Computing in Cirrus Clouds: The Challenge of Intermittent Connectivity

Cong Shi, Mostafa H. Ammar, Ellen W. Zegura, and Mayur Naik
School of Computer Science
Georgia Institute of Technology, Atlanta, Georgia 30332
{cshi7, ammar, ewz, naik}@cc.gatech.edu

ABSTRACT

Mobile devices are increasingly being relied on for tasks that go beyond simple connectivity and demand more complex processing. The primary approach in wide use today uses cloud computing resources to off-load the "heavy lifting" to specially designated servers when they are well connected. In reality, a mobile device often encounters, albeit intermittently, many entities capable of lending computational resources. In this work-in-progress paper we first give an overview of this environment, which we call a Cirrus Cloud due to its intermittent connectivity feature, and explain how it provides a spectrum of computational contexts for remote computation in a mobile environment. An ultimately successful system will need to have the flexibility to handle intermittent connectivity and use a mix of options on that spectrum. We investigate two scenarios at the extremes of the spectrum: 1) a scenario where a mobile device experiences intermittent connectivity to a central cloud computing resource, and 2) a scenario where a mobile device off-loads computation to other mobile devices it might meet intermittently. We present preliminary designs, implementations, and evaluations of systems that enable a mobile application to use remote computational resources to speedup computing and conserve energy in these scenarios. The preliminary results show the effectiveness of our systems and demonstrate the potential of computing in Cirrus Clouds.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems —*Distributed applications*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Mobile Cloud, Computation Off-loading, Task Allocation, Intermittent Connectivity

MCC'12, August 17, 2012, Helsinki, Finland.

1. INTRODUCTION

Mobile devices are increasingly used to perform compute-intensive tasks ranging from the leisurely to mission-critical. These include pattern recognition to aid in identifying audio snippets or images captured locally or remotely, reality augmentation to enhance our daily lives, collaborative tasks that enhance distributed decision making, and planning and coordination, potentially in real-time.

Smartphones and tablets are examples of ubiquitous truly portable devices on which we have to come to rely heavily in everyday life. These devices are becoming increasingly powerful in terms of processing capability and storage. Battery power availability, while still constrained, continues to improve with each device generation. In addition, such devices enjoy the availability of improved connectivity options. These have enabled applications that transcend an individual device's capabilities by leveraging remote processing and storage.

Offloading computation from mobile devices to the cloud is in wide use today. Despite its success, however, it suffers from a number of shortcomings.These include the inflexibility in apportioning work between the mobile device and the remote cloud, the latency inherent in accessing remote cloud resources, and the inability to handle intermittent connectivity as a consequence of device mobility [7, 16]. A related technique for remote processing of mobile applications proposes the use of *cloudlets* which provide software instantiated in real-time on nearby computing resources using virtual machine technology [16] to address latency issues. Meanwhile, MAUI [8], CloneCloud [7] and ThinkAir [12] automatically apportion processing between a local device and a remote cloud resource based on the computation and communication environments.

None of the above approaches address the important and challenging problem of handling *intermittent connectivity* of mobile devices. Moreover, a mobile device often encounters many entities, including other mobile devices, capable of lending computational resources. Thus, an approach capable of handling connectivity disruptions is not limited to using the cloud and might be able to leverage this additional computational power.

In this work-in-progress paper we envision a generalized remote computing paradigm, called a Cirrus Cloud, that uses various compute resources both mobile and not, to enable its operation in the presence of intermittent connectivity. We posit that an ultimately successful system should have the flexibility to handle intermittent connectivity and use a mix of compute resources. As the first step towards this goal, we investigate how to use specific kinds of resources for remote computing in the presence of intermittent connectivity. Specifically, we report our current progress in two scenarios: 1) a scenario where a mobile device experiences intermittent connectivity to a central cloud computing resource, and 2) a sce-

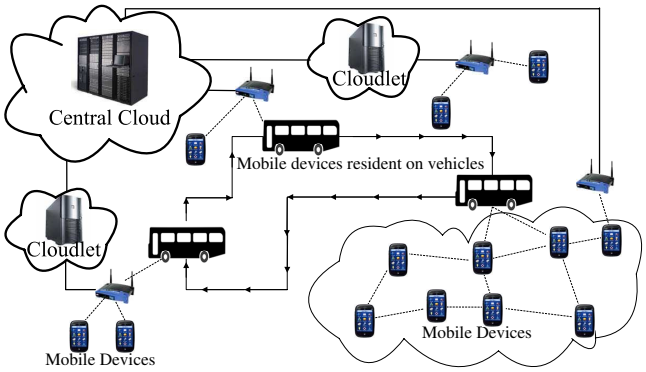


Figure 1: System components and network connectivity of Cirrus Clouds.

nario where a mobile device off-loads computation to other mobile devices it might meet intermittently.

The rest of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 presents our vision of Cirrus Clouds. Sections 4 and 5 present our preliminary work in the above two mobile scenarios, respectively. Lastly, Section 6 concludes with a note on future work.

2. RELATED WORK

Our work can be viewed as enabling a truly general vision of *cyber foraging* [3, 4] which envisions mobile applications "living off the land" by exploiting infrastructure-based computing resources. Because of the popularity of powerful mobile devices, it is now possible to extend the flexibility of this vision to include "foraging" of the available resources in other mobile devices as we propose to do in this work.

Our work also leverages recent advances in the understanding of data transfer over intermittently-connected wireless networks (also known as disruption-tolerant networks or opportunistic networks). These networks have been studied extensively in a variety of settings, from military [14] to disasters [9] to the developing world [15].

Our work is also related to systems that use non-dedicated machines with cycles that are donated and may disappear at any time. In this vein, our work takes some inspiration from the Condor system architecture [19]. Our work also resembles in part distributed computing environments that have well-connected networks but unreliable participation in the computation. Examples of these systems include BOINC [1], SETI@home [2], and folding@home [6], all leveraging the willingness of individuals to dedicate resources to a large computation problem. More recently, the Hyrax project envisions a somewhat similar capability to opportunistically use the resources of networked cellphones [13].

3. COMPUTING IN CIRRUS CLOUDS

In this section we describe our vision and the challenges of computing in Cirrus Clouds.

3.1 System Components and Network Connectivity

Mobile devices roam in a very rich communication and computing environment today, as shown in Figure 1. For our purposes we classify elements of this environment in the following four categories.

- User-carried mobile devices: Today such devices are typically smartphones or tablets. They are portable and thus experience significant mobility. Applications requiring computation are initiated on these devices.
- Mobile computing resources attached to moving vehicles: It is increasingly possible to piggyback computing resources on vehicles such as buses [18]. Such systems are not resource constrained since they can derive power from a vehicle's battery. They may also have somewhat predictable mobility patterns.
- Infrastructure-based computing resources: These are similar to *cloudlets* [16] in the sense that they are pre-provisioned computing resources that are accessible locally over a wireless access point.
- Central cloud resources: These are data centers that are always equipped and ready to undertake a particular computation task.

We are interested in enhancing computation in mobile user devices without constraining their mobility patterns. While user devices will, over time, come in contact with other devices that can provide computing resources, such contact will be intermittent and of potentially indeterminate duration. Two nodes within communication range of each other experience a contact opportunity that lasts for as long as they can hear each other. One way to characterize such connectivity is by describing the pattern of contact opportunities. During a contact, nodes can transfer data to each other. Both the duration and the transfer bandwidth of a contact are limited. A node can deliver data to a destination node directly if the latter is within radio range, otherwise delivery occurs through routing via intermediate nodes.

Intermittent connectivity could be totally unpredictable, only statistically predictable or sometimes highly predictable. Additionally control channels can be used when available to help predict contact opportunities [5]. We consider all these variations in our research.

3.2 Computational Contexts

Based on our categorization of the system components, we envision a spectrum of computational contexts, some of which are shown in Figure 2.

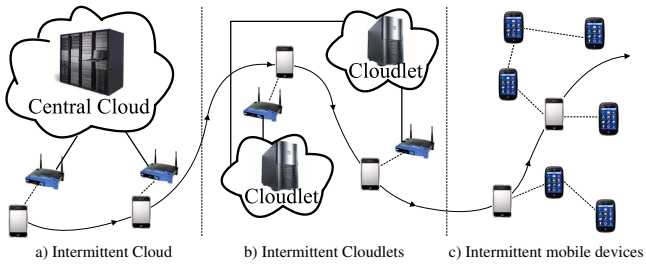


Figure 2: Senarios on spectrum of computational contexts in Cirrus Clouds.

At one extreme is the traditional cloud-computing context where a mobile device is intermittently connected to remote cloud resources maintained by a service provider with which it has an established relationship, as shown in Figure 2(a). We present our preliminary work on handling the intermittent connectivity in this scenario in Section 4.

Next on the spectrum is the use of cloudlets on which service software is dynamically instantiated to enable the off-loading of computation from the mobile device, as shown in Figure 2(b). The original cloudlet concept implicitly constrained the off-loading of computation to a single cloudlet. In our work we consider the case when the mobile device has contact with a number of cloudlet-like resources on an intermittent basis.

Moving along the spectrum, at the other extreme, we consider the case where a mobile device's contacts are only with other mobile devices, as shown in Figure 2(c). This is an extreme computational environment where both the computation initiator and the remote computational resources are mobile. We summarize our prior work on such environments in Section 5.

Our long-term work is based on the premise that an initiator mobile device's environment for remote computation will be, in the most general case, a *hybrid* of such systems. As such the device needs to learn about the capabilities of its environment and adapt its remote computation decisions accordingly.

3.3 Challenges

There are many challenges facing the realization of our vision as described above. We next outline these challenges.

Dealing with intermittent connectivity: This is one of the distinguishing features of our work and necessitates a fresh look at the issues concerned with remote execution of mobile services. Intermittent connectivity causes two types of problems. The first is the possibility that the duration of a contact may not be long enough to complete the process of handing off the computation subtask, waiting for it to complete on the remote device, and receiving the result. The second problem is the potential unpredictability of contacts. The manner in which the issue is addressed depends on the details of both the type of computation and the exact networking and computational context. In our description of preliminary work, we show examples of how these issues may be dealt with in specific scenarios.

Structuring computation to make it amenable to remote execution: Executing an application across multiple mobile or fixed devices with varying connectivity poses the challenge of how to partition the execution between the devices automatically, efficiently, and optimally. Today, a mobile application is either written in a monolithic fashion, accommodating as much functionality as possible on the mobile device; or in the traditional client-server paradigm, by delegating most computation to a remote server; or it is tailored to match an expected combination of client, environment, and service. Our setting, however, demands significantly more flexibility in how an application is partitioned: a partitioning that is optimal in one scenario (e.g., a low-end mobile device with good connectivity) might fare poorly in another (e.g., a high-end mobile device with intermittent connectivity). Often, such characteristics of the devices and network are unknown to programmers ahead of time, or the possible configurations are too numerous to customize.

Designing a decision-making framework for handing-off computation to remote resources: Addressing the application partitioning challenge above needs to be accompanied by a framework that enables adaptive and possibly real-time decisions of computation subtask allocation. Subtask profiling in terms of running time, communication requirements, and power consumption is a necessary input to enable allocation decisions.

4. INTERMITTENTLY CONNECTED CLOUD

In this section, we study the scenario that a mobile device experiences intermittent connectivity to a central Cloud, as shown in Figure 2(a). Specifically, we investigate how to extend CloneCloud [7] to work efficiently in that scenario.

CloneCloud [7] is a system that automatically partitions mobile applications and offloads the "heavy lifting" to the Cloud to minimize the execution time and energy consumption. It uses static analysis to identify legal choices for placing migration points in the code and dynamic profiling to construct a profile tree for every application. Each tree node represents a method that can be migrated and is annotated with its execution cost. Each edge represents the method call and is annotated with the data size to be transfered. At runtime, CloneCloud migrates an application from the mobile device at chosen migration points to the Cloud and re-intergrates it back after finishing execution.

Since CloneCloud uses the current communication condition to choose the migration points at runtime, it will experience extra delay when it intermittently connects to the Cloud. For example, if the mobile device loses its connection to the Cloud after migrating the application, it will have to wait for a long time to reconnect to it and re-intergrate the application back. In this situtation it may be better to execute the application locally. In another case, CloneCloud will decide to execute the application locally if the mobile device is currently disconnected to the Cloud. However, it is possible that shortly after that decision the connection is resumed. In this situation it may be better to wait for a while and migrate the application to the cloud.

To understand the impact of intermittent connectivity on CloneCloud, and as a first step in our preliminary investigation, we consider an ideal situation where the computation and future network connectivity are accurately known. This is useful to identify the best performance of the system with intermittent connectivity. In this setting, without any change to the CloneCloud system architecture, we propose a novel algorithm to choose the migration points by using the information of the network connectivity.

Algorithm 1 Find Migration Points

```
procedure FINDMIGRATIONPOINTS(root, t_0)   ▷ root is the root
of the profile tree of an application; t_0 is its start time.
    results = {}; t=t_0;
    for v ∈ root.children do
        results.addAll(FINDMIGRATIONPOINTS(v, t));
        t = computeCompletionTime(root, results, t_0);
    end for
    migrateAll = computeCompletionTime(root, {root},t_0);
    migratePart = computeCompletionTime(root, results, t_0);
    exeLocal = getLocalCompletionTime(root, t_0);
    if migrateAll ≥ exeLocal && migratePart ≥ exeLocal then
      return {};
    else if migrateAll ≤ migratePart then
      return {root};
    else
      return results;
    end if
end procedure
```

Algorithm 1 describes how to choose the migration points of an application that minimizes its execution time with intermittent connectivity. Given the profile tree of an application, the algorithm recursively chooses the optimal migration points from the root. At every node, it computes the completion times to decide if it executes the entire subtree locally, migrates it entirely to the cloud, or migrates some parts to the cloud. For the last choice, it iteratively applies the same algorithm to all the children of the node according to their execution order. The function $computeCompletionTime$ uses the information of computation time and future connectivity to compute the completion time. As stated in the following theorem, this algorithm finds the optimal partitioning. Because of space limit, we omit the proof in this paper.

THEOREM 1. *Given accurate information on the computation and future network connectivity, Algorithm 1 finds the optimal partitioning of the application that minimizes its execution time.*

4.1 Preliminary Experimental Results

We describe results from a set of preliminary experiments we ran to evaluate the performance of our algorithm and demonstrate the impact of intermittent connectivity.

To obtain realistic network connectivity traces for experiments, we rode a campus shuttle bus and measured the network connectivity between a tablet that we carried and the WiFi access points deployed on our campus. We also used the application *FaceRecognition* (whose profile tree was described in [8]) as the basic application. In all the experiments, we randomly choose the time to start the application and repeated every experiment 100 times. The reported results represent average values. We use three strategies as baselines: always running on the phone (i.e., labeled as Phone), always migrating the entire application to Cloud (i.e., labeled as Cloud), and CloneCloud. Our strategy which uses algorithm 1 to make offloading decision is labeled as CirrusCloud.

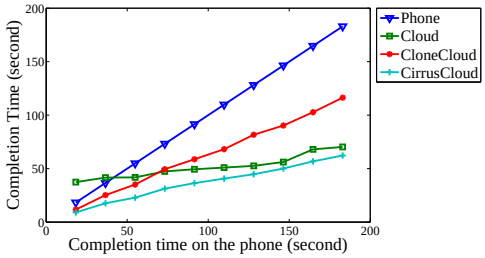


Figure 3: The impact of applications' execution time

In the first set of experiments, we evaluate the impact of computation on the performance of the migration algorithms by proportionally scaling the computation time of every method of the application. The computation time on the phone is assumed to be ten times of that in the Cloud, a typical value obtained from [7]. Figure 3 shows the results.

As expected, CirrusCloud performs best in all the experiments. With the increase of the computation complexity, the performance gap between CloneCloud and CirrusCloud is also increasing, while the Cloud strategy is approaching CirrrusCloud. This means that if the application's computation time is very small, it's good enough to use current network condition to make the migration decision. Meanwhile, if the application's computation time is very large, it's a good strategy to wait for the migration opportunity.

In the second set of experiments, we evaluate the impact of the computation power of Cloud resources on the performance of the migration algorithms by changing the speedup of the Cloud to the phone. Figure 4 shows the results.

CirrusCloud also performs best in all the experiments. Moreover, we make the following observations. First, when the speedup is large enough (e.g., 10 in the experiments), the completion times of all the systems converge. This is because the extra delay caused by intermittent connectivity dominates the completion time. Second, when the speedup is small (e.g., 2 in the experiments), always migrating to the Cloud will perform worse than running on the phone. This is because the low speedup cannot compensate for the extra network delay.

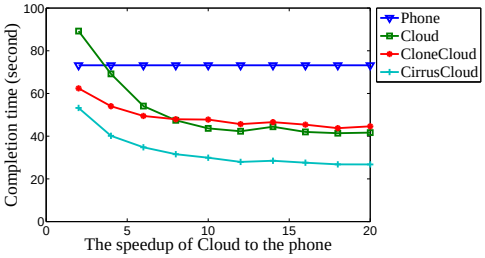


Figure 4: The impact of the speedup of the Cloud

4.2 Future Work

We will continue to investigate computation offloading in more practical settings (e.g., the intermittent connectivity is statistically predictable, or it is totally unpredictable). In such settings, besides the migration algorithm, we also need to change CloneCloud's system architecture to overcome the uncertainty of network connectivity. For example, besides migrating the application to the Cloud, the mobile device may also run a local clone to ensure that it will finish in time.

We will also study various computation offloading strategies for different kinds of applications to achieve the desired tradeoff between energy consumption and computation completion time. For example, some applications are delay tolerant, while others are real-time applications. These applications should be treated differently when offloading their computation. We will study how to express their requirements in a uniform framework and design the system to adaptively support these applications.

5. MOBILE NETWORKS AS A COMPUTING PLATFORM

In this section, we summarize our prior work on Serendipity [17], a system that enables a mobile computation initiator to use remote computational resources available in other mobile devices in its environment to speedup computing and conserve energy, as shown in Figure 2(c). This scenario represents the opposite extreme in turns of cmputation and network connectivity. Thus, the challenges call for a different solution.

Intermittent connectivity poses three key challenges for remote computing. First, because the underlying connectivity is often unknown and variable, it is difficult to map computations onto nodes with an assurance that the necessary code and data can be delivered and the results received in timely fashion. This suggests a conservative approach to distributing computation so as to provide protection against future network disruptions. Second, given that the network connectivity is intermittent, the network is more likely to be a bottleneck for the completion of the distributed computation. This suggests scheduling sequential computations on the same node so that the data available to start the next computation need not traverse the network. Third, when there is no control channel, the network cannot be relied upon to provide reachability to all nodes as needed for coordination and control. This suggests maintaining local control and developing mechanisms for loose coordination.

In response to these challenges, we propose a computation model and various subtask allocation strategies based on it.

5.1 Task Allocation in Serendipity

Different from the tree-structured applications supported by CloneCloud, an application supported by Serendipity is described as a directed acyclic graph whose nodes are *PNP-block*s. A PNP-block is composed of a *pre-process* program, n parallel *task* programs and a *post-process* program. The pre-process program processes the input data and passes them to the tasks. The workload of every task should be similar to each other to simplify the task allocation. The post-process program processes the output of all tasks. All pre-process and post-process programs can be executed on one initiator

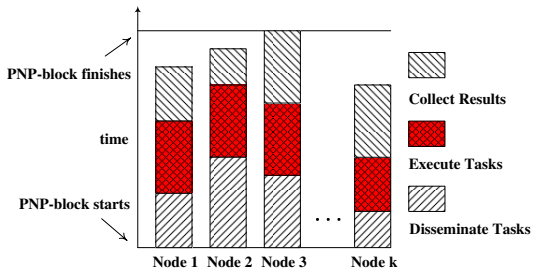


Figure 5: The PNP-block completion time is composed of a) the time to disseminate subtasks, b) the time to execute subtasks and c) the time to collect results.

device, while parallel tasks are executed independently on other devices. The data transfer delay can be reduced as the initiator device can choose nearby devices to execute tasks.

Efficient subtask allocation algorithms will reduce the PNP-block completion time and, thus, the entire computation completion time. Figure 5 illustrates the timing and components of a PNP-block execution. Along the x-axis are the k remote nodes that will execute the parallel subtasks of the block. Along the y-axis is a depiction of the time taken at each remote node to receive disseminated subtasks from the initiator, execute those subtasks, and provide the result collection back to the initiator. Our goal for subtask allocation is to reduce the completion time of the last subtask which equals to the PNP-block completion time. We consider three types of intermittent connectivity and design subtask allocation algorithms for them.

Predictable Connectivity w/ Control Channel: We first consider an ideal setting where the future contacts can be accurately predicted, and a control channel is available for information sharing.

Dijkstra's routing algorithm for DTNs [11] uses future contact information to compute the required data transfer time. The control channel is used to obtain the time and number of subtasks to be executed on the target node with which to estimate the time to execute a subtask on that node. Therefore, given the starting time and the target node, the subtask completion time can be accurately estimated. Using this information, we propose a greedy allocation algorithm, *WaterFilling*, that iteratively chooses the destination node for every subtask with the minimum completion time.

Predictable Connectivity w/o Control Channel: It is impossible to reserve subtask execution time in advance without a control channel. WaterFilling will cause contention for execution among different computations on popular nodes. To solve this problem, we propose an algorithm framework, Compute on Dissemination (CoD).

The basic idea of CoD is that during the subtask dissemination process, every intermediate node can execute these subtasks. Instead of explicitly assigning a destination node to every subtask, CoD opportunistically disseminates them among those encountered nodes until all finish. Every time two nodes meet, they exchange a set of subtasks to minimize their completion time. The key function of the framework is how to exchange the subtasks.

Unpredictable Connectivity: We develop a version of the CoD subtask allocation algorithm for unpredictable contacts (upCoD). It is still based on CoD with the constraint that future contact information is unavailable. Every time two nodes meet, they exchange a set of subtasks to minimize their execution time.

By replacing the time with an energy aware utility function, we can adapt these algorithms to optimize energy use.

5.2 Experimental Results

We built a testbed on Emulab which loads the node contact traces to emulate intermittent connectivity. Two contact traces, Haggle [10] and RollerNet [20], are used in the experiments. To evaluate our ideas, we implemented Serendipity on the testbed and tested a speech-to-text application that translates audio to text.

We carried out an extensive set of experiments. A complete analysis of our results can be found in [17]. We describe below a sample of those results that explore the impact of job size and contact traces. We use three audio files of different sizes, i.e., 20 Mb, 200 Mb, and 600 Mb.

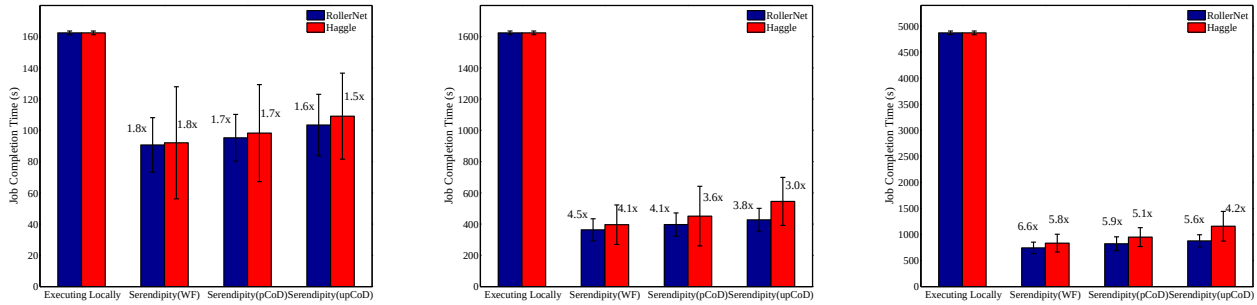
Figure 6 shows how Serendipity improves performance compared with executing locally. We make the following observations. First, as the workload increases, Serendipity achieves greater benefits in improving application performance. When the audio file is 600 Mb, it can achieve as large as 6.6 and 5.8 time speedup. Considering the number of nodes (11 for RollerNet and 9 for Haggle), the system utilization is more than 60%. Second, the ratio of the confidence intervals to the average values also decreases with the workload, indicating all nodes can obtain similar performance benefits. Third, in all the experiments WaterFilling consistently performs better than pCoD which is better than upCoD. In the Haggle trace of Figure 6(c), WaterFilling achieves 5.8 time speedup while upCoD only achieves 4.2 time speedup. The The results indicate that with more information Serendipity can perform better.

6. DISCUSSION AND FUTURE WORK

To summarize, our work explores paradigms for enabling computationally intensive mobile applications that make opportunistic use of available resources in nearby cloud/cloudlet infrastructure or other mobile nodes. It is based on the following observations: 1) The computational need of mobile devices often exceeds an individual device's capability, 2) The current trends in mobile devices point to increasingly capable mobile resources, with some devices deployed on vehicles or other mobile infrastructure providing high-end capability, 3) Often, mobile applications are collaborative or replicated in nature making it natural to involve multiple mobile devices in a particular computation, and 4) Mobile devices experience *intermittent connectivity*, an inherent feature in how they communicate with each other as well as with fixed infrastructure.

We described a spectrum of Cirrus Cloud Computing contexts (Figure 2) and we presented our preliminary work on two points on that spectrum, demonstrating the benefits in power consumption and task completion time.

Our work will continue its consideration of the entire spectrum of computational contexts. Our ultimate goal is to develop an overarching framework for Cirrus Cloud Computing. More specifically, in addition to continuing to work on the scenarios described in Sections 4 and 5, we will consider the intermittently-connected cloudlet scenario shown in Figure 2(b). In this scenario, mobile devices are in intermittent contact with multiple (stationary) cloudlet resources over time. This scenario adds the multiple compute resource dimension to the problem. This environment presents additional concerns beyond the problem of partitioning for remote execution. Among the problems is how to provide for continued execution if a mobile device loses contact with a cloudlet before it completes the processing of an allocated task. Depending on the contact model there is the possibility that the mobile device will meet the same cloudlet again, in which case results can be obtained at this subsequent meeting. Alternatively, we can consider a "hand-off" process where data and computation are migrated among cloudlets over the Internet in anticipation of future contacts with the initiator device.



(a) 10 tasks (b) 100 tasks (c) 300 tasks

Figure 6: A comparison of Serendipity's performance benefits. Serendipity(WF), Serendipity(pCoD) and Serendipity(upCoD) corresponds the three scenarios discussed in Section 5.1. The average job completion times with their 95% confidence intervals are plotted. We use two data traces, Haggle and RollerNet, to emulate the node contacts and three input sizes for each.

Ultimately, we envision that the environment in which a user-carried mobile device operates will be a hybrid of the individual scenarios we previously discussed. The goal of our work in general is to allow the device to leverage the combination of resource types available in its environment from which it derives the most benefit while adhering to the constraints of the environment (such as willingness of other devices to contribute resources). There are two types of such environments. The first is where multiple options for remote computation are available contemporaneously. In this case the question is how to optimally use the mixture of available resource types to maximum benefit. The second type is where the environment can change over time. For example the initiator device can encounter a cloudlet for some time and then be in an environment with a number of neighbor user-carried devices some time later. In this case, the environment needs to be monitored and strategies for adaptation of the computation off-loading need to be adopted.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful feedback. This work was supported in part by the US National Science Foundation through grant CNS 0831714.

7. REFERENCES

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *IEEE/ACM GRID*, 2004.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 2002.

[3] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *ACM SIGOPS European workshop*, 2002.

[4] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. In *ACM MobiSys*, 2007.

[5] N. Banerjee, M. D. Corner, and B. N. Levine. Design and Field Experimentation of an Energy-Efficient Architecture for DTN Throwboxes. *IEEE/ACM Transactions on Networking*, 2010.

[6] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande. Folding@home: Lessons from eight years of volunteer distributed computing. In *IEEE IPDPS*, 2009.

[7] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the 6th European Conference on Computer Systems (EuroSys'11)*, pages 301–314, 2011.

[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *ACM MobiSys*, 2010.

[9] K. Fall, G. Iannaccone, J. Kannan, F. Silveira, and N. Taft. A disruption-tolerant architecture for secure and efficient disaster response communications. In *ISCRAM*, 2010.

[10] P. Hui, J. Scott, J. Crowcroft, and C. Diot. Haggle: a networking architecture designed around mobile users. In *WONS*, 2006.

[11] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 34:145–158, August 2004.

[12] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *IEEE Infocom*, 2012.

[13] E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. Master's thesis, Computer Science Dept., CMU, September 2009.

[14] P. Marshall. DARPA progress towards affordable, dense, and content focused tactical edge networks. In *IEEE MILCOM*, 2008.

[15] A. S. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking connectivity in developing nations. *Computer*, 2004.

[16] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 2009.

[17] C. Shi, V. Lakafosis, M. Ammar, and E. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *ACM MobiHoc*, 2012.

[18] H. Soroush, N. Banerjee, A. Balasubramanian, M. D. Corner, B. N. Levine, and B. Lynn. DOME: A Diverse Outdoor Mobile Testbed. In *ACM HotPlanet*, 2009.

[19] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurr. Comput. : Pract. Exper.*, 2005.

[20] P. U. Tournoux, J. Leguay, F. Benbadis, V. Conan, M. D. de Amorim, and J. Whitbeck. The accordion phenomenon: Analysis, characterization, and impact on dtn routing. In *Proc. IEEE INFOCOM*, 2009.