Enabling Cyber Foraging for Mobile Devices

Mads Darø Kristensen
Center for Interactive Spaces, ISIS Katrinebjerg
Computer Science Department, University of Aarhus, Denmark
Email: madsk@daimi.au.dk

Abstract

This paper presents the LOCUSTS *framework. The aim of the* LOCUSTS *project is to enable easy use of cyber foraging techniques when developing for small, resource-constrained devices. Cyber foraging, construed as "living off the land", enables resource poor devices to offload tasks to nearby computing machinery, thereby enabling the small devices to 1) save energy and time, 2) take on tasks that would normally not be possible on such small devices, and 3) co-operate to perform tasks.*

This paper is concerned with foraging for processing power, i.e. remote execution of tasks, and discusses how distribution and migration of tasks can be done in a highly mobile environment.

The main contribution of LOCUSTS *is the focus on highly mobile cyber foraging. Here highly mobile means two things: 1) that the mobile devices are physically moving through the environment, which calls for task migration, and 2) that this mobility moves the devices into unknown environments where they would still like to be able to perform cyber foraging, which calls for the use of mobile code.*

1. Introduction

In recent years the usage of small mobile devices has increased dramatically. Today most people own a mobile phone, many have PDAs, and other forms of mobile devices, such as electronic gaming devices, are gaining in popularity. A shared characteristic of these devices is that they are in some way resource-constrained devices, if for no other reason then at least because they are all battery powered. Because of the scarcity of resources users of mobile devices often avoid doing resource intensive work when using these devices, because executing such tasks is very slow and consumes a disproportional amount of energy.

Cyber foraging, as described by Satyanarayanan [1] and Balan *et al.* [2], enables the mobile devices to take on more resource intensive tasks by leveraging unused resources on larger computers in the vicinity. Cyber foraging is foraging for a multitude of resource types – not just processing power. Among the resources that could be foraged for is network connectivity or bandwidth, storage, processing power, and much more. All of these resource types are equally important in a cyber foraging scenario. In the present article, however, only foraging for processing power will be considered.

There are many possible usage scenarios where cyber foraging can be utilised. Some visions for pervasive computing calls for *wearable computing* devices - i.e. small computing devices that may be worn by their users like clothes, e.g. see [3]. Users of such devices are obviously not interested in carrying around heavy equipment, and these devices must therefore be as lightweight as possible. This is counter to the user's wish to have as powerful a device as possible. The desired computing power can be added to these small wearable devices through techniques such as cyber foraging.

Consider the following scenario: a doctor doing house calls is wearing a small headset (similar in size and form to the well-known Bluetooth headsets for mobile phones). Using this headset he would like to be able to enter information about his patients into an electronic journal. This means that the headset is faced with the difficult task of continuous voice recognition. The headset is unable to perform this translation task itself, so instead of performing the actual voice recognition it merely records the utterances made by the doctor. Whenever the headset comes within range of usable computing resources (surrogates) it forwards some of the recordings to these machines who respond by returning the translated text. If the surrogate has an Internet connection it may even be given the task of updating the patient's journal directly. After translation the headset may discard the recording and

thus free storage for additional recordings.

A notable thing in the preceding scenario is that the application running on the mobile device works in two modes; *high fidelity* and *low fidelity*, as defined by Noble *et al.* in [4]. When no surrogates are within range the headset simply stores the recordings (low fidelity), and when surrogates can be used the recordings are immediately translated into text (high fidelity). This high/low fidelity aspect is inherent in all cyber foraging applications – when surrogates are available high quality work may be done, but this does not mean that the applications will only work in the presence of surrogates. For cyber foraging to be usable a low fidelity setting must also be possible, where the mobile device itself is running the application, albeit at a diminished fidelity. In the scenario low fidelity means that the headset only stores the recordings, but it would also be possible to ask the mobile device to do the processing itself, or even to do it in conjunction with other mobile devices that reside in the doctors personal area network.

To be able to perform the actions described above a number of things are needed. First off the mobile device must be able to monitor the network looking for any available surrogates. Once found the mobile device must be able to distribute tasks to surrogate machines, and, in the case that the user is moving while tasks are being performed, surrogates must be able to migrate tasks between each other so that the result may be returned. These are fairly complex operations, and it should not be the responsibility of the application programmer to implement this. LOCUSTS aims to create a toolbox so that a developer just needs to mark the pieces of code that could be distributed, and then the rest will be taken care of by the framework; distributing computation to and migrating tasks between surrogates as needed. Such frameworks have been proposed before [5] [6], but these approaches have no provisions for distribution of code or task migration, and as such they do not consider the high level of mobility and flexibility that LOCUSTS caters for.

In Section 2 the main challenges faced in cyber foraging are presented, then, in Section 3, the design of the LOCUSTS framework is presented and its architecture is described to show how these challenges will be met. Related work is discussed in Section 4 and the paper is concluded in Section 5.

2. Cyber Foraging Challenges

There are a number of challenges that must be addressed when designing a framework for cyber foraging. In this paper only the two challenges most central for the remote execution of tasks will be discussed:

- **Task distribution.** How can tasks be delegated to surrogates, and exactly *what* should be moved onto the surrogates.
- **Task migration.** When mobile devices are using surrogates the tasks that are distributed to surrogates must be migratable – i.e. it must be possible to move running tasks between surrogates and also to move a task back to the mobile device.

Apart from these remote execution specific challenges a number of other challenges are posed as well in a cyber foraging framework, challenges such as device discovery, capability announcement, data staging, etc. These are of course covered in the design of LOCUSTS, but are not described in any detail here.

One final critical challenge for cyber foraging is security. Surrogates must execute code on behalf of, possibly unknown and thus untrusted, mobile devices, and data must be transmitted over wireless links that are easy for an eavesdropper to monitor. Finally, the client must be able to trust that the surrogate actually performs the task that it is asked to. How can this be done in a secure manner? A fine balance between security and flexibility must be found here. A full description of how this is handled in LOCUSTS is beyond the scope of this paper.

3. The LOCUSTS Framework

The LOCUSTS framework aims to provision developers with a complete cyber foraging toolbox that can ease the process of developing applications that utilise cyber foraging. In the following the architecture of LOCUSTS will be briefly described in Section 3.1, then the chosen approach towards task distribution is described in Section 3.2. Finally, task migration is illustrated in Section 3.3.

3.1. Architecture

A simplified view of the current architecture of LOCUSTS is depicted in Figure 1. The LOCUSTS daemon is running as a separate process on both client and surrogate devices, and the individual applications can communicate with the local LOCUSTS instance. As shown, a cyber foraging enabled application consists of some local code, executed by the local device, and a number of distributable tasks. Whenever a task is executed, the local LOCUSTS instance is contacted so that it may find a suitable execution plan. This execution plan is created by the scheduler which relies on resource measurements, both local and remote, and

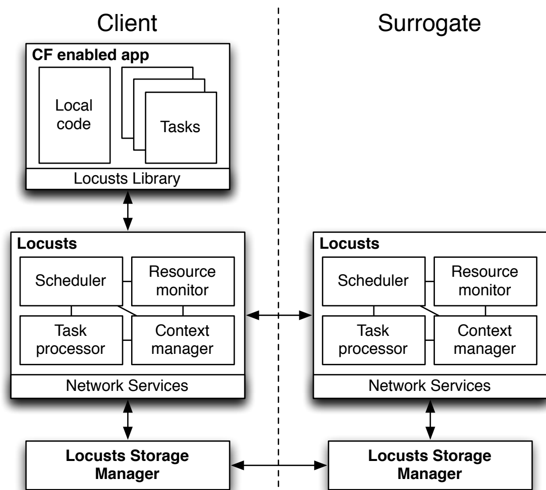

Figure 1. LOCUSTS architecture.

in some cases also on storage specific information such as input availability. When an execution plan has been derived the task may either be distributed to one or more surrogates, or it may be handed over to the task processor for local execution. At the bottom of the LOCUSTS client is the network services. These enable the mobile node to do necessary P2P operations such as peer discovery, network roaming etc. All devices can choose to act as surrogates and thus the software running on surrogates and clients is the same. When operating as a surrogate, a device basically offers three things: 1) to execute known tasks an behalf of clients, 2) allowing clients to author new tasks, and 3) full task migration support. Describing these three subjects fully is out of the scope of this paper, but a short description is given in the following sections.

The storage manager shown below the LOCUSTS daemon in Figure 1 provides a simple file system that can be accessed by tasks executed by LOCUSTS. The storage manager provides a virtual file system that can be accessed from within tasks. When executing a task on the local device, files in this virtual file system simply point to the local files, but when a task is delegated for remote execution, the storage managers of the client and the surrogate are *linked*, so that remote files may be read transparently. This small distributed system is kept as simple as possible and is designed specifically for the purpose of cyber foraging. It uses on-demand synchronisation of file data to reduce the amount of data transferred, and has built in support for temporary files that will only be synchronised if the task is migrated.

3.2. Task Distribution

Task distribution is at the core of cyber foraging; the delegation of heavy work to surrogates is the whole idea of cyber foraging. When designing a cyber foraging framework it must be decided exactly what is delegated, when it is delegated, and at which granularity.

The question about task granularity is a hard one to give a definite answer to, since it depends on a number of factors that may vary from system to system. The main factors to consider are network bandwidth and latency, processing power of the mobile device and the surrogate, the amount of energy used at the mobile device when communicating with the surrogate, and the velocity of the mobile device. When delegating tasks to a surrogate the mobile device needs to send the task to the surrogate, and likewise the surrogate must transmit a response back to the mobile device. This means that data must be transmitted over the wireless link between the mobile device and the surrogate. It must be considered whether the cost of this transmission, both in time and energy, is acceptable, i.e. whether the cost of distributing the task is smaller than the cost of doing the processing locally. To this end it makes sense to distribute only larger, longer running tasks, since the cost of delegating a small task will exceed the cost of local execution. But what designates a small task? This will vary from situation to situation depending on all the factors mentioned above.

To approach this challenge, decisions about when to distribute a task must be taken dynamically depending on the current resource availability. This is done by monitoring resource usage at the client and surrogates and using this information in the scheduler when planning future execution. This is also the approach taken in existing remote execution systems such as Spectra [5] and Chroma [6]. The LOCUSTS framework takes the same approach towards task distribution; monitoring resource usage and dynamically deciding where and when to distribute tasks. Aside from resource monitoring and subsequent planning LOCUSTS also works with the concept of resizable tasks. A resizable task is a task that can be solved to different degrees, in some ways similar to *fidelity* as introduced Noble *et al.* in [4]. But, apart from being able to solve the task at different fidelities, a resizable task in LOCUSTS may also be solved to a certain degree, meaning that a surrogate may choose to solve only a small fraction of the task before returning the task to the client. This will normally be done when the surrogate is subject to timing constrains given by the client, e.g. if a highly mobile client only allows the surrogate to use one second on the task to make sure that

it will receive the answer before going out of range. Related to this, LOCUSTS also works with the concept of migratable tasks as will be described in Section 3.3.

The next important question to answer is exactly *what* is distributed and how it is done. When a mobile device is running an application that is capable of utilising cyber foraging, a part of the application will always be running locally while other parts may or may not be distributed to surrogates. Just like when working with parallelising programs, the task of preparing a program for distribution requires some work by the developers of the program. The parts of the program that can be distributed must be identified and, possibly, altered to make the distribution possible.

After identifying the parts of a programs that can be delegated to surrogates a mechanism for actually distributing these tasks must be found. Existing cyber foraging systems use RPC for remote execution, and the surrogates must have the software behind these RPCs installed prior for the clients to be able to utilise them. LOCUSTS differs from these systems because it strives to do away with the need to have anything pre-installed on surrogates. A task in LOCUSTS is therefore more than just an RPC invocation, it also contains the actual source code of the task represented in a way such that any surrogate, regardless of architecture etc., will be able to execute it. This means, that the portions of the code that designate distributable tasks must be written in a specific, interpreted language so that it can be moved on to surrogates, and thus allowing the clients to *author* new tasks on the surrogate. Allowing clients to execute unknown code on surrogates of course leads to an abundance of security issues that will have to be addressed, but that is out of the scope of this paper. Currently the language used for distributable tasks is Python but other interpreted languages could be used as well.

3.3. Task Migration

Distributing very large tasks increases the benefits of remote execution, since it helps to diminish the overhead of sending tasks back and forth. But, in existing cyber foraging frameworks, working with large tasks requires the user to stay within range of a specific surrogate for an extended period of time. To alleviate this problem, provisions have to be made so that tasks may span multiple surrogates throughout their lifetime. The solution to the problem is task migration. Task migration enables surrogates to move running tasks to other surrogates or even back on to the mobile device. Using migration a client no longer needs to stay within range of a surrogate while performing a task, and it is thus possible to distribute larger tasks, which alleviates the considerable overhead of remote execution. Task migration is depicted in Figure 2.

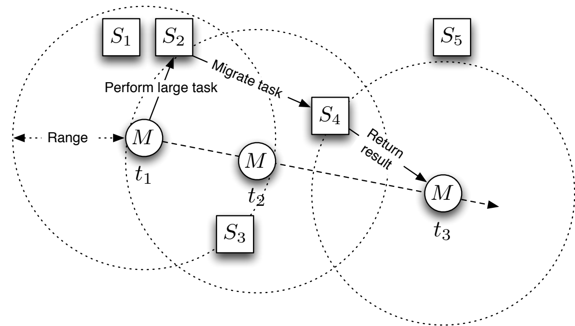

Figure 2. Migration of a single task. In this scenario a mobile device M is depicted at time t_1, t_2, and t_3. A task is initiated on surrogate S_2 at time t_1, migrated to S_4 at time t_2 when M moves out of range, and finally the result is returned by S_4 at time t_3.

The ways that such task migration could be implemented range from simple surrogate-to-surrogate proxies to task checkpointing. Both methods are used in LOCUSTS. Proxies are used in some circumstances when high speed network connections exist between surrogates. Take for example the scenario in Figure 2. The task is initiated at S_2 but when M moves out of range of S_2 the task is migrated to S_4. If, for some reason, it makes sense to let S_2 keep the task S_4 will simply be asked to proxy for S_2. In the eyes of the client M surrogate S_4 is the one executing the task and all communications regarding the task goes through S_4. Alternatively, the task could be moved entirely to S_4. In this case the running task would be checkpointed by S_2 and its code and state sent to S_4. A multitude of factors must be considered when choosing which kind of migration to use – factors such as network bandwidth between the surrogates, current resource usage at the surrogates, checkpoint size, estimated finish time of the task etc.

This preceding description of task migration touches lightly on a very complex matter – the full complexity of task migration and how it is implemented in LOCUSTS is outside the scope of this papers.

4. Related Work

Remote execution of tasks is a well-studied research field, but remote execution of tasks done by mobile, resource constrained devices is less so. Spectra, described by Flinn *et al.* [5], and Chroma, described by Balan *et al.* [6], are two related examples of remote execution frameworks that consider these factors. These

systems include very sophisticated schedulers that try to dynamically find the best execution plans for a given task execution. They measure resource usage on different levels, both at the client, at the surrogates, and in the network, and thus try to choose the best possible placement of the tasks. Spectra and Chroma thus solve many of the problems when using remote execution in a cyber foraging setting. One shortcoming of these systems, when considered in the highly mobile usage scenarios envisioned for the LOCUSTS framework, is that they do not provide any means for dynamically distributing the *code* of the tasks. Surrogates must therefore be prepared beforehand to enable the execution of tasks. Furthermore, they do not consider migration of tasks between surrogates. Likewise, Spectra and Chroma use the Coda [7] filesystem for data staging, which, in a highly mobile scenario such as the ones LOCUSTS aims to support, would be too complicated.

The Coign system, described by Hunt and Scott in [8], makes distribution of tasks possible without even altering the source code of the application. But, as it is also noted by Flinn *et al.*, a little application-specific knowledge can go a long way when preparing an application for distribution. In many cases the inclusion of distribution alters entirely the way to think about a given program, e.g. it may make sense to execute parts of an initially linear program in parallel (possibly even on multiple surrogates). Such optimisations would be hard to detect for an automated distribution algorithm. Ideally a cyber foraging framework should cater for both kinds of distribution, falling back to automatic distribution when no instrumented version of an application is available. The Coign system only works on applications consisting of Microsoft COM components, and is therefore limited to distributing applications running on different versions of Microsoft Windows. This is a big limitation in a cyber foraging setting where many different kinds of mobile devices must be supported. Furthermore, Coign is *not* a cyber foraging framework – it is only concerned with the partitioning of applications.

5. Conclusion

This paper presents the LOCUSTS framework. The LOCUSTS framework extends cyber foraging to encompass highly mobile foraging for resources in unknown environments. The focus on unknown, or unprepared, environments leads to new challenges in task distribution since unknown code has to be distributed to untrusted surrogates. The focus on mobility means that task migration becomes a necessary part of LOCUSTS. This has to our knowledge not been studied in detail before in a cyber foraging setting.

LOCUSTS is still a work in progress and much work and experimentation needs to be done. The most challenging future questions are: 1) how can code be distributed safely in an untrusted environment? 2) how will task migration perform compared to converting the problems into smaller tasks? 3) Which features are needed in a minimal distributed file system to fully support usage in a cyber foraging setting?

Acknowledgements

This paper has been funded by a research grant from the Danish Research Council for Technology and Production Sciences.

Furthermore, I would like to thank Niels Olof Bouvin for proof-reading and providing helpful insights into the subject matter.

Finally, I would like to thank the anonymous reviewers for their excellent and very insightful comments. I hope to have honoured most of their requests for improvement in this version of the paper.

References

[1] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[2] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*. New York, NY, USA: ACM Press, 2002, pp. 87–92.

[3] S. Mann, "Wearable computing: a first step toward personal imaging," *Computer*, vol. 30, no. 2, pp. 25–32, 1997.

[4] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, vol. 31, no. 5. New York, NY, USA: ACM Press, December 1997, pp. 276–287.

[5] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pp. 217–226, 2002.

[6] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices."

[7] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: a highly available file system for a distributed workstation environment," *Transactions on Computers*, vol. 39, no. 4, pp. 447–459, 1990.

[8] G. C. Hunt and M. L. Scott, "The coign automatic distributed partitioning system," in *Operating Systems Design and Implementation*, 1999, pp. 187–200.