A NOVEL THIN CLIENT ARCHITECTURE WITH HYBRID PUSH-PULL MODEL, ADAPTIVE DISPLAY PRE-FETCHING AND GRAPH COLOURING

Sumalatha.M.R[1] Sridhar S[2] Satish G[3]

Department of Information Technology, Madras Institute of Technology, Anna University, Chennai

{sumalatha.ramachandran,ssridhar2802,grn.satish}@gmail.com

ABSTRACT

The advent of cloud computing has driven away the notion of having sophisticated hardware devices for performing computing intensive tasks. This feature is very essential for resource-constrained devices. In mobile cloud computing, it is sufficient that the device be a thin client i.e. which concentrates solely on providing a graphical user interface to the end-user and the processing is done in the cloud. We focus on adaptive display virtualization where the display updates are computed in advance using synchronization techniques and classifying the job as computationally intensive or not based on the complexity of the program and the interaction pattern. Based on application, the next possible key-press is identified and those particular frames are pre-fetched into the local buffer. Based on these two factors, a decision is then made whether to execute the job locally or in the cloud or whether we must take the next frame from the local buffer or pull it from server. Jobs requiring greater interaction are executed locally in the mobile to reduce interaction delay. If a job is to be executed in the cloud, then the results of the processing alone are sent via the network to the device. The parameters are varied in runtime based on network conditions and application parameters to minimise the interaction delay.

KEYWORDS

Cloud Computing, Mobile devices, Thin clients, Virtualization, Remote display

1. INTRODUCTION

The user's perspective on what can be done with a mobile device is changing. Not long ago, the mobile devices were used just for the purpose of making phone calls and sending messages. Users now need the complete graphic rich content rendered in their mobile device giving them just as much experience as they would get in a fixed device like PC. The processing powers of servers are increasing according to Moore's law and the bandwidth of wireless links have also improved with technologies like 3G, LTE. This has led to the development of a number of thin client solutions. A thin-client computing system consists of a server and a client that communicate over a network using a remote display protocol. The protocol allows graphical displays to be virtualized and served across a network to a client device, while application logic is executed on the server. Using the remote display protocol, the client transmits user input to the server, and the server returns screen updates of the user interface of the applications from the server to the client. The thin client remote computing displays are expected to be responsive to the clients as if they are local machines. However, the complicated graphical interfaces and multimedia applications present technical challenges to thin client developers for achieving efficient transmissions with low bandwidth links. The main concern however though is the Interaction Latency.

DOI : 10.5121/ijasuc.2012.3305

Table 1. Requirements

Functional Requirement	1. The client should be able to access his resources in the cloud. 2. The display rendering should be done in the server side. 3. Access control mechanisms must be setup.
Non-functional Requirement	1. Interaction delay must be less than 2 ms. 2. Frame rate shouldn't drop under low bandwidth conditions.
Hardware Requirement	Server side: 1. Multi-core processors supporting virtualization Client side: 1. Relatively thin clients suffice. 2. It should be able to access the internet.
Software Requirement	Server side: Hypervisors:, vmware. Monitoring: ganglia. Client side: Thin client session software.

Mobile and cloud computing have emerged as the new computing platforms and are converging into a powerful cloud mobile computing platform. In a virtualized screen, screen rendering is done in the cloud, and delivered as images to the client for interactive display. This enables thin-client mobile devices to enjoy many computationally intensive and graphically rich services. Hence, one can access even the most demanding applications in the cloud from intrinsically resource-constrained mobile devices by physically separating the user interface from the application logic. Table 1 gives the general requirements of the thin mobile client system.

2. RELATED WORK

Reference [1] summarizes the solutions that have been proposed to tackle the main issues associated with remote display systems such as battery life time, wireless bandwidth availability. Optimal selection wireless network interface card sleep times to maximize the energy efficiency in thin clients have been studied [2]. Motion based differential encoding for transmitting only the essential information over the limited available wireless bandwidth has been studied [3]. Reference [5] ensures availability of virtual resources by immediate instantiation of VMs in a resource rich server or cloudlet accessing over wireless LAN. Reference [6] discusses the rendering of graphical intensive content in thin clients with end to end streaming, rate control policies and buffer management mechanisms. Reference [7] proposes a system where a cluster of PCs, equipped with accelerated graphic cards managed by Chromium software, is able to handle remote visualization sessions based on MPEG video streaming involving complicated 3D models. Reference [8] virtualizes the entire personal environment on server. They have also added diverse functions intended to adjust the balance of load on virtual PCs, as well as improve operability.

Limitations with existing system:

The operational time of mobile devices is often limited when extensively used. These battery capacity shortcomings result in short recharge cycles and refrain users from relying completely on their mobile device. Over the last decade, the advances in nominal battery capacity have been modest. Consequently, extending device autonomy should primarily be realized by making the device itself more energy efficient.

Compared with fixed access networks, bandwidth availability on modern broadband mobile and wireless technologies is limited, variable and expensive. Typically, UMTS users receive up to 384 kbps, practical throughputs of 347 kbps for LTE and up to 6.1 Mbps for WiMAX. Moreover, the actual throughput will vary due to user mobility and interference and fading effects. Besides technological limitations, economical considerations drive the demand for highly efficient remote display compression technologies. More and more, users are confronted with volume based subscription plans and hence will not tolerate any redundant byte to be sent on the network.

Interaction latency, i.e. the delay a user experiences between generating some user input and having the result presented on his display, is key challenge of mobile cloud computing. Whereas bandwidth limitations are likely to disappear with technological advancements, interaction latency is an intrinsic key challenge of mobile cloud computing because even the most trivial user operations need to be communicated to the server.

3. SYSTEM DESIGN

3.1. Overview of the Thin Client Architecture

The system is architected as a coordinator-assisted server cloud, comparable to systems deployed today by infrastructure service providers. The overall architecture of the system is depicted in Figure 1. It is composed of a coordinator, a group of clusters comprised of nodes connected in a LAN, a storage server infrastructure within a cluster, and a number of external, heterogeneous clients through which users access the system. The coordinator acts as a broker that allocates the best cluster based on the requests from clients across the Internet. The back-end compute servers host completely virtualized environments within which the computing sessions of our users run. The network storage server infrastructure is used for all persistent file storage. The clients are merely inputting and outputting devices connected to the cloud providers across the Internet.

Users interact with our sessions through a thin-client session viewer, a simple device or application that relays the user's input and the session's output between the client and the server through a secure channel. Each user in the system is issued credentials by the coordinator to connect to the cloud. The user gets a complete set of operating system resources. The cloud setup is multitenant and the resources and data of each of the clients are isolated from each other. Multiple users can share sessions with credentials. This gives the ability to access remote sessions of any client from anywhere. Sharing of data can also be done by copying the data into the respective user sessions.

The thin clients behave just like the normal clients but the display they must render is computed in the cloud. The client software just transfers the user input to the cloud. Thus in the client side the display is virtualized and in the cloud side, the user input is virtualized and behaves as though the user actually keyed it in that system. Hence, it is easily compatible with the existing systems and applications needn't be modified.

The system follows a push pull model. The clients can request for the next display update from the server (pull) and the server can also compute display updates in advance and stream it to client (push). Pull is used in interactive processes and push is used when idle time during interaction is more. Pull methodology is generally not used much because they add to the additional network latency.

In the thin client architecture, a mechanism is devised to minimise the interaction latency due to network limitations like computing display updates in advance.

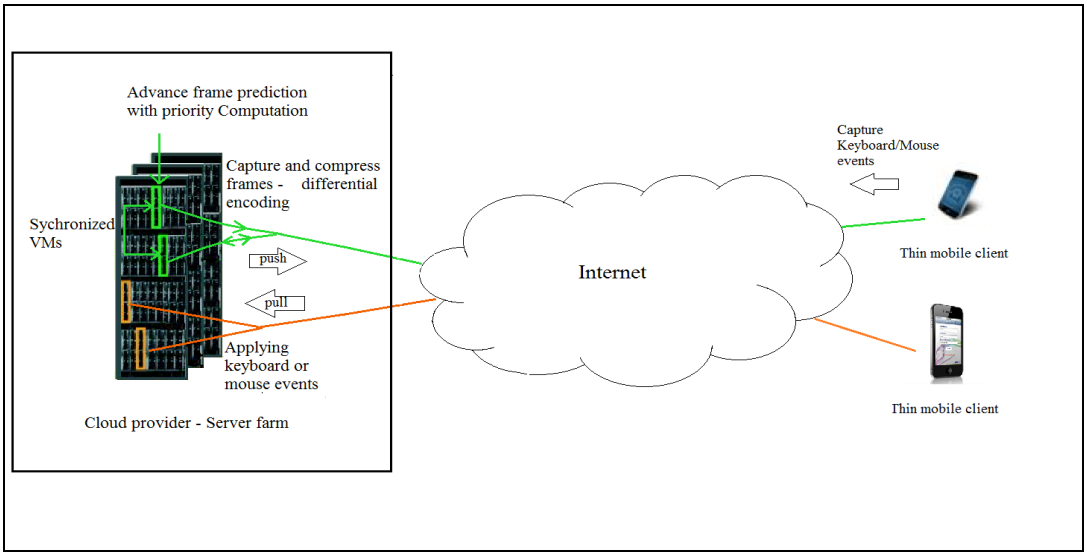

Figure 1. Architecture of coordinator assisted Mobile cloud computing system

3.1. Methodology Proposed

The system uses a display update pre-fetching mechanism similar to [6] to minimize the interaction latency in the system. The idea is to have all possible states that the user can go to during the next step available on the client side. This makes use of the fact that, the idle time in the client side is generally high and the updates can be conveniently pushed into the thin client. At the server side, the application decides what display states will be requested to the rendering engine based on the application metadata. For example if there are 'n' possible key presses all possible display updates are computed by the cloud and the frames are sent. Only the first frame is sent fully, and the successive frames are sent only in the form of differences from the previous frame since there won't be significant differences in adjacent states. The server side components will also issue priorities to each of these states and higher priority states are buffered first. A graph is constructed in the server side with sophisticated machine learning techniques to predict the next key press and assign a high priority to it. The neighbour of any node in a graph is the next possible state to which we can move from the current state. The states are numbered by a sequence number so that they needn't be transmitted again in case it is present locally in the buffer in the client side.

The system employs a job classification scheme with which, small jobs are processed locally, and computationally intensive tasks are submitted to the cloud. The rendering and display are also adaptive. If the bandwidth of the wireless link is low, then the

resolution is scaled down having fixed frame rate instead of having high resolution and lower frame rates.

We have also applied Graph colouring algorithm for allocation of the pre-fetched frame in buffer. Buffer allocation is extremely important as the gap between memory latency and network latency widens. We construct a graph such that every vertex represents a unique frame in the thin client. Interference edges connect pairs of vertices which are live at the same time, and preference edges connect pairs of vertices which are involved in move instructions. Frame allocation in the buffer can then be reduced to the problem of K-coloring the resulting graph, where K is the number of chunks of memory available on the target architecture where a chunk represents the maximum possible size of frame. No two vertices sharing an interference edge may be assigned the same color, and vertices sharing a preference edge should be assigned the same color if possible. As graph coloring in general is NP-complete, so is frame allocation.

Figure 2, shows the sequence of steps involved in client and server side. Based on the user input, if the next frame exists in the buffer, it is taken from the local buffer else retrieved from the server. In the server side, the display updates are captured, encoded and sent to the client as shown.

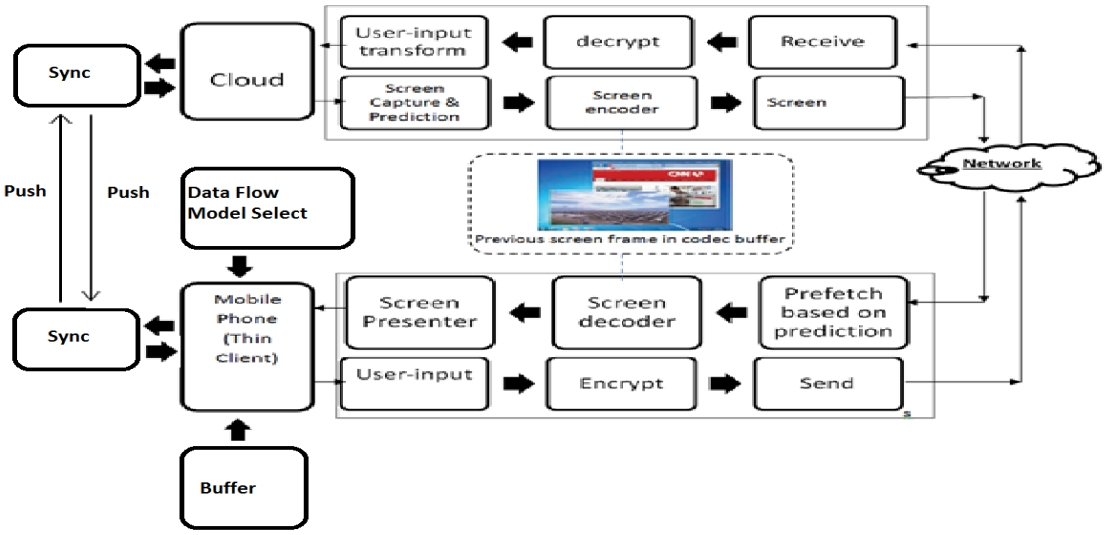

Figure 2. Flow diagram indicating the sequence of steps involved in the cloud provider and the client side.

Server:

Module serverSidePush Input: user session request. Output: Pushing updated data to client thereby keeping the client synchronized with the cloud. Module priorityCompute Input: Current Job j Output: Highest priorityKeypress Lookup application metadata or learn next possible keypress from the user to compute highest priority keypress

```
Module: ContentEncoder
Input: Two successive frames.
Output: Encoding of second frame.
    1. Check for differences between two successive frames.
    2. If( there is no difference between two frames )
Skip the frame from transmitting.
3.      Else
Use Block classification and entropy coding to localize the different block and transmit that
block alone with the corresponding frame sequence number.

Module: UserInputTransform
   Input: key press interrupt
   Output:TransformedInput
   Map the entered input with the corresponding transformed input from the lookup table.
```

Client

```
Module: Job submission agent
Input: Job j
Output: 0 if job not submitted successfully
        1 if job is in the local execution queue.
        2 if job is submitted to the cloud.
    1. If(jobqueue full)
Wait()
        Else
Classify the job as computationally intensive or less intensive
If(job = intensive)
        Execute job locally.
Else
        Dispatch the job to the cloud.


Module: DisplayUpdate
Input: Job j
Output: Rendering of the frames on client's device

1. If dataFlowModel == PULL
                If(prefetched frame available)
                        Render it in the client screen.
  Else
        Request the server for new frame.
2. If dataFlowModel == PUSH
        If(message available in push queue)
          Perform an incremental update on the screen with the help of metadata.

Module: dataFlowModelSelect
Input:  Two adjacent frames f_k, f_k+1 where each frame is an m X n array of pixels.
Output:  0 if job is executed by pull model
        1 if job is executed by push model.
1.  For every time period τ
        For each i in m
                For each j in n
```

Diff = Diff + $f(f_k(i,j),f_{k+1}(i,j))$
(where f(A, B) = 1 if two pixels are same, 0 otherwise.)
If Diff > Threshold
dataFlowModel = PULL
else
dataFlowModel = PUSH

Module: Graph colour
Input: Frame Occurence Graph
Output: Frames allocated correspondingly in buffer

Use breadth first search to determine the frames which will be alive at the same time.

- For every node n in CFG, we have out[n]
  - Set of temporaries live out of n
- Two variables interfere if
  - both initially live (ie: function args), or
  - both appear in out[n] for any n, or
  - one is defined and the other is in out[n]

find a node with at most K-1 edges and cut it out of the graph, push it to stack

When the simplified subgraph has been colored, add back the node on the top of the stack and assign it a color not taken by one of the adjacent nodes.

once all nodes have K or more neighbors, pick a node for spilling using heuristics
- Storage on the stack

rewrite code introducing a new temporary; rerun liveness analysis and frame allocation

4. PERFORMANCE EVALUATION

4.1. Mathematical Analysis

Response time in Pull model=Propagation time + Processing time.

$$T_r = ((\alpha \cdot \text{Old\_RTT}) + ((1 - \alpha) \cdot \text{New\_RTT\_sample}))*(s1+s2)/B + \delta \quad (1)$$

where B is the Bandwidth in Bytes per second.

s1 is the size of the request packet.

s2 is the size of the response content.

δ is the processing time.

Response time in Hybrid Push Pull model

$$T_r=(P*\tau_b) + (1-P)((\alpha \cdot \text{Old\_RTT}) + ((1 - \alpha) \cdot \text{New\_RTT\_sample}))*(s1+s2)/B + \delta) \quad (2)$$

where P is the probability of buffer hit.

τ_b is the time taken to retrieve from the local buffer.

The value of P should be maximum for minimum latency. Consider a priority based pre-fetching scheme consisting of 'n' priorities $\{p_1,p_2 \ldots p_n\}$. Assume there are m items in each priority category.

Time taken to transfer packets of priority p_1,

$$T_{p1} = (m*b)/B \quad (3)$$

Where m is the no of data items of priority p_1.

b is the size of the data item.

Time taken to transfer packets of priority p_n,

$$T_{pn} = (n*m*b)/B \quad (4)$$

When an arbitrary request of priority p_i arrives and the idle time of the client is T_i, the probability that packet will be available in buffer is

$$P = \begin{cases} 1 & \text{if } T_i > (i*m*b)/B \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

4.2. Experimental Design

The system is tested with one coordinator and 3 clusters each consisting of 5 nodes running VMware hypervisors. The mobile devices connect to the cloud via the coordinator. The setup time, interaction latency under various conditions is measured. The latency varies according to the nature of the application. Two types of tasks are chosen for this purpose, one is an interactive task such as a game and another task which requires relatively lesser interaction such as switching between screens, pointing.

While creating multiple VMs for a single session, it is ensured that they're mostly allocated in the single node so that synchronization of VMs takes minimal time.

2.5. Results

Figure 3. The graphs show the variation in interaction latency with respect to the interaction rate for both pre-fetching and non pre-fetching schemes. The non pre-fetching scheme has a fairly constant latency whereas the pre-fetching scheme performs extremely well under low interaction patterns since the next frames can be retrieved locally without much processing and the latency increases rapidly with the interaction rate. This is because of the time required to pre-fetch the pages exceed the inter-key-press time leading to poor performance under high interaction rates. Thus the amount of pre-fetching that should be done must be varied according to the interaction pattern.

Figure 4 gives the plot between the bandwidth of the wireless link and the frame rate of the thin client displays for two buffer sizes 1 Mb and 2 MB. Since the successive frames are stored in the buffer, the frame rate is limited by the size of the buffer. So, the frame rate cannot exceed the threshold value which is determined by the buffer size.

If a frame to be displayed is available locally in the buffer, a buffer hit occurs and if a miss occurs, the client needs to pull the update frame from the server. The presence of a frame in the local buffer is dependent on the parameters like bandwidth and buffer size. If the bandwidth is more, the server can push more number of frames by pre-computation within a given time frame. Buffer size is also an important requirement to store the frames as and when they arrive from the server side.

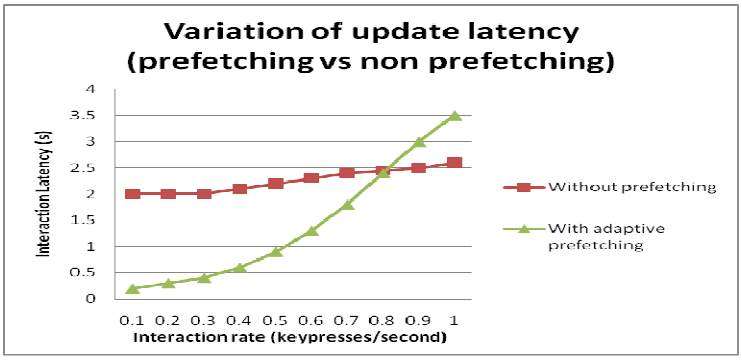

Figure 3. The graphs show the variation in interaction latency with respect to the interaction rate for both pre-fetching and non-prefetching schemes. The non-pre-fetching scheme has a fairly constant latency whereas in pre-fetching scheme, the latency increases rapidly with the interaction rate.

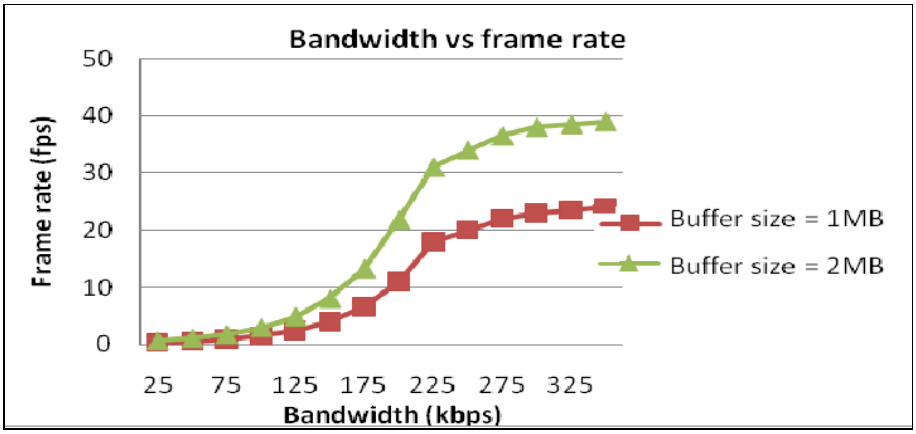

Figure 4. Variation of frame rate in the client's display vs bandwidth for given buffer size.

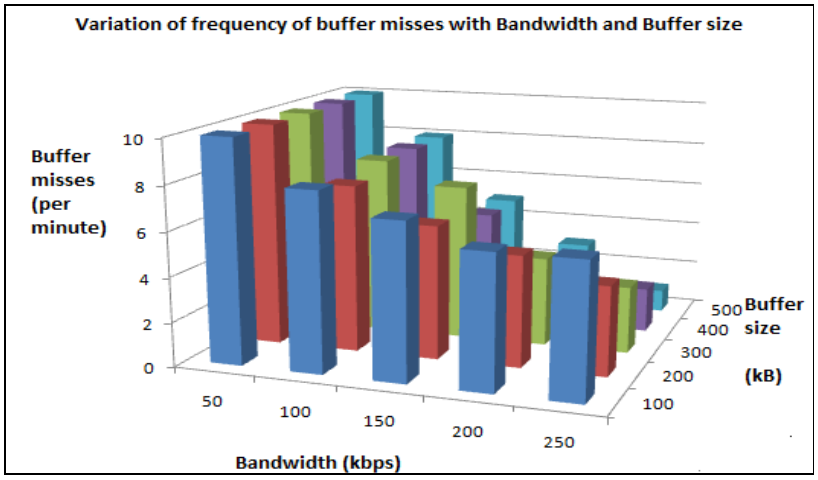

Figure 5. 3d plot showing relationship between frequency of buffer misses, Bandwidth and size of the buffer.

A combination of good bandwidth as well as buffer size is necessary for good performance.

3. CONCLUSIONS

The integration of mobile and cloud has given an entirely new perspective on what can be done using the simple handheld mobile phones. Users can share data, safeguard their data and access it from anywhere with credentials, and run any applications which can run in a powerful desktop computer over their mobile phones. Cloud offers SMS's and all can be sent merely through TCP by copying the message into the receiver's address space. The idea presented in this paper takes into the fact that the client has a high speed internet connectivity to push advance updates during idle time. Future work is to render the display updates to users with low-bandwidth connectivity. Also the pre-fetching module used consumes relatively high battery power and methods to reduce the battery consumption should be looked upon. While we create multiple virtual machines for client, instead of maintaining separate mirrored ram's for each VM, we can have a single copy and store the dirty pages particular VMs separately. This could make synchronization easier and less resource consuming.

REFERENCES

[1] Baratto Ricardo A , Potter Shaya, Su Gong, Nieh Jason (2004) "MobiDesk: Mobile Virtual Desktop Computing", MobiCom '04 Proceedings of the 10th Annual International Conference on Mobile Computing and Networking, pp. 1-15.

[2] Boukerche A, Pazzi R W N, Feng J (2008), "An End-To-End Virtual Environment Streaming Technique for Thin Mobile Devices Over Heterogeneous Networks", Computer Communications, vol. 31, no. 11, pp. 2716–2725.

[3] Chee Yik Keong, Poo Kuan Hoong, Choo-Yee Ting (2011), "Efficient Hybrid Push-Pull Based P2P Media Streaming System", IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS), pp. 725-730.

[4] Giacomazzi P, Poli A (2010), "Push-Pull Techniques in Peer-To-Peer Video Streaming Systems with Tree/Forest Topology", International Congress on Ultra-Modern Telecommunications and Control Systems and Workshops (ICUMT), pp. 89-95.

[5] Huang Dijiang, Zhou Zhibin, Xu Le, Xing Tianyi, Zhong Yunji (2011) "Secure Data Processing Framework for Mobile Cloud Computing ", IEEE Conference on Computer Communications Workshops, pp. 614-618.

[6] Kawashima H, Koshiba K, Tuchimochi K, Futamura K, Enomoto M, Watanabe M (2007), "Virtual PC-Type Thin Client System," NEC Technical Journal, vol. 2, no. 3, pp. 42–47.

[7] Kovachev Dejan, Renzel Dominik, Klamma Ralf, Cao Yiwei(2010) "Mobile Community Cloud Computing: Emerges and Evolves ", IEEE International Conference on Mobile Data Management, pp. 393-395.

[8] Lamberti F, Sanna A (2007), "A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices", IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 2, pp. 247–260.

[9] Pendyala V S, Shim S S Y (2009), "The Web as the Ubiquitous Computer", IEEE Computer, vol. 42, no. 9, pp. 90–92.

[10] Satyanarayanan M, Bahl P, Caceres R, Davies M (2009), "The Case for VM-Based Cloudlets in Mobile Computing", IEEE Pervasive Computing, vol. 8, no. 4, pp. 14–23.

[11] Simoens P, Ali F A, Vankeirsbilck B, Deboosere L, De TurckF , Dhoedt B, Demeester P, Torrea-Duran B (2010) "Cross-Layer Optimization of Radio Sleep Intervals to Increase Thin Client Energy Efficiency", IEEE Communications Letters, vol. 14, no. 12, pp. 1095-1097.

[12] Simoens Pieter, De Turck Filip, Dhoedt Bart, Demeester Piet (2011), "Remote Display Solutions for Mobile Cloud Computing", IEEE Computers, vol. 44, no .8, pp. 46-53.

[13] Tan K J, Gong J W, Wu B T, Chang D C, Li H Y, Hsiao Y M, Chen Y C, Lo S W, Chu Y S, Guo J I (2010) "A Remote Thin Client System for Real Time Multimedia Streaming Over VNC", IEEE International Conference on Multimedia and Expo (ICME), pp. 992–997.

[14] Sridhar S, Satish G, Raja G, Sumalatha Ramachandran (2012), "Adaptive Display Virtualization and Dataflow model Selection for Reducing Interaction Latency in Thin Clients", International Conference on Recent Trends in Information Technology (ICRTIT 2012), pp. 233-238.

Authors

Sumalatha.M.R is currently the Associate Professor in Department of Information Technology in Madras Institute of Technology, Anna University Chennai. She has 7+ Years of Teaching Experience as Guest Faculty, Teaching Research Associate and Lecturer in the Department of Information Technology, Madras Institute of Technology, Anna University.

Sridhar S has completed his B.Tech Information Technology in Madras Institute of Technology, Anna University Chennai. He has published two papers in International Conferences.

Satish G has completed his B.Tech Information Technology in Madras Institute of Technology. He has published a paper in International Conference. He is also an IEEE member.