International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 03 | Mar-2018 www.irjet.net p-ISSN: 2395-0072

Improve Client performance in Client Server Mobile Computing System using Cache Replacement Technique

Sanjay Kumar [1], Sandhya Umrao[2]

[1]School of Computing Science & Engineering, Galgotias University, India
Galgotias College of Engineering and Technology, Greater Noida, India

---

Abstract: *Mobile computing is becoming more and more popular and common. In the near future, people will be able to use mobile computers (Laptops, palmtops, etc.) to do various computing tasks and to access the Internet from anywhere and anytime at will. The development of wireless facilities and mobile computers has made this goal possible, and some of the mobile applications exist already. But the situation is still far from ideal.*

There is still a lot of work to be done to reach the goal. Compared with a wired network environment, a mobile computing network has a lot of problems. A wireless link has low bandwidth and is unstable. Mobile devices are resource limited, like memory, storage, display area, power, etc. All these factors cause problems for mobile applications. Mobile network users usually suffer from long latencies and frequent disconnection, mobile applications suffers from bad performance. In recent years, a lot of researches have been done to deal with these issues, to improve the mobile application performance. The proposed techniques include caching, prefetching, etc. The goal of this project is to study the techniques to improve client performance in Client/server mobile computing systems, to identify which of the techniques are most promising. We designed and implemented a mobile client/server application – a Mobile System. Based on our study on mobile computing techniques and the features of the mobile application, we identified some caching and prefetching techniques and integrated them into our mobile application. Through experiments, we studied the impact of these techniques to the performance of the application and concluded that the method is a better solution to improve the performance of the mobile application.

Key Words: **MH-Mobile Host, GDS –Greedy Dual Size, LRU-Least Recently Used, LFU-Least Frequently Used, FIFO-First in First out, LFF-Largest File First.**

1. INTRODUCTION

1. Mobile Computing Background

With the advent of cellular technology and portable computers we are on the verge of a new computing paradigm. This computing paradigm is now widely known as “Mobile” or “nomadic” computing]. In recent years Computer and Communication technologies have been developing fast. Using computers and accessing network information resources have become a necessary part of our daily work and daily life. There are numerous computers around the world that are connected by various kind s of networks. There are numerous applications that allow people to do almost everything they want. But, this situation is mostly restricted to users at fixed locations with static desktop computers and static wired networks.

There is still a lot of time we are mobile – moving among offices, homes, planes, trains, automobiles, conference rooms, and classrooms and such. There is need to access computing resources not only when stationary but also while mobile or in transit

As technology improves in the area of wireless facilities and mobile computers, mobile computing has become feasible. As of today, a variety of advanced mobile devices, some mobile wireless systems and mobile computing applications exist already. For example, people can send and receive emails and access Internet web sites using mobile computers via wireless networks. The future trend of telecommunication is to extend the telecom and computing services to mobile users, to break the restriction of user locations, to allow people access to computing resources anywhere and anytime at will. However, this is not a trivial task. Compared to static systems, mobile computing systems are constrained in important ways. These constraints are intrinsic to mobility, and are not just artifacts of current technology. Mobile elements are resource -poor relative to static elements. Wireless links have low bandwidth and are unstable. Mobile elements must operate under a much broader range of networking conditions. The nature of wireless communication media and the mobility of computers combine to create fundamental new problems in networking, operating systems, and information systems].

Mobile computing is still at its early stage. In recent years there has been a lot of research in the mobile computing area. It has still a long way to go to reach the goal of connecting anywhere and anytime.

2. Mobile Computing Systems – Architecture and Main Components

1. A typical mobile computing system consists of the following components.
2. A static communication backbone using wired means
3. A set of static hosts
4. Sets of mobile hosts (MHs) that can communicate with a predetermined static host
5. Mobile subnets/wireless cells.

The following figure displays the architecture of a general mobile computing system.

The elements in a mobile computing system fall into two distinct sets: mobile units and fixed hosts. Some of the fixed hosts, called Mobile Support Stations, are augmented with a wireless interface to communicate with mobile units, which are located within a radio coverage area called a cell. A cell could be a real cell as in cellular communication network or a wireless local area network that operates within the area of a building. In the former case the bandwidth will be severely limited (supporting data rates on the order of 10 to 20 Kbps). In the latter, the bandwidth is much wider – up to 10 Mbps. Fixed hosts will communicate over the fixed network, while mobile units will communicate with other hosts (mobile or fixed) via a wireless channel

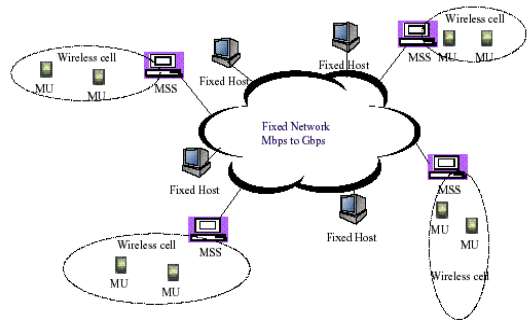

Figure 2.1. Architecture of Mobile Communication Networks

2.1. Mobile Software System

The previous sections are dedicated to mobile hardware systems. In this section we discuss mobile software systems. A mobile software system contains the following components:

1. Mobile operating systems for mobile devices.
2. Application client software running on mobile computers
3. Application server and/or database server software on workstations in wired networks
4. Mobile middleware

The following figure displays the architecture of the software system for a mobile client.

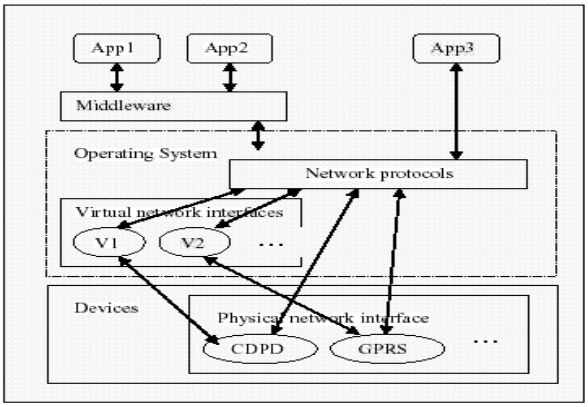

Figure 2.3. Mobile client architecture

The fundamental limitations of mobile hardware systems have important impact on the design of mobile software systems. The software system has to be small to meet the memory requirement; the algorithms have to be efficient to optimize the utilization of limited computing power and battery. In the following we discuss the operating systems, middleware, and applications respectively.

2.1.1 Mobile Operating Systems

The mobile operating system is responsible for managing physical resources, power consumption and communication interface. A mobile operating system is composed of the computer kernel, the power management facility, and the real -time kernel. The computer kernel is the entity that manages access to physical resources, such as CPU and disk space. The power management facility is responsible for reducing power consumption. The real -time kernel is responsible for managing the communication link.

The currently available mobile operating systems include Windows CE from the Microsoft Corporation, Palm OS from Palm, and Symbian EPOC from the Symbian LTD. These operating systems differ in memory requirements, the range of applications that can be supported, communications capability and interfaces.

2.1.2 Mobile Middleware

Mobile middleware is the layer of software that lies between the mobile applications and the underlying wireless networks. The middleware can be found at 3 different locations: the mobile client, the mobility gateway, and the information server. These middleware at different locations work together to shield users from

the underlying wireless networks and the mobility of the users.

The range of services provided by a mobile middleware system to the application layer may include adaptivity and QoS, service management, TCP/IP functionality, disconnected operations, proxies and agents, communication resilience, and continuous data synchronization. By delegating the complex tasks of dealing with wireless communications to the mobile middleware layer, individual applications can focus mainly on their own application –specific logic, thus significantly reducing their complexity. Mobile application developers need only to develop their applications according to a mobile API specification. With a standardized mobile API, the applications that comply to such an API will be able to run across different wireless networks. Interesting mobile and terminal aware applications can be written to make full use of the intelligence embedded within the mobile middleware.

Some tasks which are common to applications in a related area can be put in the middleware. For example, location information service and distance calculation can be included in middleware and serve various location-based applications.

Mobile middleware plays a central and brokerage role in the interaction between applications and the external wireless environments, as well as with the internal system resources. This factor makes the middleware one of the most interesting and challenging components of the mobile software systems.

2.1.3 Mobile Computing Applications

The application layer occupies the pinnacle of the mobile software system. Like the other layers of mobile software systems, designing mobile applications has to take into account of the fundamental limitations of mobile systems. Mobile devices should not be considered general -purpose computers to run complex simulations and large applications due to their limited resources. Applications for mobile computers can be divided into the following five categories:

1. Standalone applications such as games or utilities
2. Personal productivity software (word processors, presentation software, calendars);
3. Internet applications such as email, WWW browsers
4. New location-aware applications such as tour planners and interactive guides.
5. Ad-hoc network and groupware applications

The Internet applications constitute a very interesting and challenging category. This is the category we are interested in for this project. There are abundant Internet applications for desktop computers today. To introduce these applications into the mobile market place is an important ongoing trend with a high market potential. The applications getting the most attention are the ones that best take advantage of the mobility of wireless Internet devices, and which suffer least from the limitations of wireless platforms.

It's not surprising that e -mail is a top priority. E -mail is, in many ways, the perfect application for wireless devices, since technological constraints of small screen size and limited bandwidth do not limit e -mail as much as they do other applications. Another important mobile computing application is WWW mobile browser. It is the key part to extend the WWW services to mobile hosts. There are a number of browsers for handheld computers already. Since the cost of wireless services is high, general purpose web browsing might not be an attractive task for mobile users. Mobile browsers designed to provide a specific information service (e.g., financial information, daily banking, stock trading, weather information …) could be of more interest.

With the prospect that mobile devices may become the de facto standard for purchases, mobile e-commerce is a big business, and, as we might expect, it is another one of the top priorities for application developers. Finally, location -based services, for instance, providing users with information about restaurants they are close to is one of the most obvious advantages of the wireless Internet.

Developing applications for mobile systems is an area full of imaginations, challenges and with huge future market potential.

3: RELATED WORK

-Improving Performance of Mobile Computing Applications

A mobile computing system has its intrinsic constraints. Mobile devices are resource poor and wireless networks have low bandwidth and are unstable; this causes mobile computing applications to suffer from bad performance and mobile users to suffer fro m long latency. A lot of research has been carried out in this area and various techniques have been proposed to deal with this issue. The effort of improving performance of mobile computing systems contains every aspect of mobile environment, from hardware aspect to software aspect, from proper network configuration to efficient application design.

In this project we concentrate on techniques to improve mobile computing performance at the application level. We research techniques to reduce user perceived latency

and effective wireless bandwidth usage in mobile computing applications.

For mobile computing users, they always want fast access to networks and low cost.

This means short user perceived latency and small bandwidth consumption. Research in recent years in mobile computing has proposed many techniques for optimizing data transfer over wireless links and utilizing mobile device resources. The commonly used techniques include caching, prefetching and compression. Each technique has its strength and weakness. Many of the techniques can be used in different combinations in different circumstances. These techniques often trade off one resource for another. For example, compression trades off computation for bandwidth, caching trades off memory for latency (and bandwidth), and prefetching trades off bandwidth for latency. The utility of these techniques thus depends on the service desired, the device, the network characteristics and the features of the application.

In the following sections, we review and discuss these techniques respectively. To keep the project consistence and make it easy to explain, we refer to all data objects in this chapter as files.

3.1. Caching

In a mobile computing environment, the bandwidth of the wireless link is very limited.

This implies that each mobile client should minimize wireless communication to reduce the contention for the narrow bandwidth. Caching of frequently accessed data at the mobile clients has been shown to be a very useful and effective mechanism in handling this problem.

Caching improves the efficiency and reliability of data delivery over the network. A cache in the mobile client can serve a user's request quickly and reduce user perceived latency. When a user makes a request, the client checks the cache, if the required information is there, the client replies with it; otherwise, the client fetches it from the information server, replies to the user, and stores a copy in the cache as well. Other benefits of caching include energy savings and cost savings.

A caching strategy consists of the following components: cache size and replacement strategy. For mobile devices, since cache storage is limited, what to cache, when to cache, how long to cache, and how to replace the cached data have to be carefully considered. File size and data type impact the caching policy. Research in this area has resulted in many caching strategies, and they have been widely used in network applications, for example, in web browsers and mobile applications.

The efficiency of a caching strategy is measured by the following performance metric.

1. The cache hit rate: the ratio of the number of files sent to users from the cache to the total number of files served from the cache and content servers on the Internet.
2. The byte hit rate: the ratio of the average size of files sent to users from the cache to the average size of all the files served from the cache and content servers on the internet.

The byte -hit ratio gives more information of the network bandwidth. As files have different sizes, only recording the cache hit rate will not give a correct insight on how the strategy impacts the network bandwidth.

Cache consistency maintenance is another important issue to consider when applying caching in an application. Due to limitation of battery power, mobile computers may often be forced to operate in a sleep or even totally disconnected mode. As a result, the mobile computer may miss some cache invalidation reports broadcasted by a server, making the cached file out -dated or forcing it to discard the entire cache contents after waking up.

3.1.1 What to Cache and When to Cache

If there is space and the file is not dynamic, the file should always be cached. When space is insufficient, there are three simple strategies for deciding which files to cache: "cache all", which removes other files to make space; "threshold", the same as the previous strategy, but only caching files below a certain size; and "adaptive dynamic threshold," whereby the maximum file size threshold alters dynamically.

3.1.2 Cache Replacement Schemes

File replacement strategies have a great impact on cache performance. The replacement process is tightly related to the cache size: if an infinite cache exists, there is no need for such a file removal process since all the files that enter the cache are kept for ever. In reality, the cached files have to be replaced because of a lack of memory space, and the choice of the next file to be removed is quite important. When deciding which file to discard, the key point is to discard the file that is most unlikely to be used again.

Factors that impact cache performance include: number of references to the files, the file size, and the file time -to-live, and file retrieval time. For example, consider the factor of file size. For all caches, an upper limit is defined for the size of the caching area. If the cache is full when it receives a request to store a large file, what is the best strategy to maximize caching benefit? Is it more sensible

to replace a single large file than several smaller ones? It becomes more and more clear that an appropriate algorithm for selecting the files should consider several factors. However, the appropriate algorithm to combine these factors has not been defined yet.
The file replacement strategy can be split in two phases: first, the documents are sorted in order to determine the removal order; second, one or more documents are removed from the removal list. Cache replacement strategies can be divided into two groups: on -demand replacement strategies and periodic replacement strategies. In an on -demand replacement strategy, the removal process is started once the cache is full or the remaining amount of memory is not enough to store the new incoming file. In a periodic replacement strategy, files are removed after a certain amount of time. The possible disadvantage of the on -demand replacement strategy is that if a cache is nearly full, then running the removal policy only on -demand will invoke the removal policy on nearly all document requests, if removal is time consuming, it might generate a significant overhead. On the other hand, periodic removal reduces the hit -ratio because files are removed earlier than required. In the following we study different cache replacement strategies.

1. Random replacement: A random replacement simply discards a randomly chosen file.
2. The least recently used (LRU): The simplest and most common used cache management approach, which removes the least recently accessed files until there is sufficient space for the new file. This approach not only stores data also the time that each file was last referenced. LRU is widely used as a replacement policy. It is well understood and shown to perform well in a wide variety of workloads.
3. The least frequently used (LFU): LFU keeps track of the number of times a file is used and replaces the least frequently used file to make room for an incoming file. LFU is more complex to implement than LRU. In LRU, a referenced file is just placed at the head of removal list. In LFU, the referenced file needs to be placed in a sorted (by frequency of use) list and this in general is more complex. One modified version of LFU is LFU-aging. LFU-aging allows an incoming file to replace only blocks with a frequency count of 1.
1. First-in-first-out (FIFO): discards the file that is the oldest in the cache. It is easily implemented as a circular buffer.
2. Largest-file-first (LFF): discards the largest file in the cache.

There are a lot of hybrid strategies and generalizations of the above caching strategies.

Some generalizations of LRU attempt to make it more sensitive to the variability in file size and retrieval delays, like Greedy Dual algorithm and Greedy -Dual-Size (GDS). Some generalizations of LRU attempt to incorporate access frequency information into LRU, like LRU -K.

All the above methods only consider a single factor. Research has shown that a single or a combination of several parameters in the file replacement strategy is not enough to achieve higher performance. New methods have been proposed in. In these methods, each document is assigned a priority according to which the removal process will select the next document to be replaced. The priority is computed according to a certain mathematical model. In each model, the different factors are assigned a weight which is calculated using weight optimization methods or simulation techniques

3.2 Prefetching

Prefetching is a technique which allows a client to download information in advance, ready for the user to use later. Prefetching tries to reduce the perceived latency by look -ahead. If a user can anticipate what files he will fetch in the future, it is possible to preload these files to the client before they are requested.

The idea of prefetching stems from the fact that, after retrieving some information (webpage, file, etc), the user usually spends some time viewing and processing the information. During this period, the connection link is idle, if we can take advantage of this phenomenon by prefetching some information that will likely be accessed soon to the local machine, there will be no transmission delay when the user actually requests them. The perceived latency can then be reduced to that for fetching information from local disk or file servers on local nets.

Prefetching is only effective when future access pattern can be determined. The key challenge in prefetching is to determine "what to prefetch, when to prefetch, and how much to prefetch" so that performance will be enhanced. The difficulty of realizing efficient prefetching lies in the fact that it is impossible to predict exactly what a user is going to need and hence some prefetched files may never be used. In mobile network environments, link bandwidth is a very scarce and high cost resource; prefetching information based on inaccurate prediction which are not used by the user will waste precious bandwidth. For the overall system, applying a prefetching scheme may increase the network load, because prefetching can cause downloading files that are not used. Therefore it degrades the whole system performance. So, for prefetching in network applications, there is tradeoff between user perceived latency and bandwidth. For applications with an obvious access pattern, a prefetching scheme is suitable. For applications with an arbitrary access pattern, a prefetching scheme is not suitable.

The prefetch problem has two aspects: the first one i s how to estimate the probability of each file being accessed in the near future. In general, this probability changes with time as the user issues new requests. The second aspect is how to determine which files to prefetch such that the overall system performance can be optimized relative to some criteria. The performance of a prefetch scheme is measured by the following criteria:

1. Hit rate: refers to the percentage of user requests satisfied by the prefetched files. Generally, the higher the hit rate, the more reduction in user -perceived latency is achieved.
2. Successful prediction rate: the probability of a prefetched file being used eventually. A high successful prediction rate indicates that less bandwidth is wasted due to prefetching unused files.

Information on user's future access may be derived from server's access statistics, client's configurations, application's nature, user's personal preference, and user's planning tools.

In the following we review various existing prefetch techniques. These techniques are mainly targeted for Internet web access, but since the World Wide Web is a very common and typical network application, these techniques can be applied to other network and mobile computing applications based on the features of the techniques and the applications.

3.2.1 Statistical Prefetching vs. Deterministic Prefetching

In, statistical and deterministic prefetching schemes for Internet Web access are discussed. In a statistical scheme, the interdependence of web page accesses are calculated periodically based on the most recent access logs; web pages with interdependencies higher than a certain threshold are grouped for prefetching. Statistical prefetching can potentially have wide applications as the process can be easily automated. However, by its speculative nature, some bandwidth will be wasted, thus it can increase total bandwidth consumption. There is a general tradeoff between bandwidth and latency here. If we reduce the threshold for statistical prefetching, the latency may improve, but at the price of increased bandwidth consumption. Unfortunately, bandwidth is still a scarce resource in most networks, particularly over wireless links and long distance paths. Thus statistical prefetching has to be used with great care to avoid bandwidth waste. Note that, although the perceived delay for prefetched files are very low, the retrieving delay for non -prefetched files may actually increase as a result of the extra traffic load caused by prefetching. When traffic is heavy, aggressive prefetching, such as "get all links", may actually increase the average latency of all accesses. It is concluded in that, unless the traffic is very light or the prefetching efficiency is very high, statistical prefetching may not necessarily reduce perceived latency.

In a deterministic scheme, prefetching is configured statically by the users as part of their personalized user interface, or even by page designers as part of the content design. Deterministic prefetching is the most conservative types as it often has little or no bandwidth overhead, although its scope of use is limited. Nevertheless, when users know what needs to be prefetched, it can reduce perceived latency, and to some extent, even ease congestion at very little cost. Some potential uses of deterministic client-initiated prefetching are discussed. The following are some of the examples:

1. Batch Prefetching: Web pages that are read on a regular basis, such as on-line newspapers, weekly work reports, can be prefetched in a batch mode during the less busy period.
2. Start-up prefetching: when a browser is started, a set of pages users need to look at that day may be prefetched in the background.
3. Pipelining with prefetching: for some information services where users can easily specify the sequence of pages to be viewed, such as on -line newspapers, stock market prices, and headline tracking services, we can potentially pipeline the operations by fetching the next page while the user is looking at the current one.

3.2.2 URL Graph Based Prefetching vs. Context-Specific Prefetching

Prefetching algorithms used in today's commercial products are either blind or user -guided. A blind prefetching technique does not keep track of any user web access history, it prefetches document based solely on static URL relationships in a hypertext document.

User-guided prefetching techniques are based on user -provided preference URL links.

Research in prefetching is targeted at intelligent predictors based on user access patterns. A popular approach is to represent the access pattern as a URL graph. The URL graph based approaches are demonstrated effective for documents that are often accessed in history. However, they are unable to pre -retrieve those documents whose URLs are never touched before. A new approach, context -specific prefetching, is proposed in, which overcomes this limitation. It relies on key words in anchor texts of URLs to characterize user access patterns and on neural networks over the keyword set to predict future requests.

3.2.3 Real-time Online Adaptive Prefetching vs. Off-line Prefetching

In, a general prefetching scheme is describe for real -time online web access.

We refer to it as an adaptive network prefetch scheme. The adaptive network prefetch scheme comprises a prediction module and a threshold module. The prediction module computes the access probability, which is defined as the conditional probability of a file being requested by the user in the near future, given the file currently being used. The threshold module computes a prefetch threshold for the information server based on network and server conditions as well as the costs of time and bandwidth to the user, such that the average delay is guaranteed to be reduced by prefetching files as long as their access probability is greater than its server's prefetch threshold. The prefetch threshold is a function of current system condition and certain cost parameters. The prefetching scheme works in the following way: Basically, whenever a new file is requested, the prediction module updates the local access history, if needed, and computes the access probability of each file somehow related to the current file. At the same time, the threshold module computes the server's prefetch threshold. Finally all the files with access probability greater than the server's threshold are prefetched.

This prefetch scheme is a general one that may be applied to almost any network application to decide what information to prefetch. For specific network information access application, the problem of applying this prefetching scheme is to decide the prediction algorithm and the algorithm to compute the server threshold. Also described a prefetching scheme to deal with disconnectivity of mobile clients. This scheme allows users to prefetch a group of files together for the user who is about to disconnect from the network. In this case, the prefetching threshold is not applicable. The prediction module is still used. At first the user specifies an upper bound for the bandwidth cost or the amount of data to be downloaded. The user also specifies some initial files. The system starts t o download files. First initial files are downloaded, after downloading each file, access probabilities of other files are calculated, and the file with the highest access probability is downloaded. This process continues until the total bandwidth cost or the total amount of data downloaded exceeds the user specified limit. More sophisticated functions can be added to select files more intelligently. Research has shown that prefetching is a good approach to reduce latency for network applications. However, it must be implemented in such a way that the overall system performance can be improved, and the tradeoff between saving user's time and bandwidth usage must be considered.

4. DESIGN CLIENT/SERVER MODEL

In the designing, we develop a mobile computing system. The first benefit of this work is that this is an interesting mobile computing system. The second benefit is that it can be used as a base to study the effect of various prefetching and caching techniques on the performance of the application.

This Design is organized as follows. Section 4.1 gives a description of the system. Section 4.2 covers the system software design and architecture. The features of the system are analyzed. The major components and their cooperation relationship are discussed.

4.1 System Description

The Mobile is a client/server network system running on a user's handheld computer and an Internet server. When a user is on the move (traveling, driving, exploring the nature), with a computer at hand and with the computer connected to the Internet server via a wireless network, also with a GPS device integrated in the computer, he will always see where he is and where he is heading to. He will never worry about getting lost. It downloads the object segment around the user's location from the Internet server, and displays the user's moving trace and the map segment on the computer screen. The user can also choose to view the object at different resolution levels. Whenever the user changes the object resolution level or moves out of the current object segment, the client detects the change and automatically downloads the next object segment and displays the user trace and the new object segment.

4.2 System Analysis and Design

Object oriented analysis and design approaches are used to develop this system.

4.2.1 Client/Server Model

As described in Section 4.1, the Mobile is a client/server network system. The client resides in a mobile computer (handheld, palm...) featuring small memory, small storage, small display, slow performance, slow network bandwidth, less functionality, and less user friendly input method. The server resides in a large computer or workstation in the Internet that has large memory, large storage, and fast processing ability. The following figure shows the Mobile client/server model.

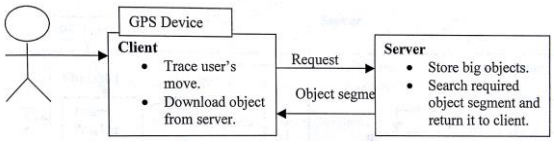

Figure 4.1: Mobile Client/Server Model

In this paper, we concentrate on a single server and a single client model. Big objects of different resolution levels are stored in the server. The server listens to client requests, searches and processes the required object segment, and returns it to the client. The client traces a user's move with the assistance of the GPS system, and requests relevant object segments from the server. Upon receiving the requested object segment, the client displays the object segment and user trace on the hand held computer screen. Whenever the user moves out of the current object segment or the user requests a new resolution level, the client issues a new request for a new object segment.

4.2.2 Main Components and System Architecture

The Figure 4.2. Shows the main components and architecture of the system.

(a) Basic Transmission units: Request and ObjectUnit are the two basic data units transmitted between client and server. A request contains user location (latitude, longitude), object resolution level (0 - 4) and user moving direction (0-NE; 1-NW; 2-SW; 3-SE). An ObjectUnit contains a small object segment and the coordinates (latitude and longitude) of the top-left point and bottom-right point of the object segment.

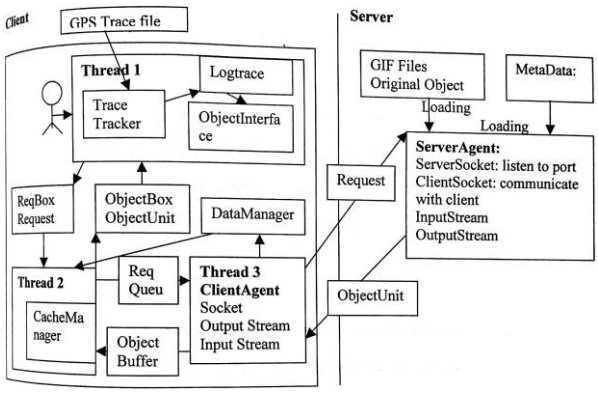

Figure 4.2: Main Components and Architecture of Mobile System

Main components at the client side:

(b) ObjectInterface: This is a GUI user interface that resides in the client. It contains a canvas for displaying the object and user trace and a panel containing three buttons: ZoomIn, ZoomOut, and Exit, which allow the user to view a object at different resolution levels or exit the application at any time. The display screen size is 160 pixels*160 pixels. The displayed object size is 160 pixels*150 pixels.

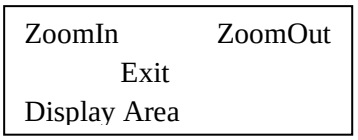

Figure 4.3: GUI of the Mobile System

(c) TraceTracker: It traces user movement (trace of locations) and returns the user's next location.

(d) CacheManager: It is the storehouse of object segments at the client side. Upon receiving a new request from ObjectInterface, it searches its object cache for the requested object segment, if found, send it to the ObjectInterface, otherwise, pass the request to the client agent to retrieve it from the server on the Internet.

(e) ClientAgent: Its responsibility is to communicate with the server, pass requests to the server, and receive requested object segments from the server.

Main components at the server side:

(a) ServerAgent: It is responsible for the communication with the ClientAgent. When receiving a request from the ClientAgent, it passes the request to the server to search for the required object segment and sends it to the ClientAgent.

(b) Server: It is responsible for loading and storing all initial big objects. When receiving a request from the server agent, it searches and fetches the required object segment and passes it to the ServerAgent.

4.2.3 Multi Thread Programming

Three threads are implemented in the system. Thread1 traces the user's movement and displays the user trace and object on the screen. It also informs Thread2 of new requests whenever the user moves out of the current object segment or the user requests a new object resolution level.

Thread2 manages the object cache, responds to Thread1's requests and stores object segments received from Thread3. It listens to Thread1, whenever a new request comes, it searches the object cache for the required object segment, if found, it returns the object segment to Thread1, if not found, it passes the request to Thread3. It collects object segments received from Thread3 and puts them in the object cache. Caching and prefetching schemes are implemented in Thread2.

Thread3 is responsible for communication with the server, fetching required object segments from the server and passing them to Thread2. Thread1 and Thread2 communicate through a request box and an object box.

Requests pass from Thread1 to Thread2 through the request box. Object segments pass from Thread2 to Thread1 through the object box. Thread2 and Thread3 communicate through a request queue and an object buffer. Requests missed from the cache and generated by the prefetching are put in the request queue by Thread2. Thread3 fetches requests from the request queue and retrieves object segments from the server. The received object segments are put in the object buffer. Thread2 collects the object segments in the object buffer.

The following figure shows the three-threaded architecture of the system.

This three -thread implementation is to insure the real time feature of the application. Thread1 continuously tracks the user's move and displays the user trace and object (if available) while Thread2 and Thread3 are searching and retrieving required object segments.

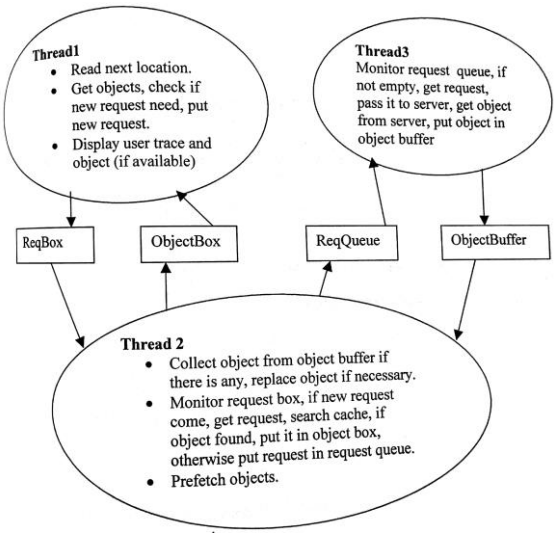

Figure4.4: Three–threaded Architecture of the System

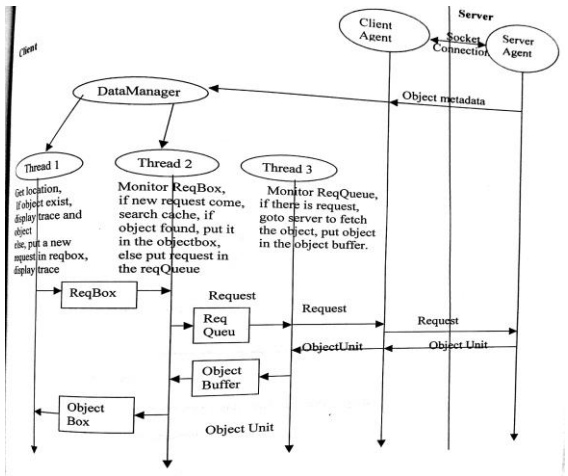

Figure4.5: Sequence diagram of Client / Server Model

4.3 - Server Algorithm:

Load in objects
Establish socket connection between ServerAgent and ClientAgent
Send object meta data to client
While true
{
Get request from client
If (request=End) exit;
Else
{
Search object segment
Send object segment to client
End if get stop signal from client
}
}//end of while

4.4 -Client Algorithm:

Establish socket connection between ClientAgent and ServerAgent
Receive object meta data from server
Thread1 start running
Thread2 start running
Thread3 start running

Thread1 Algorithm:

get location; //from log data
update logtrace; //user trace
get direction; //from loctrace
form a new request and put it in ReqBox;
get object segment frame; //from DataManager
display trace;
while(true)
{
get location //from log data)
if (end of file) put End request in ReqBox, exit;
update logtrace; //user trace
get level; //from user input)
get direction; //from loctrace
form a new request;
if request matches current object segment frame
{
check ObjectBox
if required object arrived, display object and trace
else
display trace

```
}
else //a new request
put new request in the ReqBox.
}//end of while
```

Thread2 Algorithm:

```
Get the first request from ReqBox;
Put the request in ReqQueue;
Execute prefetching;
while( true)
{
Collect object  segments from ObjectBuffer if there is any;
If cache full, replace object;
check ReqBox
if (no new request)
{
check if current request has been met
 if not
 {
search cache,
if required object  found, put it in ObjectBox.
 }
}
else // new request arrive
{
if (request=end), put End request in ReqQueue, exit;
else
{
clear ReqQueue;
search cache for required object segment,
if found, put it in ObjectBox;
else put request in ReqQueue.
Execute prefetching;
}
}
}//end of while
```

Thread3 Algorithm:

```
while( true)
{
if there is request in the ReqQueue, get request
if (request==end)
send End request to server, exit;
else
{
send request to server
receive object from server
put object unit in ObjectBuffer
} }// End of while
```

5. PROBLEM ANALYSIS

Due to low wireless bandwidth and slow processing ability, a user may suffer long latency. Our study objective is to find techniques to improve the application performance and at the same time efficiently use the precious wireless bandwidth.

User perceived latency is the time interval from the time a user is sending a request until the time the required object is displayed on the screen. First we analyze the factors of user perceived latency. In the mobile application, big objects are stored in the server as Image objects. Each time when the client requests a new object segment, the server filters out the required object segment from its big objects, encodes the object segment into GIF data, and sends it to the client. At the client side, the received GIF data is converted into an Image object and displayed on the palm screen. The user perceived latency is mainly composed of three factors: object transmission time, object processing time and object displaying time.

Object transmission time is the time interval to transmit the object GIF data of an object segment from server to client.

Object processing time is the time interval to decompress object GIF data of an object segment and convert it into an Image object.

Object displaying time is the time interval to display an object segment on the computer screen.

The object displaying time is very short and can be neglected. The transmission bandwidth between client and server is low which causes long object transmission times, and the process for the client to decompress and convert GIF data into an image object takes a long time. Therefore users experience long latency to see the required object segment. Finding good techniques to reduce user perceived latency and at the same time, efficiently use wireless bandwidth is the objective of our study.

5.1. Caching:

Within the memory limit, cache as much as possible, replace the object segment, which is efficient to the current user location when necessary.

At the client side, object segments received or prefetched from the server are stored in the object cache. Based on

the memory space of the computer and object segments size, we set the maximum object cache size to be 50 object units. Within this limit, we cache as much as possible. When the object cache is full and there is a new object segment, we choose to remove the object segment in the cache that is efficient to the current location and to make room for the new object segment.

5.2. Proposed Cache Replacement Technique:

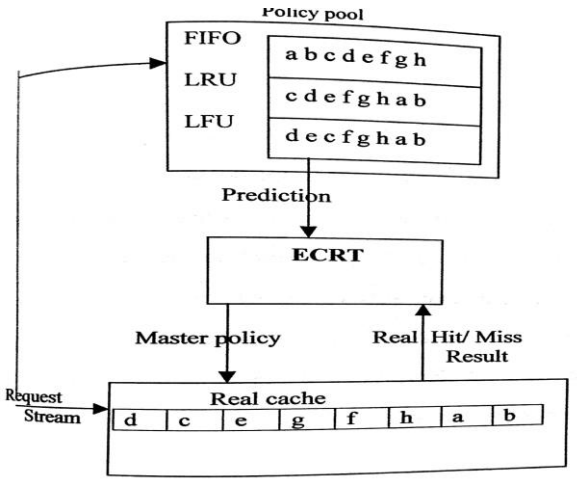

Figure 5.1: Design of ECRT: Voting Mechanism

Cache management divides the cache into static partitions and lets a few successful policy works in separate partitions. CacheManager make predictions on whether objects should be cached or replaced. A Voting of these predictions defines the master policy that manages the real cache. We have two separate Policy Pools (PP), PP1 and PP2. Each pool has the same set of policies all with equal cache sizes which are also equal to the physical cache size.

The first sets of policies, those in PP1, are only used for voting purposes. All policies in PP1 directly observe the request stream and may choose to keep metadata for different objects.

The second sets of policies, those in PP2, are used to cooperatively act as the *master policy* that governs the physical cache space. The policies in this pool only keep metadata for those objects in the physical cache, but are allowed to order their metadata independently. Since all policies in PP2 were assumed to have the same cache size, the set of objects cached at this period will also be exactly the same. Whenever the physical cache is full, some objects need to be replaced from the physical cache to make room for the incoming ones. We replace from the physical cache by the rule of this policy.. Successful policies have larger votes and are thus more likely to be selected as the policy that will govern the physical cache at any given instant. The selected policy indicates which object or objects should be ejected and then all the other policies in PP2 obey its choice releasing the record of the selected objects from their queues.

1. **Physical cache** = Physical memory where real data for objects is stored
2. **Policy pool (PP)** = a policy that gives an ordering of objects by using only the headers of objects
3. **PP1 (Policy Pool 1)** = the set of Policies where each Policy orders objects seen in the request stream
4. **PP2 (Policy Pool 2)** = the set of Policies where each Policy orders objects kept in the physical cache

5.3. Proposed Algorithm:

1. Initialize Counter for different policies.
2. repeat following steps.
3. while (page replacement required)
4. apply voting mechanism.
5. find vote.
6. if(vote=majority)
7. {
8. replaced that suggested page.
9. update counter of majority.
10. }
11. else if(page replaced)
12. exit
13. else
14. check History.
15. select page with maximum counter value and replaced that page or if counter is same randomly select page for replacement.
16. update counter value of that policy.
17. exit.

5.4. Mobile System Performance Criteria:

To study the effects of the chosen techniques on the mobile application, we must first decide how to measure the effect of each technique and the performance of the application. We consider the following criteria: *user perceived latency, request meet rate, cache hit rate, and wireless bandwidth efficiency.*

- User perceived latency (UPL): time interval between the client sending a request until receiving the required object – measures how fast a user's request is met, the shorter the better.

- Request meet rate (RMR): ratio of the number of object displayed to the number of total requests – measures how many user requests are met, the bigger the better.

RMR = (objectsdisplayed / totalrequests)

- Cache-hit-rate (CHR): ratio of the number of requests met in cache to the number of total requests – measures how efficient the caching and prefetching algorithms are, the bigger, the better.

CHR= (cachehits / totalrequests)

- Bandwidth efficiency (BE): ratio of the number of object units used to the total number of object units fetched from server - measures how efficiently the wireless bandwidth is used, the bigger the better.

BE=(objectsdisplayed / objectstransmitted)

6. EXPERIMENT RESULT

6.1. Client memory space, object segment size and object cache size :

Client program memory space is 220 Kbytes. The program takes about 50 kbytes. Within this restriction, how to set the object segment size has a fundamental effect on the experiments. Initially, we set the object segment size to be the full screen size that is 160*150 pixels (excluding the buttons panel). But the memory can hold at most two object segments of full color and of size 160*150 pixels, the user perceived latency of such a object segments is about 5 minutes and most of the time the user will not see the required object because he has moved out of the current object boundary before the object segment is retrieved. And hence there is no way to test any prefetching and caching schemes.

6.2. Transmission Bandwidth between Client and Server

We estimate the transmission bandwidth between client and server by transmitting a certain amount of data from the server to the client and recording the transmission time.

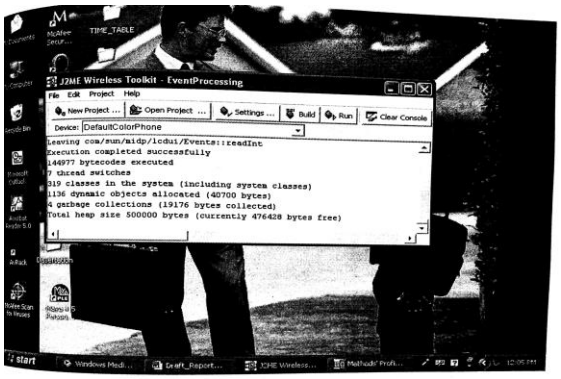

Figure 6:1 J2ME Wireless Toolkit and MIDP

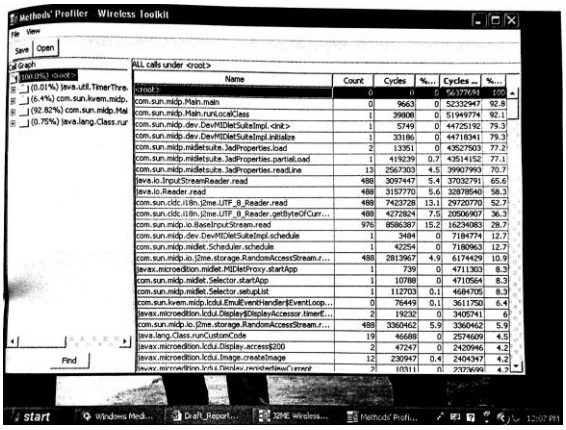

Table 6.1: Execution Time and Frequency use of the method root.

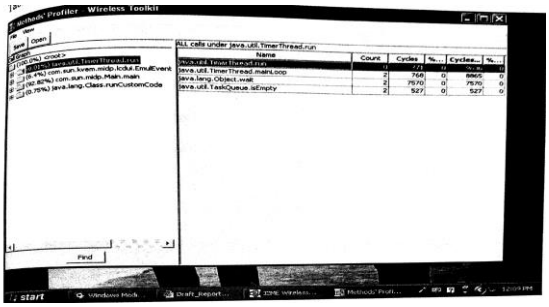

Table 6.2: Execution Time and Frequency use of the method java.util.timer.run.

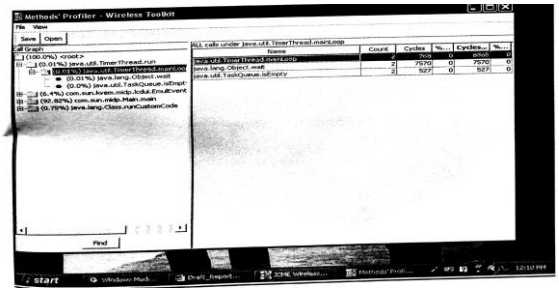

Table 6.3: Execution Time and Frequency use of the method java.util.TimerThread.mainLoop.

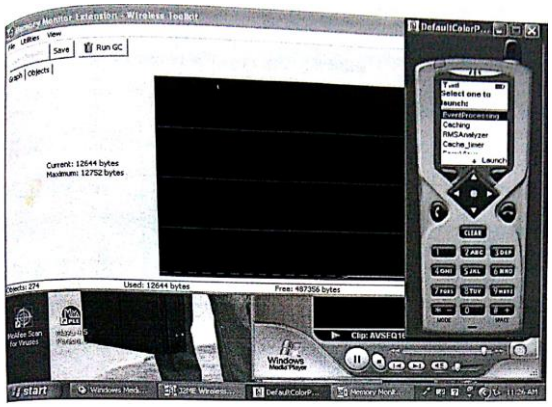

Figure6.4: Memory Monitor Graph

Stage	Current Memory	Maximum Memory	No. Of Objects
Start up	12248	12356	263
Event Processing	11844	13008	188
Char S	12516	13008	296
Char A	12412	13008	292
Char N	12464	13008	294
Char J	12412	13008	292
Char A	12412	13008	292
Char Y	12516	13008	296
Save	12496	13008	293

Table 6.4: Memory Usage at different stages

Text: SANJAY

Stage	Current Memory	Maximum Memory	No. Of Objects
Start up	9548	9656	197
Event Processing	10248	12024	226
Char S	9816	12024	230
Char A	9712	12024	226
Char N	9764	12024	228
Char J	9712	12024	226
Char A	9712	12024	226
Char Y	9816	12024	230
Save	9796	12024	227
Exit	9576	12024	221

Table 6.5: Memory usage at without caching scheme.

Stage	Current Memory	Maximum Memory	No. Of Objects
Start up	9968	10076	208
Event Processing	10116	11892	235
Char S	10304	11892	243
Char A	10148	11892	243
Char N	10200	11892	237
Char J	10148	11892	237
Char A	10148	11892	237
Char Y	10252	11892	241
Save	10232	11892	238
Exit	10564	11892	234

Table 6.6: Memory usage at caching scheme

Stage	Current Memory	Maximum Memory	No. Of Objects
Start up	9960	10068	208
Event Processing	9556	11332	233
Char S	10328	11332	244
Char A	10140	11332	237
Char N	10192	11332	239
Char J	10140	11332	237
Char A	10140	11332	237
Char Y	10244	11332	241
Save	10224	11332	238
Exit	10556	11332	234

Table 6.7: Memory usage at voting scheme

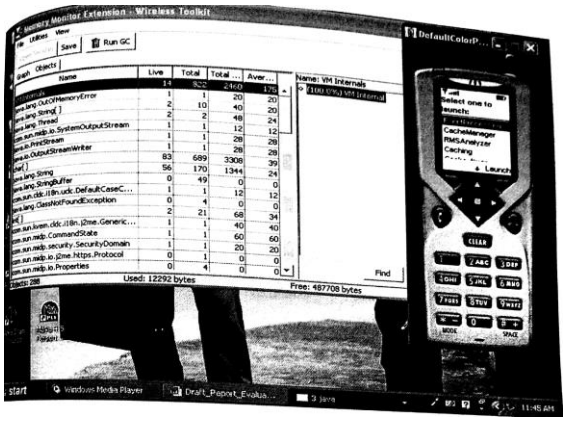

Table 6.8: Memory usage of Virtual Memory internal class.

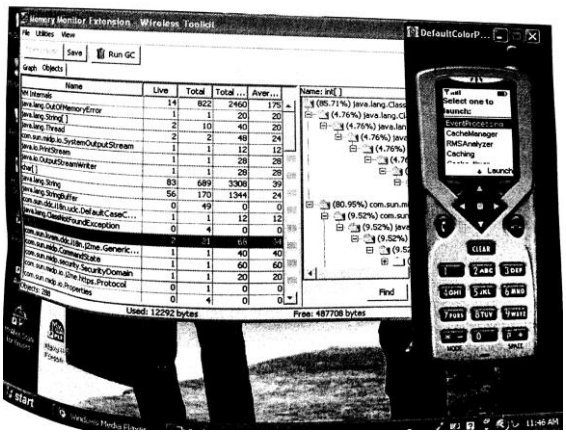

Table 6.9: Memory usage of int[] class.

7. CONCLUSION

It is generally true that cache replacement scheme and prefetching schemes and integrated them in the mobile application. It was found that the caching only approach gives a better solution in both user satisfaction and wireless bandwidth utilization. This is based on the fact that the user trace file we are using is in a random roaming pattern. It is expected that the experimental results will be closely related to trace file pattern and features.

Different- criteria policies are more successful than single- criteria policy at deciding which objects to cache.

REFERENCES

[1] M. Vahabi, M. F. A. Rasid et al., "Adaptive Data Collection Algorithm for Wireless Sensor Networks", International Journal of Computer Science and Network Security, VOL.8 No.6, June 2008.

[2] et al. Vijay S. Kale and Rohit D. Kulkarni, "An Overview on Wireless Sensor Networks Technology and Simulation Software's", International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 5, May 2016.

[3] et al. Samuel Madauda and Paluku Kazimoto, "Analysis Of Concepts Of Wireless Sensor Networks", July 2014, Vol. 5, No.3 ISSN 2305-1493, International Journal of Scientific Knowledge Computing and Information Technology, Vol. 5, No.3, July 2014.

[4] I.Ari. Amer, E.Miller, S.Brandt, and D.Long. Who is more adaptive? ACME: Adaptive caching using multiple experts, In workshop on Distributed Data and Structures(WDAS 2002), Paris France, Mar, 2002

[5] Robert B. Gramacy, Manfred K. Warmuth, I.Ari.Adaptive caching by refetching

[6] W. Dargie and C. Poellabauer, Fundamentals of wireless sensor networks: theory and practice. Wiley. com, 2010.

[7] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," Communications of the ACM, vol. 43, no. 5, pp. 51–58, 2000

[8] G. Asada, A. Burstein, D. Chang, M. Dong, M. Fielding, E. Kruglick, J. Ho, F. Lin, T. Lin, H. Marcy, et al., "Low power wireless communication and signal processing circuits for distributed microsensors," in Circuits and Systems, 1997. ISCAS'97., Proceedings of 1997 IEEE International Symposium on, vol. 4, pp. 2817–2820, IEEE, 1997

[9] J. M. Kahn, R. H. Katz, and K. S. Pister, "Next century challenges: mobile networking for Smart Dust," in Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, pp. 271–278, ACM, 1999.

[10] M.Satyanarayan, "Mobile Computing", IEEE Computer, Vol.26,No.9,Sept.1983,p.81-82.

BIOGRAPHY:

Dr. Sanjay Kumar is working as a Professor in the Department of Computing Science and Engineering, Galgotias University Greater Noida India. He obtained his Ph.D (Computer Science and Engineering) in 2015 under faculty of engineering, M.M University, Mullana, Ambala, MTECH (CSE) in 2005 from University School of Information Technology, GGSIP University Delhi and MIS (Computer Science and Engineering) in 2002 from Dr.B.R.Ambedkar University Agra. His research area includes Biometric Security, Big Data, Data Analytics, Software Engineering, Wireless Communications, Mobile Ad hoc & Sensor based Networks and Network Security.

Dr. Sandhya Umrao is working under the department of Information Technology in Galgotia College of Engineering and Technology Greater Noida (U.P.) since July 2007. She obtained her Ph.D (Computer Science and Engineering) in 2018 under faculty of engineering, M.M University, Mullana, Ambala, M.Tech. (Computer Science and Engineering) From Kurukshestra University Kurukshetra (Haryana) in 2007. She supervised 2 M. Tech students. Her research area includes Wireless Sensor Networks, Reliability Theory, Artificial Intelligent and Cryptography.