Network Latency Impact on Performance of Software Deployed Across Multiple Clouds

Adnan Faisal, Dorina Petriu, Murray Woodside

Dept. of Systems and Computer Engineering
Carleton University, Ottawa K1S 5B6, Canada
{faisal | petriu | cmw}@sce.carleton.ca

Abstract

In cloud computing, an "edge cloud" may be introduced close to some of the end users, to give faster service for very demanding applications. The transactions that require heavy processing capacity and longer processing times are seen as more suitable to be carried out at the "core" cloud. Parts in the core and edge may then have to communicate, introducing associated network latencies. An application should be deployed across edge and core with the aim to reduce the overall effect of network latencies, in order to meet end user response time goals. In this paper, we use a Layered Queueing Network performance model to explore the impact of network latency and some possible deployment choices on the responsiveness of an application called HCAT (Home Care Aides Technology). The evaluations show that the use of the edge cloud may cause performance degradation, rather than gain, for some kinds of applications.

1 Introduction

Cloud computing [1] refers to the applications and services that run on a distributed network using virtualized resources and accessed by common Internet protocols and networking standards. The distinguishing features of cloud computing include the notion of effectively limitless virtual resources and the transparent usage of physical systems by the users.

Cloud computing has gained substantial popularity in the computer industry due to its significant cost advantages. Cloud computing has made it possible to use computing resources (both hardware and software) as a utility. Cloud service providers offer web service interfaces to obtain and configure computing capacity instantly with minimal friction. But there are some characteristics of cloud computing which are holding it back. These include issues with transaction control, security and regulatory compliance and network latency.

In order to reduce the latency of accessing highly demanding, dynamic applications, the notion of an extended cloud has been introduced [2]. The extended cloud refers to the use of caching data on one or more *edge clouds* to improve the scalability and efficiency of applications deployed on the *core cloud*. The underlying philosophy of the edge cloud is to move data and possibly some parts of the application closer to the end-user of the system. Figure 1 shows the extended cloud architecture with one core cloud and one or more edge clouds. The clients use handheld devices to access the edge cloud which is located possibly in the same area or city. The edge has the advantage of low latency, but it has limitations on computing and storage capacity. Through a powerful backbone or dedicated high-speed network, the edge can exchange information with the core cloud. Due to its distant location, any request made by the user takes more time to access the core cloud, compared to the time it takes to access the edge.

Copyright © 2013 Murray Woodside, Dorina Petriu and Adnan Faisal. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

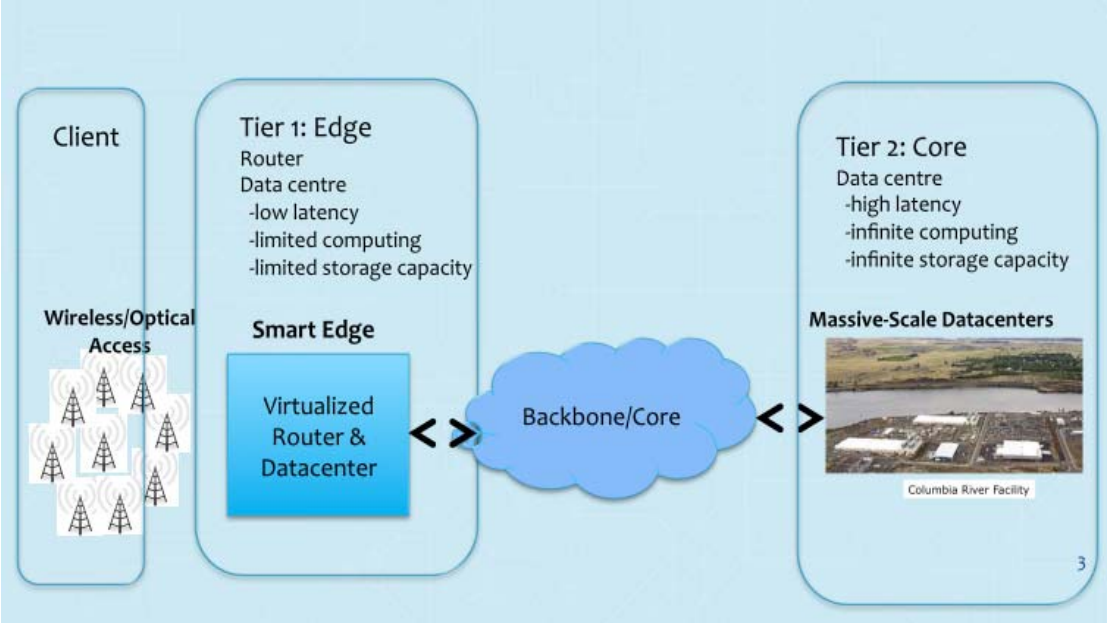

Figure 1: Extended cloud architecture (taken from [2])

Researchers who work on edge clouds mostly concentrate on developing different algorithms for edge caching that maintains data consistency between different clouds [3][4][5], partitioning data between edge and cloud [6], minimizing data accessing time for content delivery networks [7], etc. However, we have not found any comparative study on the end-user response times for various combinations of edge-cloud application deployments.

Usually the researchers simply assume that the use of an edge cloud would improve the overall user-experience by reducing end-user response times. Although it is true that the edge cloud is located nearer to the users compared to the core cloud, still its use may reduce the system performance in some cases. This is due to the fact that the edge cannot to do all the processing and storage locally and it must also access the core. This edge to core communication can add extra latency to the overall user experience, which may be significant depending on the traffic patterns and the deployment of the application.

In this paper we address this issue of network latency and demonstrate cases where the use of edge cloud may either increase or decrease system response time. To illustrate this we use a distributed application called HCAT (Home Care Aides Technology) [10][11], which includes both streaming video and transaction processing capabilities. Users of the system can make service requests through their handheld devices (e.g., mobile phone) or sometime through computers, and the requests may need to go through both edge and core clouds. We show the impact of various deployments, data partitioning and network latencies on the system performance. We find out that the impacts of deployment changes are not always as we expect them to be.

The performance modeling technique that we use in this paper is called Layered Queuing Networks [8][9], an extended queuing model which reflects software structure and interprocess communication, and the effects of resources in layered components.

The rest of the paper is organized as follows. In section 2 we describe the HCAT software, in section 3 we give a brief description of the Layered Queuing Network (LQN) model, in section 4 we discuss the main issues addressed in this paper. An LQN performance model for the HCAT software is described in section 5 and the results of experiments are discussed in section 6. The paper concludes by briefly summarizing its contributions, its limitations and possible future extensions.

2 HCAT Model Description

HCAT (Home Care Aides Technology) [10][11] is a distributed application to help health care professionals working in physically dispersed locations and their patients or clients. HCAT supports scheduling of services and video contacts, and storage and retrieval of client-related data. In an online video-conference, patients and healthcare professionals from different cities or countries can communicate with each other, each of them accessing the system through their nearest edge cloud.

2.1 Functionality Description

The description of HCAT used here represents a hypothetical distributed architecture, based on preliminary descriptions of usage scenarios. The naming of services and modules were created for the performance study, and has not been synchronized with the actual software, whose development is being completed. The HCAT system model shown here has four main services:

1. **Home Care Information Management Service,** which manages three kinds of information :
   a. Clients and their care plans
   b. Home Care Aides (HCAs), their specialties and their availability. HCAs include every kind of health care professional.
   c. Current schedule of HCA visits to clients.

A HCA using a mobile device can view and manage his/her and clients' information. For example, he can view his appointments, update his availability, and after reviewing required documents he can update clients' care plan.

2. **Scheduling Service:** This service suggests when and where the HCA should go to see patients. This might be computationally intensive depending on the complexity of the availability of the HCAs and clients.

3. **Synchronous Collaboration Service:** Clients and HCAs may communicate with each other via video conference. The video conference is initiated by the head-nurse at client's location, preferably at his home. Different specialist doctors across different cities and provinces may participate in a video conference. The HCAs will be able to take notes and annotate the video in real-time. Video-conferences are archived for record-keeping and later-viewing. The video, annotations and summary of the video-conference becomes part of the patient record.

4. **Central Analysis Service:** The HCAT administrators use this service to do manage and monitor clients and HCAs. This include analyzing and managing clients home care plans, analyzing performances of HCAs and support other administrative functionalities in general.

A high-level functional architecture of the above HCAT system is shown in Figure 2.

2.2 Deployment Description

The services described in the previous subsection might be deployed either in edge or in core or in both edge and core. Depending on the workload of the services, they can be migrated between edge and core. Nevertheless, these services have default choices of deployment due to their nature of work.

Base-Case Edge deployment: An instance of the HCAT application that contains the home care information management service, scheduling service and synchronous collaboration service is assumed to be deployed at the edge. For any edge cloud, this part of the HCAT application will contain only the data related to the clients located near to it.

It should be noted that the number of edge clouds can grow dynamically as hospitals and clients at new locations can start to join the HCAT framework. Each edge registers with the core and some of its data is archived at the core in a common format.

Base-Case Core deployment: It is assumed that the central analysis service is deployed at the core. It is accessed by the system administrators who use computationally heavy analytical software for performance evaluation of HCAs and plan management of the clients.

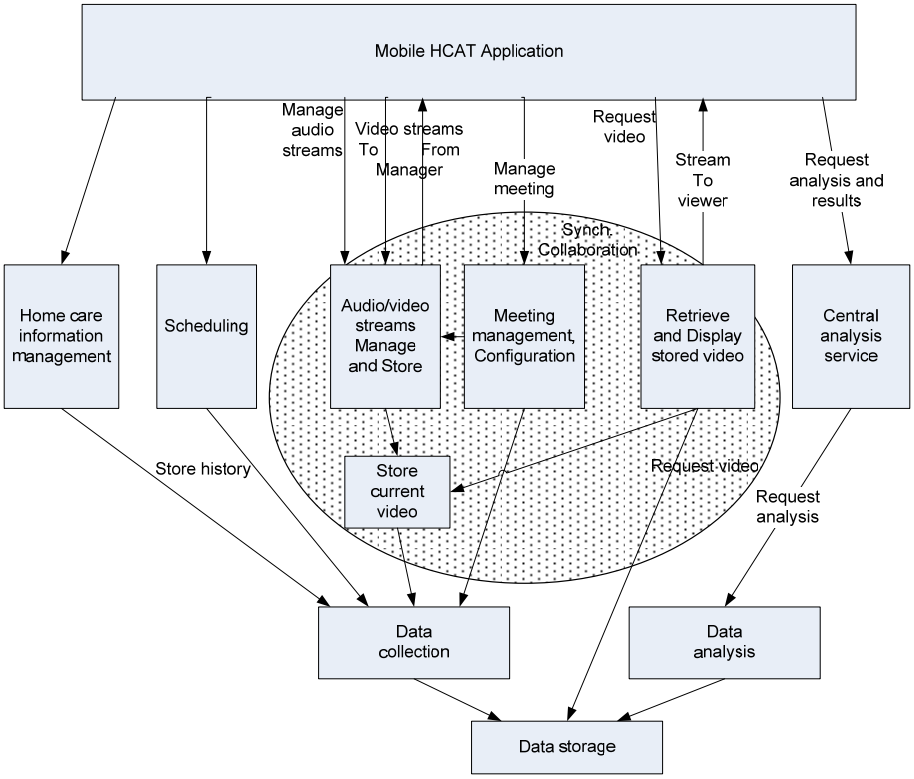

Figure 2: High-level functionality architecture of HCAT

The other three services (i.e., home care information management service, scheduling service and synchronous collaboration service) are also deployed between the edge and the core. Some of the edge queries of these services are forwarded to the core for processing and storing. Heavy computations and data storing of these services take place at the core.

The deployment description above implies that there are data which are contained only by the edge and not by the core at a given time. However, these data would be periodically pushed to the core for archiving and they would be accessible to the central analysis service.

In Figure 3 the base-case deployment of the HCAT application is shown. All four services are deployed at the core, but the edge has only three services as it excludes the central analysis service.

3 Layered Queuing Network (LQN)

The performance models in this paper use the Layered Queuing Network (LQN) technique [8][9]. The LQN models are capable to represent the software components and their deployment, to capture inter-component communications, and to analyze resource interactions between layers of the application.

Figure 4 shows a generic layered web service application as a simple example. The main players of LQN are *tasks* (representing software processes), which are shown as rectangles, with two or more parts. The rightmost part defines the task itself (its name and its thread-pool multiplicity, if greater than 1); the other rectangle parts represent the task operations, which are called *entries*, and are labeled with the host (CPU) demand for one invocation of the entry. Each task has a *host* processor (drawn as an oval). A *call* from one entry to another is represented by an arrow labeled with the number of calls. Every task and host is a queuing server, so requests to an entry first go into the queue for the task owning the entry, and are served when a task thread becomes available. An infinite server provides a pure time delay (without any waiting for the server). It is modeled as an infinite task running on an infinite host. Infinite servers are used in this work to represent network latencies.

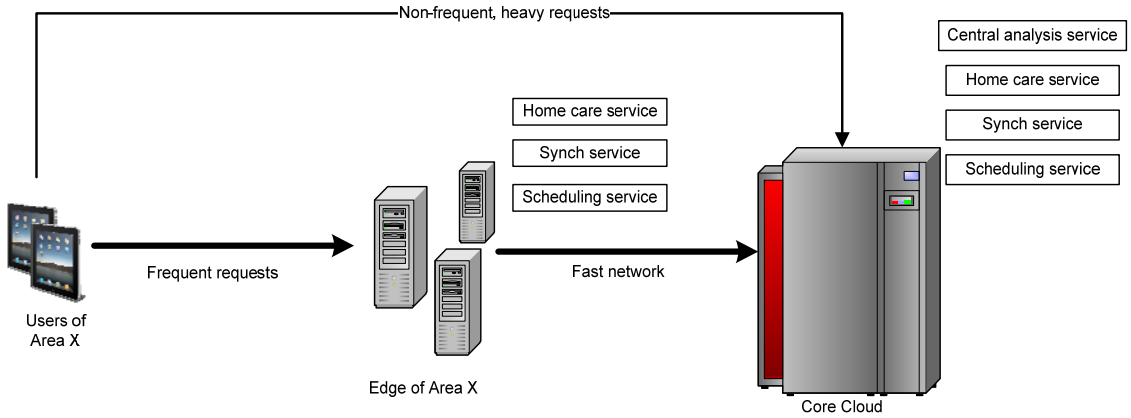

Figure 3: HCAT deployment architecture

Figure 4 shows four hosts: ClientP, InternetP, WebServerP and DbServerP, and four tasks: User, Latency, WebServer and DbServer. 100 Users each having 5 minutes think time between requests are modeled using 100 source tasks running on 100 replicated hosts (UserP). Network latency is modeled by the infinite threaded Latency task running on an infinitely replicated host InternetP. This task has an entry for each type of request carried over the network, each having a pure delay (Z) equal to the network latency of 100 milliseconds.

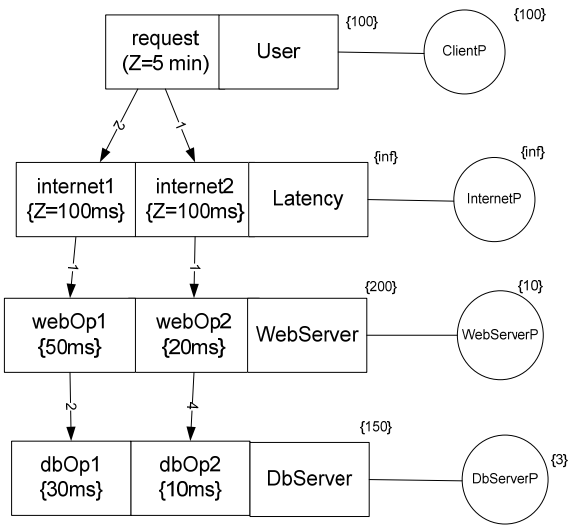

Figure 4: An example LQN model of a generic two-tier web application

The application has two layers defined by the WebServer and DbServer tasks, providing two services simply called Op1 and Op2. Each service has an entry at each layer, with is appropriate host demand. The maximum thread pool size is 200 for the WebServer task and 150 for the DbServer task. The entry webOp1 calls dbOp1 twice, and webOp2 calls dbOp2 four times.

4 Issues to be Addressed

This paper addresses three issues for HCAT:

1. **The impact of network latency on performance:** When an application is deployed across multiple clouds and user queries need to travel through those clouds, the communication that takes place between the clouds can cause extra latency. In this paper we see the impact of this extra communication by varying the network latency between edge and core clouds. We also see the effect of latency on the other links, i.e., between user and edge, and between user and core.

2. **The impact of different deployments:** We start examining the system with a default deployment in which data and computations are split between edge and cloud. We also examine the system performance at extreme cases in which the whole application is deployed solely either at edge or at core cloud.

3. **The impact of different data partitioning:** In many distributed applications the lack of data locality is one of the major reasons for bad performance. In this paper we vary the number of core data accesses done from the edge and examine their behavior. We also examine whether moving data from edge to core gives any substantial performance benefit.

5 Performance Model

To develop a compact performance model, we consider the functional architecture of HCAT, and based on its usage profile we aggregate different kinds of requests to simplify the model. The resulting modules are represented as LQN tasks and are deployed in the mobile terminal, edge cloud and core cloud.

5.1 From Functional Model to Performance Model

Given the four services described in Section 2.1, the usage profile [10][11] of HCAT identifies the important types of user requests to be analyzed for performance analysis. From the usage profile it seems probable that the central analysis service may have a substantial workload and forms one operation class (i.e., *Central analysis service transactions*); the video streaming for conferencing is distinctive, potentially heavy and time critical so it forms a second operation class (i.e., *Conference video streaming*) . There is another video streaming operation to view a video that has been stored, which will be treated as a separate third class (i.e., *Video retrieval streaming*) because it has a different traffic pattern. The remaining services have been lumped together as a mixture of request-reply transactions (i.e., *Transaction handling*), apart from the analysis operations.

This gives four classes of requests in the performance model. The flows of requests and data associated with these traffics are illustrated in Figure 5. The thicknesses of the lines give estimated volume of data flow among user and clouds.

5.2 LQN Model Elements

Different combinations of LQN tasks are used to represent the four classes of requests (See Figure 6).

- Transaction handling class is modeled using a front-end task *HandleXact*, supported by a back-end *DataAnalysis* task.
- Central analysis service transactions are also handled by the *DataAnalaysis* task but without going through the *HandleXact* task.
- Conference video steaming is modeled by the *AVInOutStream* task, which receives a video stream from one mobile and redirects it to a set of *$nVC-1* other mobiles *($nVC* = average number of participants in a video conference).
- Video retrieval streaming (for review of archived videos) is modeled by the task *AVOutStream* task which retrieves a video file from storage and sends it to one mobile that receives it via the *GetVideo* task. Video archiving and retrieval was added to this model of HCAT as a possibly interesting consequence of having videos available for review.

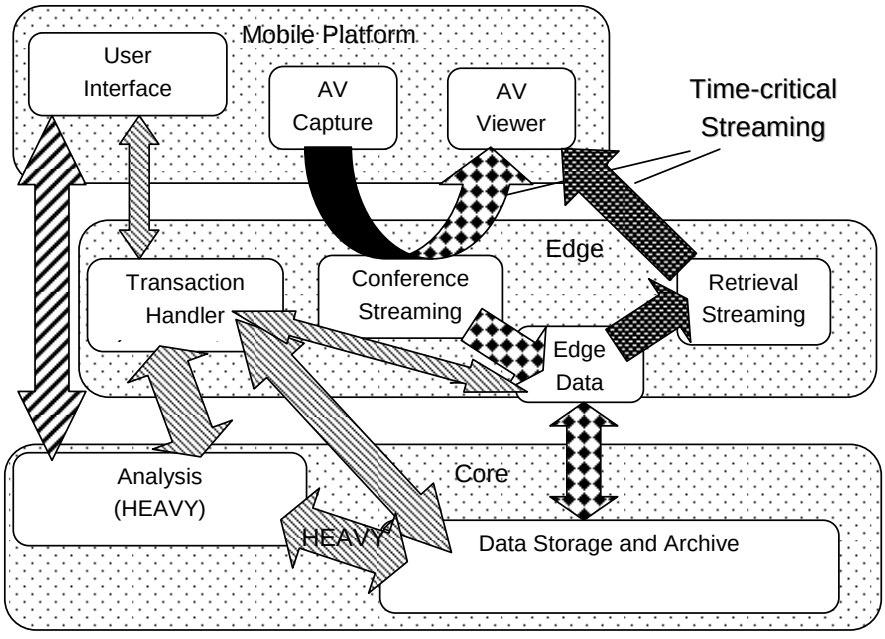

Figure 5: Information flow for the four classes of operations, indicated by shadings

Each mobile device has an interface task *MobileApp* which accepts transaction requests from the user and triggers system operations, a task *Camera* to capture video frames in the mobile and send them to AVInOutStream, and a task *VideoViewer* to receive and display a video stream. Traffic generated due to the video conference service goes from the source mobile to all other mobiles via a conference server, and the video files are also archived.

All tasks in the model can be replicated. It is assumed that each video stream is handled by a dedicated concurrent task, even within the mobile (thus, a conference with four participants would have four active video streaming tasks, one to send the image from that mobile, and three to receive streams from other mobiles).

The transaction stream in the usage profile includes transactions to set up video conferences, retrieve videos, and to do small archive retrievals, as well as scheduling and care-support transactions. Additional tasks are proposed to support edge/core operations:

- A video cache task *VCache* in the edge to capture conference videos for archiving, and to store retrieved videos while they are being viewed.

- *DataAnalysis* task is split among edge and core and two different tasks *DataAnalysisE* and *DataAnalysisC* are made. Both of these *DataAnalysis* tasks have their local storages for storing and retrieving data. These local storages are called *DataStorageE* and *DataStorageC* for edge and cloud respectively.

- To represent network latencies we have think times, between a pair of communicating entries (which are not shown for clarity in Figure 6). For mobile to edge latency, this think time is represented by the variable *$me*, for core to edge latency by *$ce*, and for mobile to core by *$mc*.

- There is one more task called *Clock* in the LQN model. This is a timing task to control the rate of operations in the video streams, otherwise the default model semantics would pull the frames out as fast as possible.

5.3 Model Description

The task *mobileApp* represents the HCAT application running on the mobile device. Through the *interact* entry, the mobile makes requests which are grouped into four classes. Two of these classes are transaction classes and the other two are video classes. The transaction classes include requests for central analysis transactions and requests for other transaction operations. On the other hand, the two video classes include requests for capturing / sending / receiving video in a conference, and requests for receiving an archived video.

The parameters of the model are entirely hypothetical, but have been chosen to represent potentially possible values. This is consistent with the purpose of the analysis, to identify possible problems and the situations in which they might arise.

The assumed usage of the four services was modeled over an average interval of 10 minutes, making service requests to:

- Home care information management and scheduling services, lumped together as *Transaction handling*, 1 transaction on average;
- Central analysis service, .01 request (average), or about one per day over all users;
- Synchronous-collaboration service (video conference service), 0.16 requests (average), or about one per hour;
- Video retrieval, 0.08 requests (average), or about one per two hours.

Central transaction analysis requests go to the *doAnalysisC* entry in the Core, where its host demand is given by the variable *$doBA* (do big analysis) set to 30,000 ms (i.e., 5 minutes). This is intended to represent something like a data-mining operation, and it causes a large number of accesses to the core database's *getXactDataCC* entry. This large number of accesses is represented by the variable *$nBDA* (number of big data accesses), set to 10,000. The host demand of *getXactDataCC* is represented using the variable *$hXD* (handle transaction data) which is set to 200 ms. Transaction handling requests go from mobile to edge and are handled by the *HandleXact* task's *acceptXact* entry with host demand $hX (handle transaction) which is set to a small value (10 ms), as this represents a small transaction.

If the request received by *acceptXact* is only a store request to the core then the access to *StoreXact* entry is needed with probability 0.2. The service demand of this entry is *$hXD* (storing transaction data = 200 ms). On the other hand, *acceptXact* can receive requests for data analysis, and a fraction 0.6 of these requests are completely processed locally by *doAllAnalysis* (service demand *$doSA* = 100 ms) accessing local database *getXactDataE* (service demand *$hXD* = 200 ms) twice for each request.

A fraction 0.2 of the requests coming from *acceptXact* go to *doAnalysisE* are also completely processed at the edge but they require core data access.

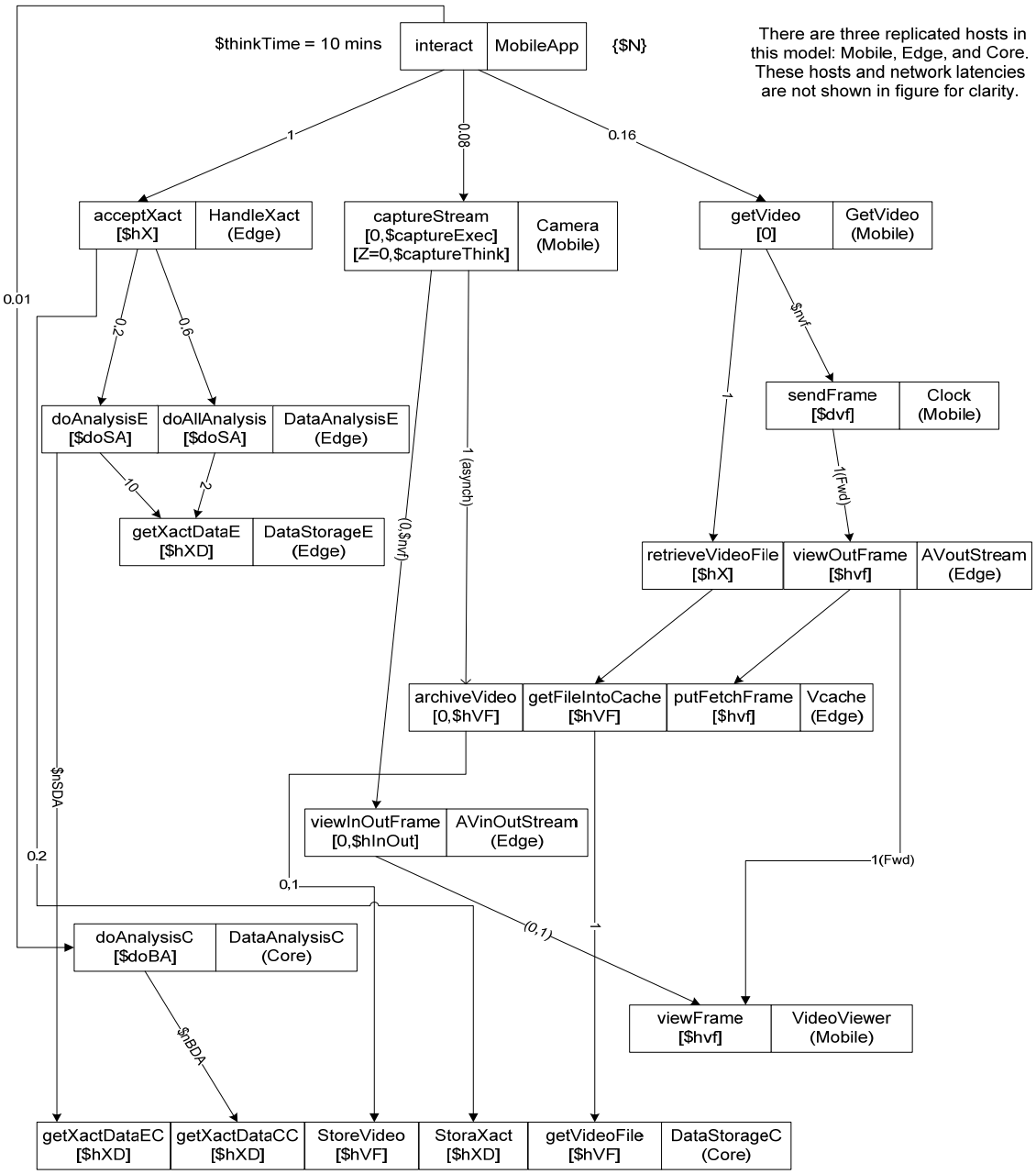

Figure 6: LQN model of the HCAT System. Parameters such as $hXD are defined in the text.

Requests for home care management service are examples of such services.Each request coming to *doAnalysisE* requires 10 accesses to *getXactDataE* and *$nSDA* (no. of small data analysis) accesses to core's *getXactDataEC* entry.

The default value of *$nSDA* is 200. This value is varied in the experiments to determine the impact of fetching different volumes of data from core to edge.

The video handling model is based on some assumptions. A video conference is assumed to be 5 minutes long on average, with a frame rate of 24

frames/s, resulting in a total of *$nvf* = 7200 frames in 5 minutes. This gives a frame interval of about 42 ms.

Initially a video is captured by the camera of the mobile, represented by the entry *captureStream* with a total service demand (over the whole 5 minutes) of *$captureExec* and an accumulated interframe delay of Z=*$captureThink*. The values of *$captureExec* is set to 10 ms/frame, or 72000 ms. From *caputureStream*, *$nvf* frames are forwarded to *viewInOutFrame* (service demand *$hInOut = ($nVC - 1)*$hvf*, where *$nVC* is the number of conference participants). These requests are further forwarded to the viewer cameras *viewFrame*. In the experiments, we assume that though many conferences can take place concurrently, but in a single video conference only 3 mobiles participate on average, therefore *$nVC* = 3. Also, from *captureStream* one request is forwarded to store the video at the core's persistent storage via edge's *archiveVideo* entry (service demand handling video file *$hvF* = 12,000 ms).

A mobile can either retrieve stored video or view an ongoing conference. A stored video is retrieved by downloading it as a large file (possibly in .wmv or .avi format) In order to download the video file, a request is sent to the edge's *retrieveVideoFile* entry (service demand *hX* that we used previously). This entry makes request to retrieve the file from cache via the *getFileIntoCache* entry (service demand *$hVF* introduced previously). The file is brought into cache from core using the *getVideoFile* entry.

In case of a video viewed by the mobile, it has to be viewed frame by frame. A 5 minute video frame has *$nvf* (7200) amount of frames, each having a duration of *$dvf* (duration of video frame = 42 ms). These frames are forwarded from *Clock* to the edge's stream controller (i.e., *AVoutStream* task) for processing; handling each of these video frames require *$hvf* (10 ms) amount of processing time. This framing mechanism is represented in the model using the entries *getVideo, sendFrame* and *viewOutFrame*. If the video is to be viewed by mobile, then it has to be fetched by the stream controller from cache and needs to be forwarded to mobile's *VideoViewer* task. In the LQN model, the view operation is carried out by mobile's *viewFrame* entry.

The base-case number of users was set at 150, and then varied in the experiments to test the system under different workloads. A compressed 5 minute long video is assumed to be 600 Mb in size and is processed both at the edge and at the core at a rate of 10Mbps.

We use think time parameters to represent network latency between every pair of communicating tasks that are not located in the same cloud. There are three variables: for mobile to edge it is *$me* = 50 ms, mobile to core *$mc* = 200 ms, edge to core *$ec* = 50 ms. These values were varied during the experiments to test the system under different latencies.

5.4 Default Deployment Description

The following deployment is taken as the default deployment:

- *HandleXact* on the edge
- *AVInOutStream* and *AVOutStream* on the edge
- *VCache* on the edge
- *DataAnalysis* partly on the edge and partly on the core
- *DataStorage* partly on the edge and mostly on the core.

The deployed LQN model of HCAT is shown in Figure 6. During the experiments, the deployment is varied to see the effect of different task migrations.

6 Experimental Results

A number of experiments were performed with the model. We have varied the number of mobiles in the system in order to find out the system saturation point. We have also observed the impact on system performance of different deployments, of network latency change, of change of number of requests forwarded from edge to core and of moving data between edge to core.

6.1 Point of Saturation

For any system, one of the most important performance information is to know the maximum number of users a system can support before getting saturated. Therefore, the model was solved with different numbers of mobiles to find its saturation point.

Figure 7 and Table 1 give results that show the effect of increasing the number users (i.e., mobiles) in the system. There are 10 edge and 10 core processors in the system, and the core is under-utilized.

Table 1: Performance results for different no. of mobiles

#Mobile	U_{eP}	U_{cP}	R_{acptX} [ms]	R_{getVid} [ms]	R_{sys} [ms]
50	2.95	0.02	10190	24437	34406
100	5.91	0.04	10194	25380	34561
150	8.85	0.07	10238	31592	35599
200	10	0.08	12575	670842	140219

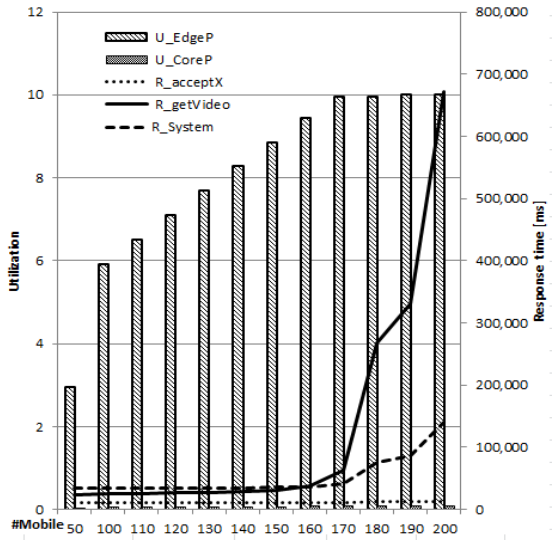

Figure 7: Utilization and Response time for different no. of mobiles (See Table 1 for data)

Under the base-case deployment, a larger amount of processing is done in the edge cloud, and only a few heavy processing and data storage requests are forwarded to the core cloud. As a result, all the edge processors become saturated at 170 users, but the core processors are utilized only about 8%. This is with only one edge cloud; as the number of edge clouds increase, more requests would come to the core cloud and the core utilization would increase.

Although we have four classes of traffic, only the response times for the most important two are shown in the graphs. For instance, the number of requests directly going from mobile to core are very few (probability 0.01), so their response time is not shown, as they have little impact on the system performance. The response time of the *acceptXaction* class (*R_acceptX*) shown because it sends a large number of request to both the edge and the core, and their impact on the system response time is considerable.

The other two classes of traffic deal with video conferencing, and since both of them have similar load and pattern only one has its response time shown, for getting an archived video file (*R_getVideo*). The other response time shown in the figure is the overall average system response time (*R_System*), which is affected by both *R_acceptX* and *R_getVideo*.

From Figure 7 we can see that, until the number of mobiles reaches 160, the response times *R_acceptX*, *R_getVideo* and *R_System* do not change much. Initially the video data do not cause much congestion, and therefore *R_System* is only slightly larger than *R_Video*. However, the system is nearly saturated at 160 mobiles, so at this point the video files contribute significantly to the congestion, so the response time *R_getVideo* becomes greater than *R_System*. The system is completely saturated at 170 mobiles, so both *R_getVideo* and *R_System* continues to increase sharply from this point.

6.2 Different Deployments

Now we see the impact of one base-case and two extreme cases of deployment. In the base case HCAT is deployed both at edge and core as described in Section 5.4. We call this a *Split* deployment. The two extreme cases use only one cloud instead of two. *AllCore* refers to the case where HCAT is completely deployed at the core cloud, and *AllEdge* to the case where HCAT is completely deployed at the edge cloud.

Table 2: Response times (in ms) for video file retrieval for 3 different deployments

#Mobile	AllEdge R_getVideo	Split R_getVideo	AllCore R_getVideo
50	24,389.2	24,437.5	28,360
100	25,369.3	25,380.6	33,136.6
150	31,242.9	31,592.2	43,192.6
200	675,963	670,842	751,508

Figure 8 and Table 2 show that the *AllEdge* deployment usually gives the best performance, which is essentially due to the closeness of the edge to the users. Moreover, both the *AllEdge* and *Split* deployments are faster than *AllCore* deployment. The reason for this is the longer network latency between mobile and core (200 ms).

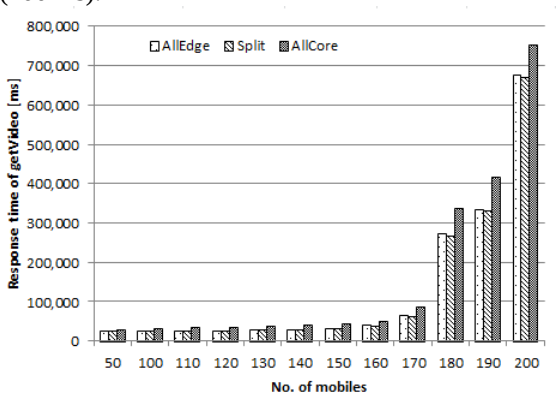

Figure 8: Comparing video retrieval times for 3 different deployments

We can also see that before saturation, the response time for AllEdge deployment is only slightly less than that of Split deployment. However, as the number of mobile phones crosses the saturation point, the response time of the *AllEdge* deployment becomes larger than that for the *Split* deployment. This is due to the edge saturation in the *Split* deployment for over 160 mobiles, so forwarding some of the traffic to the core

(which is under-utilized) gives a somewhat lower system response time.

The effect of deployment on the *acceptXaction* class is shown in Figure 9 and Table 3. *AllEdge* deployment outperforms the *AllCore* deployment. This is expected due to the longer latency (i.e., 200 ms) between mobile to core compared to the shorter latency (i.e., 50 ms) between mobile to edge.

Table 3: Transaction response times (in ms) for 3 different deployments

#Mobile	AllEdge R_acceptX	Split R_acceptX	AllCore R_acceptX
50	8180.01	10190	8330.01
100	8184.9	10194.6	8334.84
150	8233.97	10238.7	8382.43
200	10586.2	12575.6	10638.1

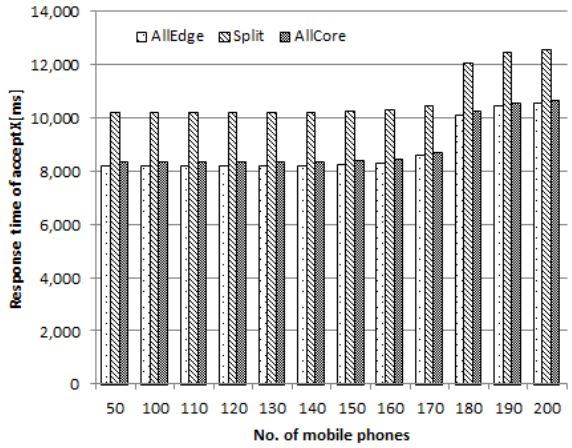

Figure 9: Comparing R_acceptX for 3 different deployments

Contrary to a reasonable expectation, the *Split* deployment gives the worst response time of the three deployments. Investigation in the result file showed that the cause is by the large volume of communications between the edge and core in the *Split* case. The conclusion is that for a system partitioned between multiple clouds, the aim should be to reduce the volume of communication between the edge and core. Otherwise, instead of performance gain, the edge could cause a performance loss.

Table 4: System response time (in ms) for different deployments

#Mobile	AllEdge R_System	Split R_System	AllCore R_System
50	32,387	34,406	33,186
100	32,565	34,561	33,971
150	33,877	35,599	35,786
200	147,071	140,219	164,326

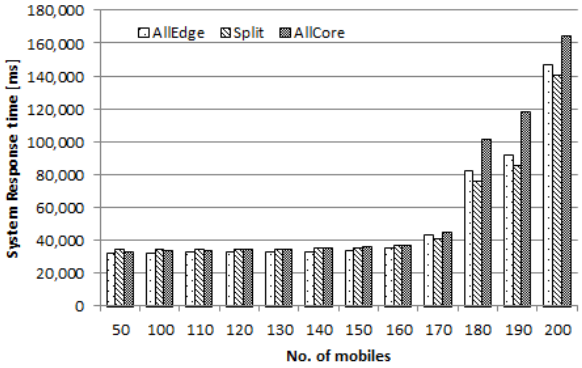

Figure 10: System response times for 3 different deployments

The system response times for the different deployments (see Figure 10 and Table 4) show a similar behavior as for the video retrieval class. The dominating workload in this model is the video workload, which has a stronger impact on the system response time than the other components.

6.3 Impact of Network Latency

The three network latencies were varied: mobile to edge (*$me*), mobile to core (*$mc*), and edge to core (*$ec*). Experiments varied each of these from zero to the maximum expected latency value (200 ms). The results show that only the $*ec* latency has an important impact on performance. Changing $*me* and $*mc* does not show much impact because the number of requests made in these two paths are far fewer than the number of requests from edge to core. Moreover, *$ec* is also the latency whose impact is of most interest to us. Because, we know that only a few requests go from mobile to core, and the edge cloud should stay close to the user, therefore $*me*>50 ms is not acceptable in practical scenarios.

For *Split* deployment and 150 mobiles the effect of changing *$ec* is shown in Figure 11 and Table 5. *R_acceptX* increases as we increase *$me* because transaction queries have a lot of communication between edge and core.

Table 5: Response times (in ms) for different edge-to-core latencies *$ec*

$ec[ms]	R_acceptX	R_getVideo	R_System
0	8,230.16	31,735.7	33,614
50	10,238.7	31,592.2	35,599
100	12,247.4	31,457.7	37,587
150	14,256.1	31,331.2	39,575
200	16,264.8	31,213	41,565

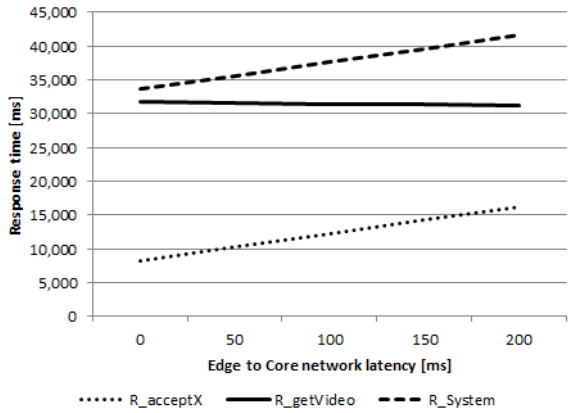

Figure 11: Response times for varying edge-to-core latency $ec

A consequence of higher congestion of *acceptXact* is that now this class of requests use fewer common resources, which becomes available for the other class of requests (i.e., video). The effect is that *R_getVideo* is slightly decreasing as we increase *$ec*, as seen in Figure 11 and Table 5.

6.4 Requesting More Data from Edge to Core

A transaction request which is being processed at the edge might not find the required data there and have to retrieve data from the core database. A large edge database reduces the probability of such an edge-miss, but edges are designed to contain only high-demand, local data, so misses are unavoidable. The system designer must decide the amount of data the edge should fetch from the core cloud.

Table 6: Impact of edge-to-core data access

$nSDA	R_acceptX [ms]	R_System [ms]
1	296.494	25,825
500	25229.9	50,399
1000	50220.1	75,174

From Table 6 and Figure 12, we can see that increasing the edge to core data access ($nSDA) increases the transaction and system response times. From this result, we can say that the system designer should aim for reducing the number of data accesses from edge to core. An intelligent partition of data and computation should reduce the edge to cloud data access and therefore improve system performance.

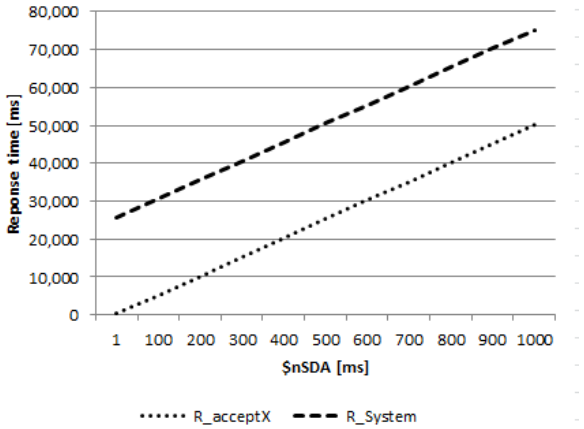

Figure 12: Response times for varying edge-to-core data access $nSDA

6.5 Moving Data between Edge and Core

To improve system performance the most frequently used data should be stored in the edge, whereas the less frequently accessed data should be stored only in the core.

We have already mentioned that in the HCAT system the latency that most impacts the system performance is the edge to core latency. So, if we can reduce the volume of edge to core communications, the system performance would be improved. Having a large edge database reduces the number of accesses the edge data analyzer needs to make to the core, and thus improves the system response time. This is seen in Table 7, where changing the core:edge data access ratio from (200:10) to (10:200) causes a huge reduction in response times of both *R_accept* (from 10,238 ms to 746 ms) and *R_System* (from 35,599 to 26,266).

Table 7: Large core DB vs Large edge DB

Xact data access ratio between Edge:Core	R_accept [ms]	R_System [ms]
10:200	10,238.7	35,599
200:10	746.098	26,266

This means that the system designer should strive for a data partitioning which would allow the edge cloud to be able to access its required data locally most of the time. The system designer would certainly want to have a large edge database, but we know that the edge database cannot grow unlimited due to the resource limitation and high cost.

Summary

Over all these experiments, we can see that:

- The amount of data processed at the edge is not trivial. If we want to gain performance by using edge cloud, the edge needs to be sufficiently powerful. This is particularly applicable for video data (See Figure 7 and Table 1)
- It is very important that the edge has a large enough database to process locally the queries submitted to it. If the edge requires to visit the core frequently, then the edge-cloud interaction might cause performance degradation.
- Among the three latencies: Mobile to Core, Mobile to Edge and Core to Edge, the most sensitive latency is Edge to Core. This is especially true for cases when the edge requires accessing the core for retrieving large amounts of data which needs to be transmitted fast enough to meet the response time requirements.
- It is commonly assumed that the use of a *Split* deployment is better (in terms of performance) than *AllCore* or *AllEdge* deployment. In the experiments we have found that this assumption is over-ambitious, as there are cases where split deployment is the worst of the three deployments.

7 Conclusion

Network latency plays a key role in performance for software deployed across edge and core clouds. It is generally assumed that the use of edge clouds would improve the system performance. In this paper, we have challenged that assumption and compared the effect of network latency for various deployments and various configurations. The experiments collected response times for two different classes of traffic – transaction and video – that are very different in terms of service demands and edge-cloud access. We have shown that some system changes may improve the performance of one class of traffic while deteriorating other class of traffic. We have also shown that data partitioning plays a key role in cloud performance. The more the edge computations access core data, the worse the system response time gets.

This work can be extended in several ways. First, a cost-benefit analysis can be carried out on growing the size of the edge cloud. This would justify why we cannot have infinitely large edge database. Second, a sensitivity analysis [13] can be carried out to estimate the uncertainty in the prediction of the model and to apportion it to different sources of uncertainty in its inputs. Third, the exercise above can be applied to the actual HCAT application once it is deployed to real edge and core clouds.

About the Authors

Adnan Faisal is a PhD student at Carleton University, Canada. He works in the RADS lab under co-supervision of Prof. Murray Woodside and Prof. Dorina Petriu. Previously, he completed his Laurea Specialistica (Master in Computing Systems Engineering) from Politecnico di Milano, Italy. His research interests and experiences are related to performance engineering and capacity planning of different types of computer systems and networks.

Dorina C. Petriu is a professor in the Department of Systems and Computer Engineering at Carleton University, Ottawa, ON, Canada. Her main research interests are in the areas of performance modeling and model-driven development, with emphasis on integrating performance engineering into the software development process. She was a contributor to two OMG standards, SPT and MARTE, which extend UML for real-time system modeling and analysis.

Murray Woodside does research in software performance engineering, especially on performance models for software systems. His work includes developing appropriate models, methods for obtaining models from designs and from run-time measurements, and the use of models for capacity planning and to improve software designs and run-time configurations. His contributions include the use of layered queueing models, efficient model solution techniques, performance annotations in software modeling languages (UML), system and design optimization, and statistical parameter estimation by Kalman filters. He is a Fellow of IEEE and a past chairman of ACM Sigmetrics.

References

[1] Barrie Sosinsky. *Cloud Computing Bible*, Wiley Publishing Inc., 2011.

[2] M. Litoiu, J. Chinneck, M. Woodside, K. Salem, Presentation on Extended Cloud Computing, *21st Centre for Advanced Studies Conference (CASCON)*, Toronto, ON, 2011.

[3] L. Ramaswamy, L. Liu, A. Iyengar. Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks. *In Proc. Of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 229-238, Columbus, OH, 2005.

[4] Ramaswamy, L.; Jianxia Chen., Efficient delivery of dynamic content: the cooperative EC grid project, *Collaborative Computing: Networking, Applications and Worksharing, 2005 International Conference on*, on page(s): 9 pp.

[5] Narravula, S.; Jin, H.W.; Vaidyanathan, K.; Panda, D.K., Designing Efficient Cooperative Caching Schemes for Multi-Tier Data-Centers over RDMA-enabled Networks, *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, on page(s): 401 - 408 Volume: 1, 16-19 May 2006.

[6] M. Smit, M. Shtern, B. Simmons, M. Litoiu, Partitioning Applications for Hybrid and Federated Clouds, *22nd Centre for Advanced Studies Conference (CASCON)*, Toronto, ON, 2012.

[7] M. Bjorkqvist, L. Y. Chen, M. Vukolic, and X. Zhang, Minimizing retrieval latency for content cloud, *INFOCOM*, Shanghai, China, 2011.

[8] Layered queuing network homepage. http://www.sce.carleton.ca/rads/lqns/

[9] G.Franks, T. Al-Omari, C. M. Woodside, O. Das, S. Derisavi, Enhanced Modeling and Solution of Layered Queueing Networks, *IEEE Trans. on Software Eng.* Vol. 35, No. 2, March/April, 2009

[10] S. King, L. Liu, E. Stroulia, I. Nikolaidis: Using Simulations to Integrate Technology into Health Care Aides' Workflow, *International Conference on E-Learning in the Workplace (ICELW)*, New York, USA, 2013

[11] E. Stroulia, I. Nikolaidis, L. Liu, S. King, L. Lessard. Home Care and Technology: A Case Study. *2nd International Conference on Global Telehealth*, pages 142-52, Sydney Australia, 6-28 November 2012.

[12] B. Sample. A Software Design Approach. In *Proc. Of the 15th Software Conference*, pages 40-50, Somecity, Somecountry, 1995.

[13] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D. Saisana, M., and Tarantola, S., 2008, *Global Sensitivity Analysis. The Primer*, John Wiley & Sons.