Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2013) 1–9

Procedia Computer Science

www.elsevier.com/locate/procedia

The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013)

Centralized Management of Scalable Cyber Foraging Systems

Manjinder Nir*, Ashraf Matrawy

Carleton University, Ottawa, Canada

Abstract

We present our work in the area of cyber foraging for mobile devices. The main focus of the work is to propose a framework for a scalable cyber foraging system (CFS), that could support a large number of mobile devices. In this paper, we focus on the resource monitoring process and its scalability in a large CFS. This work presents the challenges of implementing cyber foraging processes in large CFS i.e. the overhead of cyber foraging processes. We propose a Broker based framework to lower the communication overhead in the wireless network and enhance the resource monitoring time in mobile devices. Our simulation results show that a large CFS Broker based approach can lower the resource monitoring time and enhance the scalability of the process.

Keywords: Cyber Foraging, Mobile Devices, Pervasive Computing.

1. Introduction

Cyber foraging technology enables mobile devices to opportunistically use computing resources present in the vicinity [1]. Cyber foraging enables mobile devices to temporarily augment their resources through remote execution process. In this process, mobile devices can offload the resource intensive parts of their application to a resource-rich computing nodes. We call these nodes as 'service nodes'. The opportunistic use of resources means that mobile devices could use the computing resources of service nodes when these nodes are available and are willing to share their resources. Cyber foraging is beneficial if the cost of executing a task on a remote service nodes is less than the cost of running it locally. A number of cyber foraging systems exist [2], [3], [4], [5], [6], [7], [8], [9]. [10]. Most existing CFS base their experimental evaluation or performance analysis on a single or a few mobile devices and service nodes. These systems focus on implementing use case scenarios of resource intensive applications such as speech recognition, image processing and augmented reality, which mobile devices are unable to run at their own. While designing a CFS for a use case scenario, these systems do not consider an area of application where a large number of users could be present to use the system i.e. study the scalability of the system. The area of application of a CFS could range from a smart home (one or a few mobile devices) to a cafe or a restaurant (around 10 – 40

*Corresponding Author

Email address: `[manjinder.nir, ashraf.matrawy]@carleton.ca` (Ashraf Matrawy)

mobile devices), a university department or a corporate office (around 100 mobile devices), or a conference hall or a festival (may be more than 200 mobile devices). The scalability of a CFS is a big challenge. Satyanarayanan [1] mentioned that issues with localized scalability in a CFS could result from (i) multiple interactions from users' mobile devices to the service nodes of a system, or (ii) presence of multiple users in the cyber foraging area. The second issue gives notion of an application area of a CFS where we expect large number of users.

In this paper, we focus on scalability of a CFS, since an application area of a CFS could be an area where we expect a large number of mobile devices seeking to use the system. In the situation of a large number of users there could be a high density of users over the entire area, or over small part(s) of the area, which may lead to the formation of cluster(s). This situation could cause congestion at the service nodes, and/or at the wireless network in some parts or in the entire area. In the congested area, the wireless access points may not provide enough bandwidth, which is required for beneficial offloading in cyber foraging, and the service nodes may not have enough resources for all the users in the area [2], [4], [5]. This situation could also cause issues for cyber foraging processes. Existing systems implement these processes in mobile devices. In a large CFS, employing dynamic task scheduling requires continuous running of resource monitoring processes in mobile devices, which may incur communication overhead in the wireless network since all the mobile devices in the system are contacting all the service nodes repeatedly. This will cause delays for the mobile devices that are waiting to get updated resource monitoring information from multiple service nodes through the congested wireless network. During these delays, the battery of the mobile devices may be drained due to continuous attempt to use the congested wireless network.

Our main contributions can be summarized as follows. We present the challenges that could arise due to the implementation of dynamic task scheduling processes in a large CFS. In this paper, we propose to manage cyber foraging processes at a centralized node outside the mobile devices, that we call the Broker. Our proposed framework could lower the communication overheads of CF processes during resource monitoring, and this should lower the resource monitoring time for the mobile devices. Our contributions in this paper are (i) to investigate the overheads of cyber foraging process at different parts of a large CFS, and (ii) to lower these overheads using our proposed broker-based framework.

The structure of the paper is as follows. In Section 2, we discuss the challenges of dynamic task scheduling processes in a large CFS. In Section 3, we present related work. We discuss our proposed work in Section 4. The hybrid simulation and emulation experimental setup for a large CFS is explained in Section 5, We discuss our results in Section 6, and conclude this paper in Section 7.

2. Dynamic Task Scheduling Challenges in a large CFS

In Cyber Foraging, various processes run when a mobile device seeks to offload its task to a service node [11]. These processes help the mobile device to find available service nodes, to monitor the local and the remote service nodes resources, to make an offloading decision i.e. task scheduling to an appropriate remote execution location, to partition the resource intensive parts of the application into subtasks which are to be executed on a remote service node, and to establish a trust between the mobile device and the remote service node.

There are situations where it is more beneficial for a resource-constrained mobile device to cyber forage in the immediate vicinity rather than to utilize mobile cloud resources and to utilize dynamic task scheduling approaches rather than static approaches. A brief overview of these concepts and the benefit of dynamic task scheduling for local cyber foraging follows in the next two paragraphs.

In a cloud environment, rich computing resources could have greater potential as remote service nodes for CF. However, there are some challenges in exploiting cloud resources as CF service nodes. Implementing privacy, security and reliability of cloud resources [12] incurs energy overheads. Moreover, the default connectivity of mobile devices through 3G network could result in high communication latency, low network bandwidth and more energy consumption than a WiFi network [5], [13], [12], [14]. Therefore, despite of the availability of huge resources in the cloud environment, localized CF could be better than mobile cloud in certain situations [15].

The task scheduling process in CF is broadly classified as a static or a dynamic process. A CFS employing static task scheduling process is a prepared CF environment. The mobile devices offload their tasks based on already defined static offloading policies [2], and current status of available local and remote resources is not ascertained. However, there may be situations in a changing CF environment that the current availability of resources at the mobile device and service nodes, or the current requirement of the application may change. Under these situations, static offloading decision may not be appropriate and remote execution may not be beneficial for the mobile device [2].

On the other hand, a CFS employing dynamic task scheduling process first decides whether offloading to a remote location is beneficial or not, based on the “current” information of resources and the offloading goal(s) [2], [4] (Figure 1). Therefore a resource monitoring entity in the task scheduler ascertains: (i) the actual resources and the current available resources at the mobile device and the service node(s), (ii) the network bandwidth and latency between the mobile device and the service node(s), and (iii) the size of the application’s code and input and output data. The task scheduling entity considers user defined offloading goals, which could be (i) the execution time of the offloaded application at a remote location, and (ii) the battery energy consumption of the mobile device when the application is executed at a remote location. The task scheduler estimates the resources consumed in the mobile device in transferring the application related data over the current network resources, and the remote execution time of the application or the application latency. Based on the estimations and the user defined offloading goals the scheduler decides the remote execution location.

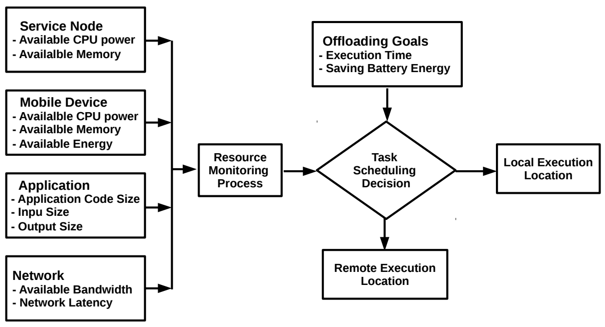


Fig. 1. Dynamic Task Scheduling Process

3. Related Work

Most of the existing CFS [2], [3], [4], [8] include task scheduler in the mobile devices itself. When we consider a large CFS, which employs dynamic task scheduling process in mobile devices, a large number of mobile devices have to communicate with multiple service nodes. In this situation, the overhead of task scheduling processes could be high in a large CFS. As discussed in Section 2, in dynamic task scheduling, the task scheduler needs up to-date information about the required and the available resources at the mobile device and the remote service nodes. Therefore, contacting the available service nodes by mobile devices is not a one time process. Whenever a mobile device has a new task to offload or there is a change in the required resources of the task, to get the situation of the resources, the mobile device repeats the same process of resource monitoring and task scheduling. The repeated communication between mobile devices and service nodes to perform resource monitoring will create excessive traffic in the wireless network, and it may cause congestion at the service nodes and the wireless network. Therefore, there may be cases where (i) all the available service nodes are not get contacted for resource description, or (ii) the resource monitoring time to get the resource description is large.

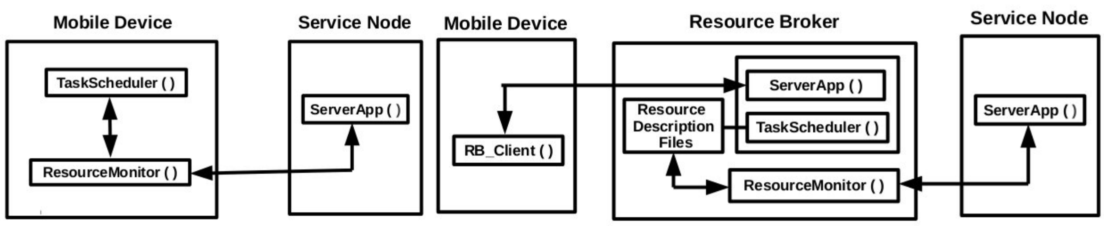
Some CFSs [5] [6] utilize a centralized place in a stationary computer. In the CFS [5], processes are implemented at centralized place to save the memory capacity, time and energy of mobile device to solve a call graph for the application partitioning. The CFS in [6] uses a stationary computer, called registry server, to find appropriate service nodes. There are systems [16], [17] in cloud computing environments, where a broker entity acts on behalf of service consumers to find service providers and negotiate for required resources on the basis of SLAs and QoS. Similarly, a resource broker entity in a grid computing environment [18] helps in finding the appropriate resources on the grid. To the best of our knowledge existing CF systems [2], [3], [4], [5], [6] and [8] do not consider resource monitoring and task scheduling overheads in a large cyber foraging system. Therefore, there is a need of a centralized entity employed outside the mobile devices, which could do resource monitoring and task scheduling on behalf of the mobile devices in a large cyber foraging system, and this is the focus of this paper.

4. Proposed Framework for Centralized Management of Cyber Foraging Processes

In this paper, we propose a solution to lower overhead of resource monitoring process in a large CFS. We propose to employ task scheduler and resource monitoring services at a centralized node instead of employing at individual mobile devices. The centralized node is referred to as 'Broker' node.

In the proposed framework, a mobile device wishing to offload its task first contacts the broker node. The broker has up to-date information of the available resources at the service nodes. The broker takes the current information of resources availability and requirement at the mobile device, and the offloading parameter(s) to be optimized by the user. Based on the information, the task scheduler in the broker node decides an appropriate remote execution location on behalf of the mobile device. In this way, the mobile devices do not communicate with all the service nodes, and moreover task scheduling is executed in the broker node. Therefore this solution may lower the delays in the mobile devices to get updated resource monitoring information from multiple service nodes, and lower communication overhead in the wireless network during resource monitoring.

To evaluate our proposed framework, we compare two scenarios of CFS: (i) a Baseline scenario (Figures 2(a)), and (ii) Centralized Broker scenario (Figure 2(b)). These scenarios differ at (i) the infrastructural level, (ii) the location of *TaskScheduler()* service, and (iii) the mechanism for the resource monitoring process. At the infrastructural level, the centralized broker scenario implements our proposed solution, and includes centralized broker node, however the baseline scenario does not include broker node. The *TaskScheduler()* service in baseline scenario is employed at mobile devices. However in the broker scenario this service is employed at the broker node, and mobile devices in this scenario employ a *RB_Client()* service, which communicates with the *TaskScheduler()* service in the broker node.



(a) (b)

Fig. 2. (a) Baseline Scenario (b) Centralized Broker Scenario

In both scenarios, the service nodes employ a *ServerApp()* service that accepts resource description requests from the *ResourceMonitor()* service. In the broker scenario, the broker node also employs the *ServerApp()* service to accept resource description requests from the *RB_client()* service in mobile devices. The request/response protocol between the *ResourceMonitor()* and the *ServerApp()*, or the *RB_Client()* service and the *ServerApp()* is same (Figure 3(a)). A client service requesting a resource description from a server first establishes a TCP connection with the *ServerApp()* through a three way handshake, whereby the client and server exchange $TCPSYN/ACK$ messages. On establishing the TCP connection the client sends a file request asking for the resource description of the service node, and the server in response sends an XML descriptor file. The client on downloading the requested file terminates the TCP connection. The downloading of a resource description file of size (approx. 200 bytes) should have very little impact on the resource monitoring time. The time taken by a mobile device to get resource description from one service node is the time interval between the instance t_1 the device send request for TCP connection and the instance t_2 when the mobile device receives the resource description file (Figure 3(a)). The *ServerApp()* service is modelled with a simple model of M/M/c/K queue with First-Come-First-Serve (FCFS) queueing discipline. The length of the queue is finite (K), and requests are serviced by multiple servers (c) (Figure 3(b)). We consider the arrival of the requests follows a Poisson distribution, therefore the inter-arrival time between the requests has an exponential distribution.

In general, when a mobile device wants to offload a task to a remote service node, the *TaskScheduler()* service in the device first finds an appropriate service node for the mobile device such that remote execution of the task is beneficial for the device. To find an appropriate service node the *TaskScheduler()* service first gathers resource description from all the available service nodes, network, the mobile device and the task. Then it estimates the cost of offloading based on the gathered information and the offloading goals seek by the mobile device user.

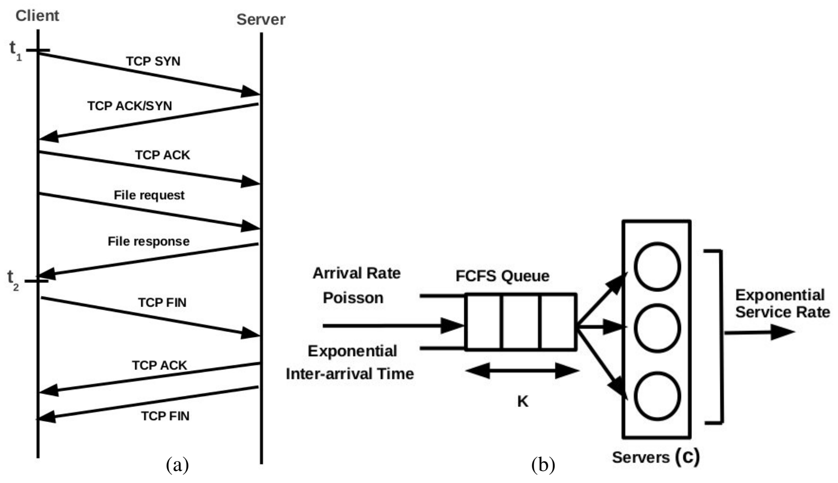


Fig. 3. (a) TCP Handshaking and Resource Description File Request/Response Protocol, (b) Service Model of *ServerApp()* Service

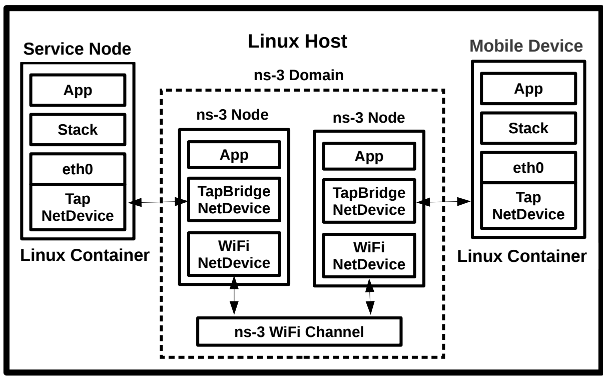


Fig. 4. Two Linux VMs representing a Service Node and a Mobile Device connected through *ns-3* WiFi network

In this paper, the objective is to investigate the overhead of resource monitoring process in a large CFS. Therefore, at this stage, we do not look at cost evaluation to decide a remote execution location and application offloading. In the baseline scenario, the *TaskScheduler()* service in mobile devices invokes the *ResourceMonitor()* service, which sends request to the *ServerApp()* service in all the available service nodes. In the broker scenario, the *RB_Client()* service in mobile devices first sends a request to the *ServerApp()* in the broker node, which further invokes the *TaskScheduler()* service to get resource description from the resource description files stored by the *ResourceMonitor()* service. The *ResourceMonitor()* service in the broker node is scheduled to be invoked periodically to get resource description from the service nodes.

5. The Hybrid Simulation and Emulation Experimental Setup

We have carried out simulations to compare CF setup for baseline (Fig. 2(a)) and centralized broker (Fig. 2(b)) scenarios. Our experimental setup for a large CFS is a hybrid of emulation and simulation techniques of Linux OS and ns-3 network simulation tool. The service nodes, the mobile devices and the broker node have been emulated using Linux virtual machines, also called as Linux container (LXC)[19]. This technique creates and allows to run multiple light weight virtual machines on the same host. A LXC combines resource management and resource isolation (*cgroup & namespaces*) of Linux kernal. The containers have their own private view of the OS, file system and network interfaces, and can be constrained to use a defined amount of resources such as CPU, memory or I/O.

We have used *ns-3* [20] network simulation tool. It has detailed wireless 802.11x models as compared to *ns-2* [21]. Moreover, in *ns-3*, *TAP* NetDevice integrates simulated CSMA or WiFi network and LXC. The *TAP* mechanism uses *TapBridge NetDevice* to make connections from *ns-3* to the LXc. As shown in Figure 4, the *TapBridge* arrangement connects the *I/O* of *ns-3 NetDevice* (WiFi *NetDevice in ns-3 node*) to the *I/O* of *TAP NetDevice* of Linux container, and is made to appear as if container is directly connected to a simulated *ns-3* network.

The setup in Fig. 4 shows a service node and a mobile device in Linux containers that are connected through a simulated WiFi network in *ns-3*. The simulation setup has multiple service nodes and mobile devices. The setup is implemented in a single server machine with Intel Xeon(R) E5420 @ 2.50GHz, quad core CPU and 8GB of RAM. The Linux OS distribution is Ubuntu 12.04.2 LTS precise. We set the CPU strength of service nodes higher than mobile devices using *cgroup* utility in Linux OS. The WiFi network in both the scenarios (Figs. 2(a) & 2(b)) is simulated using 802.11a model with its default parameters in *ns-3*. The *TaskScheduler()*, *ResourceMonitor()*, *ServerApp()* and *RB_Client* servcies are implemented in Java. We used Java multi-threading to send resource description request from a large number of mobile devices to multiple service nodes.

6. Evaluation

6.1. Simulation Experiments

In simulations, we measured resource monitoring time in a CFS having a large number of mobile devices and multiple service nodes. As defined in Section 4, that the time interval (t_2 - t_1) (Figure 3(a)) is the resource monitoring time taken by a mobile device to get resource description from one service node. In the simulations, we have measured the total resource monitoring time when a mobile device gets resource description from all the service nodes that it has contacted, and display the average of all the mobile devices. The total time is the time interval between an instance the mobile device starts sending resource description requests to the service nodes to the instance it has received resource description response from all the service nodes it has contacted. In our simulations, we did not consider the offloading processes, so the resource monitoring time in an actual situation could be less than the values we get in the simulations because some mobile devices may not be monitoring since they have already offloaded their computing requests.

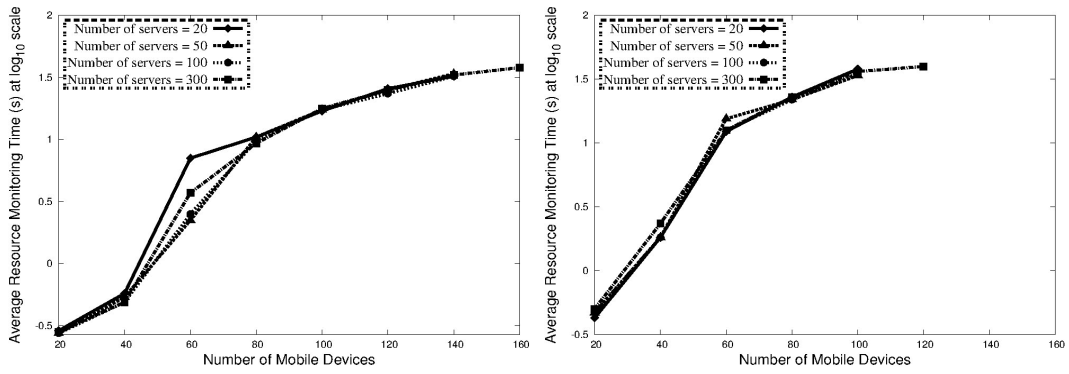
In the simulations, we wanted to study how resource monitoring is affected by: (i) the number of mobile devices currently doing resource monitoring, (ii) the number of service nodes to be monitored by the mobile device, (iii) the size of the wait queue, and (iv) the number of servers to serve the incoming requests in the queue of the *ServerApp()* service. Therefore in the simulations, we started by determining an appropriate value for the queue size and number of servers in the *ServerApp()*. Then we varied the number of mobile devices and the number of service nodes to see their effect (i) on the total resource monitoring time of the mobile devices, and (ii) on the scalability of the system. The scalability of the system is represented by how many mobile devices could do resource monitoring in the system.

The resource monitoring time is measured for different number of mobile devices when there are 3, 5 or 7 service nodes (SN) in the system. In the legends of figures the number of service nodes are denoted as SN3, SN5 and SN7, and the baseline scenario is denoted as Baseline, and centralized broker scenario as Broker. In each observation, all the mobile devices send resource description request to all the service nodes or Broker node over an interval of 15s. The queue discipline of the wait queue is *first-come-first-serve*

(FCFS), and the size of the queue is given by the number of incoming requests that could be queued when waiting to be served by the servers in the *serverApp()* service. In the results, the resource monitoring time of a mobile device is the average of 30 iterations, and each point is the average of resource monitoring time of all the mobile devices.

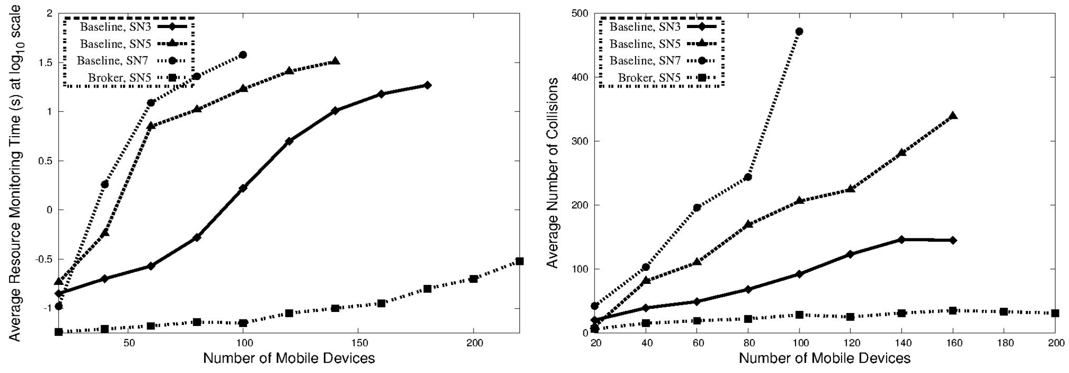
6.2. Analysis of Results

Effect of Queue Size and Number of Servers in ServerApp() service: While measuring the resource monitoring time, we have observed that even when considering small sizes (20, 50, 100) of a wait queue, the queue is never full and the incoming requests to the queue are never dropped with the increase in the number of mobile devices. Therefore we set the size of a wait queue at an arbitrary value of 500. The results in Figure 5 show the effect of varying the number of servers in the *serverApp()* service (described in Section 4 and Fig. 3(b)) on the resource monitoring time. We observed this effect in baseline scenario when there are 5 (Fig. 5(a)) or 7 (Fig. 5(b)) service nodes in the system, and in each case the resource monitoring time is observed by varying the number of mobile devices when the number of servers in the *serverApp()* are 20, 50, 100 or 300. The results in the figures show that for a given number of service nodes and mobile devices, the resource monitoring time values do not represent significant difference. Our observations show that when a mobile device does resource monitoring to a service node, the size of the wait queue, and the number of servers in the *serverApp()* of the service node do not effect the resource monitoring time. Therefore in further observations in both scenarios, we considered the size of a wait queue as 500, and the number of servers in a *serverApp()* as 20.



(a) Number of service nodes = 5 (b) Number of service nodes = 7

Fig. 5. Effect of Number of Servers in the *serverApp()* Service on the Resource Monitoring Time



(a) (b)

Fig. 6. (a) Comparison of Resource Monitoring Time and Scalability, (b) Collisions in the WiFi Channel, in Baseline and Broker Scenarios

Resource Monitoring Time and Networking Overhead: Further we compared the average resource monitoring time of mobile devices and the scalability in baseline and broker scenarios. Figure 6(a) shows the average resource monitoring time when there are 3, 5, 7 service nodes in baseline scenario. However, for the broker scenario we show average resource monitoring time when there are 5 service nodes, since we noticed a very small difference in the average resource monitoring time when there are 3, 5 or 7 service nodes in the broker scenario. The results in Figure 6(a) show that in baseline scenario the average resource monitoring time increases with increase in either the number of service nodes or the number of the mobile devices. However in broker scenario this time is not effected much with change in the number of service nodes, as mentioned above. Even the increase in the resource monitoring time with the increase in the mobile devices is negligible as compared to the baseline scenario.

During resource monitoring, the scalability of a large CFS using broker node is higher than the baseline system, and it is not effected with the change in the number of service nodes. However the scalability decreases in the baseline system with the increase in number of service nodes. In the broker scenario, Figure 6(a), we just showed resource monitoring upto 220 mobile devices when the system has 5 service nodes, however the scalability may go higher. On the other hand, in baseline scenario, when the number of service nodes are 3, 5 or 7 the number of mobile devices that could do resource monitoring are 180, 140 or 100 resp.

Thus the results show that the average resource monitoring time in our proposed framework (broker scenario) is much less than the baseline scenario. In the baseline scenario, the increase in the number of service nodes and mobile devices increases the resource monitoring, and decreases the scalability of the system. The reason for the degradation of performance in the baseline system is the increase in the amount of communications traffic between the service nodes and the mobile devices. However, this is not the case in a large CFS using our proposed centralized broker scenario, since for resource monitoring the mobile devices are only communicating with the broker node.

Effect of Wireless Network: The results in Fig. 6(b) reveal that in a large CFS, a large number of mobile devices and multiple service nodes cause congestion in the WiFi network. Consequently, the congestion in the WiFi network causes collisions, which accounts for (i) the increase in the resource monitoring time, and (ii) the decrease in scalability of the system. In a WiFi channel whenever there is a collision, the frame in MAC layer, which was ready to be transmitted to the channel is backed-off for random time. Therefore with increased collisions, the frames in the MAC layer are queued for a longer time. Figure 6(b) shows the average number of collisions in the WiFi channel during the resource monitoring process in both scenarios. The results show that in the baseline system, the average number of collisions increases with the increase in either the service nodes or the number of mobile devices. However, in the broker scenario, the number of collisions is small compared to the baseline scenario, and does not increase much with the increase in the number of mobile devices. The reason for the small number of collisions in the broker scenario is the lower amount of communications traffic between the broker node and the mobile devices as compared to the amount of communications traffic between the multiple service nodes and the mobile devices in the baseline scenario.

7. Conclusion and Future Work

In this work, we presented the challenges of CF processes in a large CFS. We proposed a framework for the centralized management of CF processes in the system (i) to lower the resource monitoring time and communication overhead, and (ii) to increase the scalability. We studied resource monitoring process in a large CFS to show the performance of the proposed framework. Our results shows that during resource monitoring in a large CFS, the large amount of communications traffic between mobile devices and multiple service nodes cause congestion in the WiFi network. The congestion increases the resource monitoring time and decreases the scalability of the system. However in our proposed framework, where the resource monitoring is handled by a centralized broker node on behalf of mobile devices, a lower amount of communications are generated between the mobile devices and the broker node, and consequently a smaller number of collisions in the WiFi network. Therefore, in our proposed framework the resource monitoring time is smaller and the scalability of the system is better than the baseline system. Thus our results show that centralized management of resource monitoring process can lower the communication overhead in the wireless network, and enhance the scalability of the system.

In future work, we plan to study how much of the energy of mobile devices could be saved through the centralized management of CF processes. Further, we will see how broker node in a large CFS could help mobile devices to schedule their tasks to an appropriate remote execution location, and how it could help service nodes in sharing their resources. Our results are based on simulations, in actual implementations there could be other impairments or factors affecting the performance. Therefore further tests in actual networks are needed before deployment to see if the same performance obtained in the simulations will be obtained in actual networks.

Acknowledgement
This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] M. Satyanarayanan, Pervasive Computing: Vision and Challenges, IEEE Personal communications 8 (4) (2001) 10–17.
[2] J. Flinn, S. Park, M. Satyanarayanan, Balancing Performance, Energy, and Quality in Pervasive Computing (2002) 217–226.
[3] R. Balan, M. Satyanarayanan, S. Park, T. Okoshi, Tactics-Based Remote Execution for Mobile Computing, in: Proceedings of the 1st international conference on Mobile systems, applications and services, ACM, 2003, p. 286.
[4] M. Darø Kristensen, Empowering Mobile Devices Through Cyber Foraging, in: Ph. D. thesis, 2010.
[5] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: Making Smartphones Last Longer with Code Offload, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010, pp. 49–62.
[6] S. Goyal, J. Carter, A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices, in: Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on, IEEE, 2005, pp. 186–195.
[7] Y. Su, J. Flinn, Slingshot: Deploying Stateful Services in Wireless Hotspots, in: Proceedings of the 3rd international conference on Mobile systems, applications, and services, ACM, 2005, p. 92.
[8] S. Kafaie, O. Kashefi, M. Sharifi, A Low-Energy Fast Cyber Foraging Mechanism for Mobile Devices, in: Arxiv preprint, 2011.
[9] H. Liang, H. Lutfiyya, A Cyberforaging Infrastructure Based on Web Services, in: Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on, IEEE, 2008, p. 59.
[10] S. Kafaie, O. Kashefi, M. Sharifi, Augmented Mobile Devices Through Cyber Foraging, in: Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on, IEEE, 2011, pp. 145–152.
[11] J. Porras, O. Riva, M. Darø Kristensen, Dynamic Resource Management and Cyber Foraging, Middleware for Network Eccentric and Mobile Applications (2009) 349–368.
[12] K. Kumar, Y. Lu, Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?, Computer 43 (4) (2010) 51–56.
[13] A. Miettinen, J. Nurminen, Energy efficiency of mobile clients in cloud computing, in: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, USENIX Association, 2010, pp. 4–4.
[14] M. Satyanarayanan, V. Bahl, R. Caceres, N. Davies, The Case for VM-based Cloudlets in Mobile Computing, IEEE Pervasive Computing.
[15] J. Flinn, Cyber Foraging: Bridging Mobile and Cloud Computing, Synthesis Lectures on Mobile and Pervasive Comp. 7 (2) (2012) 1–103.
[16] R. Buyya, R. Ranjan, R. N. Calheiros, Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, in: Algorithms and Architectures for Parallel Processing, Springer, 2010, pp. 13–31.
[17] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, Cloudsim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.
[18] D. Abramson, R. Buyya, J. Giddy, A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker, Future Generation Computer Systems 18 (8) (2002) 1061–1074.
[19] HowTo Use LXC, `http://www.nsnam.org/wiki/index.php/HOWTO_Use_Linux_Containers`.
[20] Network Simulator NS3, `http://www.nsnam.org/documentation/`.
[21] Network Simulator NS2, `http://www.isi.edu/nsnam/ns/`.