just the latency of the data, but also the cost of using edge servers to cache data, which is based on a pay-as-you-go pricing model. Data transmission and transfer over the network also incurs a cost. As a result, from the perspective of an app vendor, it is crucial to find a collaborative data caching strategy that minimises the total system cost with limited storage spaces on edge servers while fulfilling the aforementioned constraints in the edge computing environment, such as server capacity constraint, server coverage constraint, and server adjacency constraint. Inevitably, as time progresses, fresh data will replace older data that has been stored on the edge servers. A vendor's cached app data and the storage space it has rented on edge servers together make up the edge caching system. There are a lot of positives to this concept. Cloud computing, which allows for the creation of a "content centric network" and a "content delivery network," is considerably different from EC. Connectivity between nearby EC servers deployed at separate base stations allows for bidirectional data transmission in the EC environment.

II. LITERATURE SURVEY

Authors: P. Lai, Q. He, M. Abdulrazak, F. Chen, J. Hosking, J. Grundy, and Y. Yang In recent years, the world has witnessed a surge in the number of cloud and mobile network connected end-devices. According to Ericsson’s mobility report, it is predicted that there will be produced a great challenge for online service providers in terms of guaranteeing a reliable and low-latency connection to end-users, which is one of the key quality-of-service (QoS) requirements. To tackle this issue, edge computing came in to place. In in which computation, storage, and networking resources are pushed closer to the edge of the network by deploying a number of intermediate edge servers with closer proximity to end-devices. 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, they 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. They have conducted three series of experiments to evaluate the

proposed approach against two representative baseline approaches, greedy and random. 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.

Authors: A. Mukhopadhyay, N. Hegde and M. Lalage Recent years have seen an explosive growth in Internet traffic, stemming mainly from the transfer of multi-media contents, e.g., streaming videos, movies etc. Such growth in multi-media traffic has led to the emergence of content delivery networks (CDNs) and peer-to-peer systems. Large CDNs usually consist of a central server, storing an entire catalogue of contents, and a large number of edge servers, each storing a small fraction of these contents in their caches and serving requests of the stored contents. In such systems, it is assumed that access to the central server is expensive. Therefore, a large portion of the content requests must be served by the edge servers that are constrained by their limited memory and bandwidth capacities. In this paper, they model these servers as loss servers and aim at minimizing the number of requests blocked at these servers. In such systems, the throughput crucially depends on how contents are replicated across servers and how the requests of specific contents are matched to servers storing those contents. In this project, they first formulate the problem of computing the optimal replication policy which if combined with the optimal matching policy maximizes the throughput of the caching system in the stationary regime. It is shown that computing the optimal replication policy for a given system is an NP-hard problem. They then propose a simple randomized matching scheme which avoids the problem of interruption in service. The dynamics of the caching system is analysed under the combination of proposed replication and matching schemes. We study a limiting regime, where the number of servers and the arrival rates of the contents are scaled proportionally, and show that the proposed policies achieve asymptotic optimality. Extensive simulation results are presented to evaluate the performance of different policies and study the behaviour of the caching system under different service time distributions of the requests.

Authors: L. Chen, S. Zhou, and J. Xu Many new applications, turning data and information into actions that create new capabilities and unprecedented economic opportunities. Although cloud computing enables convenient access to a centralized pool of configurable and powerful computing resources, it often cannot meet

the stringent requirements of latency-sensitive applications due to the often-unpredictable network latency and expensive bandwidth. The growing amount of distributed data further makes it impractical or resource prohibitive to transport all the data over today's already congested backbone networks to the remote cloud. mobile edge computing (MEC) has recently emerged as a new computing paradigm to enable in-situ data processing at the network edge, in close proximity to mobile devices and connected things. Located often just one wireless hop away from the data source, edge computing provides a low-latency offloading infrastructure, and an optimal site for aggregating, analysing and distilling bandwidth hungry data from end devices. In this project, we study computation peer offloading in MEC-enabled small cell networks aiming to address the aforementioned challenges. Our goal is to maximize the long-term system- 4 wide performance (i.e., minimizing latency) while taking into account the limited energy resources committed by individual SBS owners.

Authors: C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang However, traditional wireless cellular networks are becoming incapable to meet the exponentially growing demand not only in high data rate but also in high computational capability. In order to address the data rate issue, the heterogeneous network structure was recently proposed, in which multiple low-power, local coverage enhancing small cells are deployed in one macro cell. To address the spectrum allocation issue, the work in proposes a graph colouring method to assign physical resource blocks(PRBs) to user's equipment (UEs). On the other hand, to address the computational capability issue, mobile cloud computing (MCC) systems have been proposed to enable mobile devices to utilize the powerful computing capability in the cloud. In order to further reduce the latency and make the solution more economical, the fog computing has been proposed to deploy computing resources closer to end devices. In this paper, they presented an ADMM-based decentralized algorithm for computation offloading, resource allocation and internet content caching optimization in heterogeneous wire-less cellular networks with mobile edge computing. They formulated the computation offloading decision, spectrum resource allocation, MEC computation resource 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.

III SYSTEM DESIGN

In this process we define the system architecture and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to the development of the product. The design phase helps to produce the overall design of the software. The goal of this phase is to figure out the different modules that can be used for the given system to achieve its goal with the greatest possible accuracy and efficiency. The system design contains details about each of the modules being used along with the way they interact with the other modules and help produce the output. The output of the design process is a description of the software architecture.

SYSTEM ARCHITECTURE

Architecture diagram represents mainly flow of request from the users to database through servers. In this scenario overall system is designed in three tiers separately using three layers. This project was developed using 3-tier architecture.

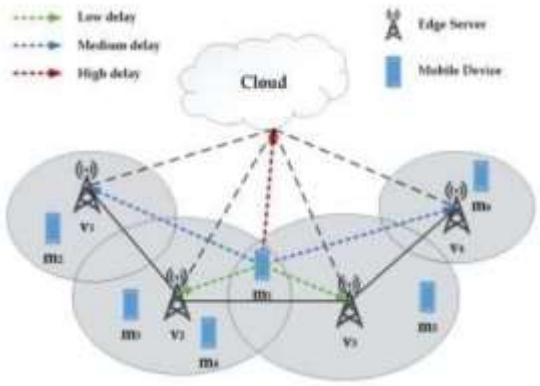

Fig 4.1.1 System Architecture

DATA FLOW DIAGRAM

A data-flow diagram is a way of representing the flow of a data of a process or a system. The data flow diagram also helps us to monitor what data we are feeding to a given component of the program and what output data it generates after processing. The data flow diagram is just the graphical representation of the flow of data through the information system. A flow of events is a sequence of transactions (or events) performed by the system.

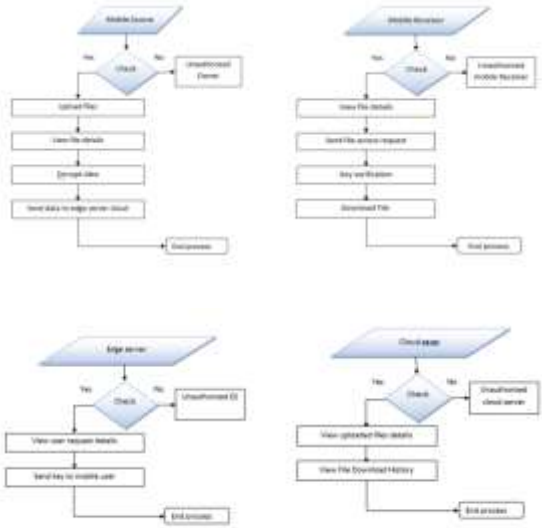

Fig 4.2.1 Data Flow Diagram

UML DIAGRAMS

UML is a graphical notation used to visualize, specify, construct and document the artifact of software intensive. UML is appropriate for modelling systems ranging from Enterprise Information Systems to Distributed Web-based Application and even to Hard Real-time Embedded systems. UML effectively starts with forming a conceptual modelling of the language is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

• UML stands for Unified Modelling Language.

• UML is different from the other common programming languages such as C++, Java, COBOL, etc.

• UML is a pictorial language used to make software blueprints.

• UML can be described as a general-purpose visual modelling language to visualize, specify, construct, and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design. After some standardization, UML has become an OMG standard. UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system. The system can be a software or non-software system. Thus, it must be clear that UML is not a development method rather it accompanies with processes to make it a successful system.

Use case Diagram:

A use case diagram describes a set of sequences in which each sequence indicates the relation with outside things. A use case involves the interaction of actor and system. A use case contains an actor. An actor refers to various people that use system. The use case diagram is used to capture the dynamic aspects of a system and the functional requirements of a system.

Fig 4.3.1.1 Use Case Diagram

IV IMPLEMENTATION

ONLINE CACHING ALGORITHM DESIGN

To solve the CEDC problem optimally, the complete information about the system over all the time slots must be known. However, this cannot be realistically fulfilled for real-world scenarios. To practically fulfil the app vendor's long-term latency constraint , we need to convert P1, a non-convex problem, to a linear and convex problem. To do so, we propose an Online Collaborative Edge Data Caching (CEDC-O) algorithm based on Lyapunov optimization for finding near-optimal solutions to the CEDC problem in individual time slots without future information. The notations adopted in this section are summarized in Table 2.

TABLE 2: Notations in our Algorithm Design

Notation	Description
C	value of system cost produced by λ
C	0 value of system cost produced by λ 0
C	opt value of system cost produced by λopt
Cmin	smallest system cost of all possible solutions
Cmax	largest system cost of all possible solutions
DP(t)	Lyapunov drift-plus-penalty function
Lavg(λ t)	average system latency in time slot t
L(σ(t))	Lyapunov function, calculated by 1 2 σ 2 (t)
Q	constant value equal to 1 2 L2
Q0	constant value equal to Q + γ · (Cmax − Cmin)
Γ	positive parameter adjusting the trade-off between system cost C(λ t)
λ	solution obtained by CEDC-O
λ t	λ in time slot t
λ 0	feasible solution fulfilling (21)
λ ∗	feasible solution fulfilling (27)
λ ∗t	λ ∗ in time slot t
λopt	optimal solution to P1 over all time slots

Online Collaborative Edge Data Caching Algorithm: We provide an online algorithm, named CEDC-O, based on Lyapunov optimization, to convert the long-term optimization problem P2 to optimization problems in individual time slots. The most significant characteristic of CEDC-O is that it only requires the information in the current time slot rather than the complete information in all the time slots when solving P2. While trying to minimize the system cost, the app vendor also needs to stabilize the system latency to ensure low-latency data access for its users. Thus, in this paper, the system metric to stabilize by CEDC-O is the time averaged system latency perceived by the users over the long term. Lyapunov optimization is typically applied in the communication and queuing systems. With the application of Lyapunov optimization, the problems can be formulated as problems that optimize the time averages of certain objectives subject to some-time

average constraints, and they can be solved with a common mathematical framework that is intimately connected to queuing theory. Unlike the typical application of Lyapunov optimization that models the problem as a queuing network, we define the accumulated latency in Definition 1 to stabilize the system latency over time. Definition 1 (Accumulated Latency). Accumulated latency σt is the overdue delay accumulated over t time slots, calculated as: σ(t + 1) = max{σ(t) + Lavg(λ t) − L, 0} (15) where Lavg(λ t) = P m∈M P d∈D θ t m,d·l t P m,d m∈M P d∈D θ t m,d , and σ(0) = 0 because there is no latency at the very beginning. Based on Definition 1, the accumulated latency will increase if the latency is over L in the previous time slot. This can be employed as a penalty to adjust the data caching strategy to stabilize the system latency over time as specified by (13). Now, we can convert the long-term latency constraint (13) to a new constraint based on accumulated latency: Lim T→∞ 1 T X−1 t=0 E[σ(t)] ≤ 0 (16) Given (15), a Lyapunov function can be defined as L(σ(t)) , 1 2 σ 2 (t). It indicates the system.

stability measured by its accumulated latency L(σ(t)). Here, the Lyapunov drift Δ(σ(t)) is applied in each time slot to enhance the system stability: Δ(σ(t)) = E[L(t + 1) − L(t)|σ(t)] = 1 2 E[σ 2 (t + 1) − σ 2 (t)|σ(t)] = 1 2 E[(Lavg(λ t) − L) 2 |σ(t)] + σ(t) E[Lavg(λ t) − L|σ(t)] ≤ Q + σ(t) E[Lavg(λ t) − L|σ(t)] (17) where Q = 1 2 L 2 because of Lavg(λ t) ≥ 0. As we obtain the upper bound of the Lyapunov drift function, we introduce the penalty in our CEDC-O algorithm based on the total cost objective (12). We denote γ as a positive parameter in Lyapunov optimization for adjusting the trade-off between the system cost C(λ t) and the number of time slots needed to converge the time-averaged latency back to L when (13) is violated. Here, we introduce the Lyapunov drift-plus-penalty function DP(t), defined as: DP(t) = Δ(σ(t)) + γ · E[C(λ t)|σ(t)] (18) In each time slot, the data caching strategy is formulated to minimize the total cost C(λ t) and to keep the system stable, and we can get the upper bound of this function by: DP(t) ≤ Q + σ(t) E[Lavg(λ t) − L|σ(t)] + γ · E[C(λ t)|σ(t)] (19) The pseudocode of the CEDC-O algorithm is presented in Algorithm 1. In each time slot, the data caching strategy is formulated by finding the optimal solution to P2: P2 : min(Q + σ(t)(Lavg(λ t) − L) + γ · C(λ t)) s.t. : (1), (3), (6), (4), (15)

V RESULTS

Fig 8.1 Home Page

Fig 8.2 Mobile Source Home Page

Fig 8.25 Download File

Fig 8.26 Received file

VI CONCLUSION

In this project, we are able to build a web application. It is a promising approach to improving the performance of distributed systems. By leveraging the resources of edge devices, such as smartphones and routers, data can be cached closer to end-users, reducing network latency and improving response times. It also provides security and privacy to the data by generating a confidential key. So, in that case data is secure and available only for the user it sends. This application helps user with privacy to their data and consumption less time and less cost. In this Project, 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 N Completeness of the CEDC problem. We proposed CEDC-O, an online algorithm with provable performance guarantee, and evaluated its performance with extensive simulations. This research has established the foundation for the CEDC problem 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] P. Lai, Q. He, M. Abdulrazak, 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.

[2] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," IEEE COMSOC MMTC Commun.-Frontiers, vol. 12, no. 4, pp. 29–33, 2017.

[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] A. Mukhopadhyay, N. Hegde and M. Lalage, "Optimal content replication and request matching in large caching systems", Proc. IEEE Conference Computer Communication., pp. 288-296, 2018.

[6] 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.

[7] 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.

[8] 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.

[9] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," Computer, vol. 23, no. 6, pp. 12–24, 1990.

[10] J. D. Owens, D. Luebke, N. Govinda Raju, 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.

[11] B. Jacob, S. Ng, and D. Wang, Memory systems: cache, DRAM, disk. Morgan Kaufmann, 2010.

[12] A. J. Smith, "Disk cache—miss ratio analysis and design considerations," ACM Transactions on Computer Systems (TOCS), vol. 3, no. 3, pp. 161–203, 1985.

[13] S. Polding and L. Bosz ¨ ormenyi, "A survey of web cache replacement strategies," ACM Computing Surveys (CSUR), vol. 35, no. 4, pp. 374–398, 2003.

[14] K. Elhardt and R. Bayer, "A database cache for high performance and fast restart in database systems," ACM Transactions on Database Systems (TODS), vol. 9, no. 4, pp. 503– 525, 1984.

[15] A. Mukhopadhyay, N. Hegde, and M. Lalage, "Optimal content replication and request matching in large caching systems," in IEEE INFOCOM 2018-IEEE Conference on Computer Communications, 2018, pp. 288–296.