Leveraging the Cloudlet for Immersive Collaborative Applications

Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt

***Abstract*—Cyber foraging has already shown itself to be successful for enabling resource-intensive applications on mobile devices. Offloading applications or parts thereof to nearby infrastructure – also denoted as cloudlets – can lower execution time and/or save energy. However, in view of immersive applications, two challenges still remain. First, due to the real-time constraints of immersive applications, optimizing generic metrics such as execution time or energy is not sufficient. Therefore we propose a component-based cyber foraging framework that optimizes application specific metrics, not only by offloading, but also by configuring application components at runtime. Second, as immersive applications tend to process location aware data, much processing is replicated when multiple users are in the same environment. Therefore, our framework also enables collaborative scenarios by sharing components between multiple devices.**

***Index Terms*—Cloudlet, Cyber Foraging, Augmented Reality**

I. Introduction

MOBILE devices are pervading people's daily lives. Many use a smartphone to access their e-mails on the road, read an e-book on their tablet, or use a laptop to work on the train. Due to recent advances in hardware technology, current mobile devices are not only gaining more memory and processing power, they are also packed with a large number of sensors. In combination with developments in near-to-eye display technologies such as Google Glasses, these advances will lead to the emergence of mobile immersive applications such as augmented reality [1]. By combining virtual 3D objects with the real world, such applications submerge the user in a new, mixed world to interact with. These applications utilize complex algorithms for camera tracking, object recognition etc. that require considerable computational resources.

Despite the increasing hardware capabilities, mobile devices will always be resource poor compared to their desktop or server counterparts. Due to additional limitations concerning weight, size, battery autonomy and heat dissipation, mobile devices simply cannot compete with fixed hardware. These resource limitations imply considerable obstacles for realizing mobile immersive applications, that typically require heavy and real-time processing. As a developer, one also has to cope with a high level of device heterogeneity, meaning that one has either to develop for the lowest common denominator, either to tailor the application for each device by providing a specific configuration.

In order to be able to run such resource-intensive applications on a mobile device, a solution is to offload the application or parts thereof to resources in the network, also known as cyber foraging [2]. With the emergence of cloud computing, processing power is nowadays available in the network as a commodity [3]. However, due to the high and variable latency to distant datacenters, the cloud is not always suitable for code offload, especially not for real-time immersive applications. Therefore, Satyanarayanan et al. introduced "cloudlets": infrastructure deployed in the vicinity of the mobile user [4].

Current cyber foraging systems face two major challenges for supporting immersive applications. First, most cyber foraging systems focus on optimizing generic metrics such as energy usage [5] [6] or execution time [7] [8]. Immersive applications however, often need to execute complex image processing algorithms in a timely manner, and thus it makes more sense to take into account application specific metrics such as the framerate, image resolution or the robustness of camera tracking. From a developer's perspective, the goal is then to achieve an acceptable framerate at the highest possible quality for each device, where "quality" is related to the image resolution and the tracking configuration. To address this issue, we propose a cyber foraging framework that not only decides on offloading, but also takes into account application specific configuration parameters in order to enhance application quality. We adopt a component-based approach, where application developers define software components with their dependencies, configuration parameters (i.e. image resolution, tracking parameters) and constraints (i.e. minimal required framerate). These components are then distributed and configured at runtime, in order to optimize the user experience and meet all performance constraints.

A second challenge consists of the fact that existing cyber foraging solutions only decide on offloading from one mobile device to one or more servers. However, when multiple devices are connecting to one or more servers, a global optimal solution is preferred, instead of all devices competing for the available network or compute resources. Moreover, in the case of immersive applications, devices in the same neighbourhood will often process the same or similar data. For example regarding object recognition, co-located devices will recognize the same objects, and could benefit from one another. By sharing offloaded components between multiple devices, collaborative immersive application scenarios become possible.

II. Use case: parallel tracking and mapping

With the breakthrough of head-mounted displays, a whole new type of immersive applications becomes possible. In a museum, in addition to listening to an explanation through headphones, visitors can be guided using visual annotations.

Ghent University - iMinds, Department of Information Technology.

Scenes can come to life right before their eyes immersing them in ancient history. An immersive shopping application can guide customers through the mall, annotating all products on their shopping list. Children can play with virtual toy cars or spaceships racing on a table top, colliding with real obstacles.

The above mentioned applications require similar functionality:

- The position of the glasses with respect to the world has to be known, in order to correctly position an overlay.
- There is a need for a correct model of the world, to let the overlay "blend in" without becoming too intrusive.
- Many applications require reliable object recognition, to identify objects that need to be annotated.

When multiple users run such applications in the same environment, they can clearly benefit from collaboration as they will require a model of the same environment and likely recognize the same objects.

Current state of the art algorithms providing these features rely on visual information. The Parallel Tracking and Mapping (PTAM) algorithm by Klein et al. [9] tracks the position of a camera with respect to the world based on detected visual features. In parallel with the tracking, a model of the world's visual features is constructed in a mapping phase. Castle et al. [10] combined this approach with visual object recognition, allowing to recognize objects and localize them in the map. However due to the complexity of these algorithms, current high end mobile devices still fall short to execute this in a timely manner, especially when scaling to large environments and many objects to be recognized.

III. A component-based cyber foraging approach

In order to run resource-intensive immersive applications on a mobile device, a first solution would be to run the application on a remote server and use the device as a thin client. This however requires the device to continuously send camera frames to the server, and receive rendered frames as result, which would require considerable bandwidth, and would not scale to multiple devices in the same wireless network. Therefore we adopt a component-based approach, splitting the application up into a number of loosely coupled software components that each can be offloaded and/or configured in order to optimize the user experience.

We identified different components for our use case augmented reality application as shown in Figure 1. In addition to a component for fetching video frames from the camera hardware (VideoSource) and for rendering an overlay on the screen (Renderer), there are three components for each main functionality. The Tracker component processes the camera frames and estimates the camera position with respect to the world based on a number of visual feature points. The more feature points used for tracking, the more stable the tracking becomes and the more robust to sudden camera movements. In parallel, the Mapper creates a model of the world by identifying new feature points and estimating their position in the world, which can then be used for tracking. The ObjectRecognizer tries to recognize known objects in the world and notifies the Renderer of their 3D position in the world when found.

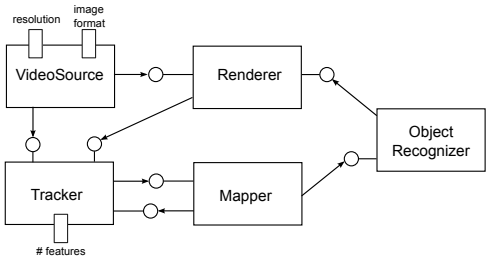


Fig. 1. Identifying loosely coupled software components in an example immersive application.

When executing such an augmented reality application, optimizing generic metrics such as energy or execution time is not a silver bullet. More important is to optimize some application specific metrics. For example, to achieve smooth camera tracking, the Tracker should be able to process 15 to 20 frames per second. On the other hand, a higher resolution of the camera frames or a higher number of tracked feature points will also enhance the user experience. Therefore, our framework enables developers to specify application performance constraints, as well as configuration parameters that influence the end user quality. To meet the imposed constraints the framework can not only offload application components, but can also configure their quality parameters.

In our use case we defined three configurable parameters. Increasing the camera resolution or the number of features used to track the camera position increase the quality, at the cost of more required processing time. The images can also be fetched either as raw image data or a compressed JPEG format. A compressed image format requires extra processing time for encoding and decoding the frames, but can lower the required bandwidth when offloading.

By adopting a component-based approach, collaborative scenarios can easily be constructed by sharing application components. In the parallel tracking and mapping use case for example, the Mapper component can be shared between multiple mobile devices in the same physical environment. This way, all devices receive the same world model to track the camera position. Because the model is updated and refined using camera images from multiple devices, the model will also be much more accurate than one device could create on its own.

IV. An autonomic cloudlet management platform

In order to benefit from the effort of developing applications as a set of loosely coupled components, a platform is required that is able to manage and configure components at runtime, as well as to distribute them among available resources in the near vicinity, called the "cloudlet". We propose a hierarchical architecture as depicted on Figure 2 that operates on three management levels: the execution environment, the node and the cloudlet.

A. Execution Environment

An execution environment is to be thought of as a container into which components can be deployed. When a component is

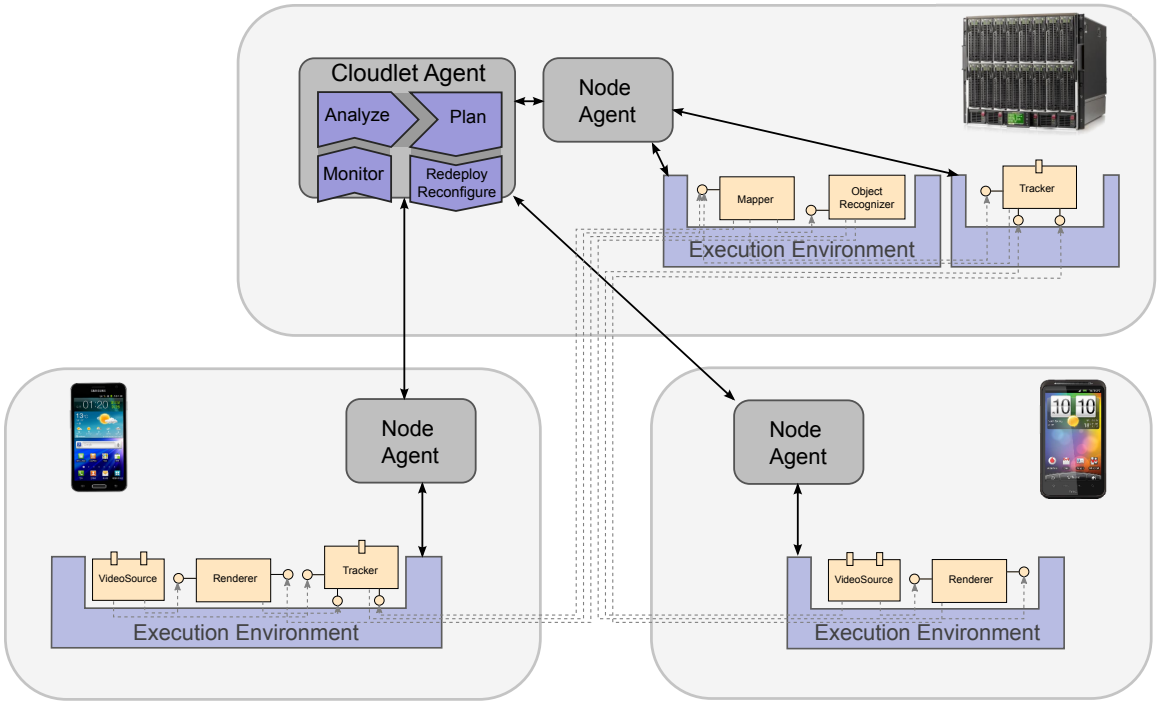


Fig. 2. Overview of the cloudlet platform. Execution environments are containers for application components. All communication between components passes through the execution environments enabling transparent monitoring and offloading. The Node Agents aggregate monitoring information for each device. The cloudlet agent analyzes the monitoring information and decides whether the deployment or configuration has to be adapted.

deployed, the execution environment manages the component life cycle and resolves the dependencies to other components. Each component can expose configuration parameters that can be adapted by the execution environment at runtime. The execution environment also enables monitoring of method calls between components, capturing the size of the arguments, return value and the execution time. This monitoring information is then sent to the node level, where the information is aggregated and used as input for decision taking on the cloudlet level.

B. Node

Each device runs a node agent, that manages the device as a whole. The node agent aggregates monitoring information from all the execution environments running on the device, and sends these to the cloudlet agent. In addition to the information about the execution environments, the node agent also provides device specific information, such as the device capabilities (i.e. processor speed, number of processor cores, etc.).

C. Cloudlet

Among all devices in the network, one is chosen to host the cloudlet agent. The cloudlet agent receives monitoring information about all the nodes, and has a global overview of all execution environments, their components and how these components interact. This information is then analyzed and used as input for a global optimization planning algorithm, that calculates both the deployment and the configuration for all components [11]. This solution is then enforced and new monitoring information is used again as feedback. In this way we realize an autonomic feedback loop that on the one hand is able to adapt to changing context, but on the other hand strives to a stable deployment, as component migration is also costly.

The device that runs the cloudlet agent can be predefined (i.e. a fixed server available in the network), or can be determined at runtime as part of the discovery process. When first initialized, each device runs both a node and cloudlet agent. When another device running a cloudlet agent is discovered, the most suitable of the two devices (i.e. the one with the most CPU power) is chosen as the host for the cloudlet agent, and the other one joins as a node. This way an ad hoc cloudlet can be constructed, where the strongest device runs the cloudlet agent performing global optimization for all others.

The cloudlet agent also identifies collaborative components, that can be shared among multiple users. By offloading collaborative components to the most resourceful node available in the network and redirecting calls of all users to this node, users not only save computational resources, but also gain information from the input of others. As depicted on Figure 2, in our use case example the Mapper and ObjectRecognizer components are shared between two devices. This way, both users receive updates to the map provided by one another, and objects have to be recognized only once in the scene in order to annotate them on each device.

V. Dynamic Configuration and Deployment

To illustrate the effectiveness of dynamic configuration and distribution of application components, we have built a prototype framework. Our implementation builds on OSGi, a standard for creating modular applications in Java, and which

we also used in previous cyber foraging implementations [8]. In order to allow easy application development targetting our platform, we use a programming model based on annotations. By adding annotations to the source code, the developer can define software components, their dependencies and their configuration parameters that are to be optimized, or define application specific constraints. Listing 1 shows the annotated source code of the Tracker class.

The Tracker component is defined, with a dependency to the Mapper component, and the number of features to track as a configurable parameter. To impose a minimum number of frames processed per second, the Tracker has to process a frame within 60 ms. These annotations are processed at build time, from which OSGi components are generated [8], as well as a number of XML descriptors defining the configurable properties and constraints which are used by the framework at runtime.

```
package ptam.tracker;

@Component
public class Tracker implements TrackerService,
    VideoListener {

  @Property(values=200,500,1000,1500,2000)
  private int featurePoints = 1000;

  @Reference(policy=dynamic)
  private MapperService mapper;

  @Constraint(maxTime=60)
  public void processFrame(byte[] data) {
    // process frame
    ...
  }
  ...
}
```

Listing 1. Example annotated Java source file defining the Tracker component, a configurable property (i.e. the number of tracked feature points) and a timing constraint (i.e. processing a frame should take less than 60 ms).

Our cloudlet agent implements an autonomic feedback loop, that periodically gathers all monitoring information from all nodes and execution environments. From this information a new deployment and configuration is calculated. In order to predict the change in behavior when changing configuration parameters, the cloudlet agent is bootstrapped with monitoring information gathered in an offline profiling stage.

Suppose N migratable components, M available devices and K configuration parameters with each k_i possible parameter values, the total number of possible solutions becomes $M^N \times k_1 \times ... \times k_K$. As the solution space rapidly grows with the number of devices, components and parameters, we propose a greedy search heuristic, which calculates close-to-optimal solutions in general cases, and has been shown to yield the optimal solution in our specific use case [11].

As our framework is implemented in Java, it runs on both Android as on regular desktop or server machines. The components of the use case application are built as OSGi modules in Java, and use native C libararies for the complex computer vision routines. The native libraries are included compiled for both ARM as x86 architectures in order to have cross-platform components.

The developer provides three parameters to configure: the input size of the camera frames (640x480 or 320x240), the format of the camera frames (raw grayscale or JPEG) and the number of feature points to track in the frame (2000, 1500, 1000, 500 or 200). By limiting the time required to process a frame for tracking, a constraint is imposed to assure a tracker framerate between 15 and 20 frames per second. Maximizing the quality is reflected as maximizing both the camera resolution and the number of tracked feature points.

We evaluate the framework on a Samsung Galaxy S2 Android device, equipped with a dual core 1.2 GHz processor and a laptop with an Intel Core 2 Duo clocked at 2.26GHz. The device and laptop are connected using a virtual network over USB. This enables us to accurately set the available bandwidth on the link, in order to show how different configurations and deployments are chosen in different scenarios.

Every 5 seconds the monitoring information is sent from the execution environment to the cloudlet agent, which decides whether an action is needed. From this monitoring information we calculate the number of processed frames per second, which is plotted in Figure 3. The experiment starts with a bandwidth of 100 Mbps, which allows the device to send the raw camera frames with a resolution of 640x480 over the network, and offload the Tracker, Mapping and Object Recognition to the laptop. After two minutes the bandwidth is lowered to 20 Mbps, resulting in a drop to 5 FPS. The framework thus adapts the configuration of the VideoSource to change to the MJPEG format. Now extra CPU cycles are used for compressing and decompressing the video frames to JPEG, which lowers the required bandwidth. The tracking is still executed on the laptop, as this allows to keep the number of tracked feature points (and thus the quality) high. When the bandwidth is even further reduced to 5 Mbps, the framerate drops again to 5 FPS, and the Tracker component is moved back to the smartphone. Because of the limited processing power on the smartphone, only 500 points can be tracked now, lowering the tracking quality. Figure 3 shows that it takes between 10 (first adaptation) and 20 (second adaptation) seconds to fully recover the framerate. This is due to the fact that the monitoring information is only sent each 5 seconds, and thus only after 5 seconds the deployment and configuration are re-evaluated and the adaptation is performed. During the second adaptation, the longer migration time is due to the Tracker component that has to be migrated from the server to the client, and also its state has to be sent over the network, while at the time of migration the network bandwidth is scarce. To further reduce the adaptation time one could for example notify the cloudlet agent much earlier when a performance drop is experienced, instead of waiting during the full 5 second period. To reduce the migration time one could periodically synchronize the state of offloaded components with the clients, such that when the component is migrated back only an incremental state change has to be transferred.

VI. Collaborative immersive applications

As multiple users in the same environment typically look at the same scene, track the same environment and need to recognize the same objects, it makes sense to share com-

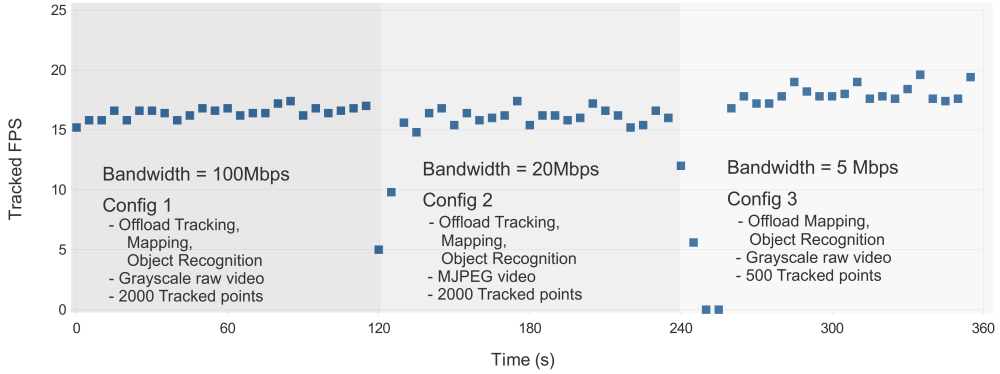


Fig. 3. As the bandwidth decreases the configuration quality is lowered in order to keep the achieved framerate between 15 and 20 FPS. The configuration is changed within 10 to 20 seconds.

putation results with eachother. By adopting the component-based application offloading model, such collaborations can be easily implemented by sharing components between different devices. When a component is marked as shared by the developer, the framework can transparantly redirect calls from different devices to the same instance of this component.

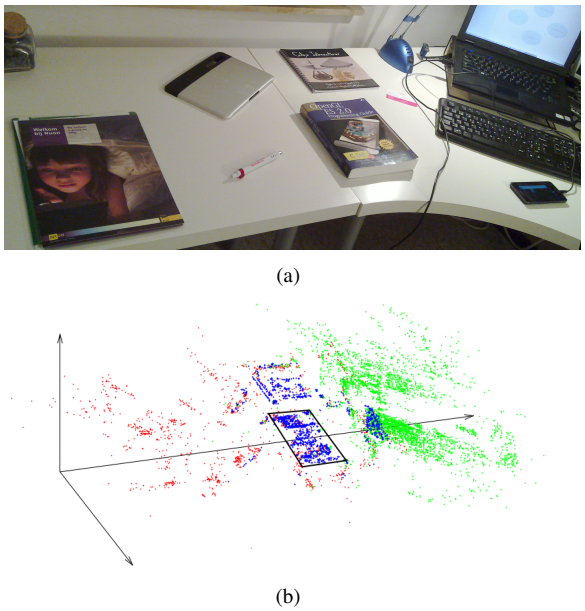
An example application for our use case is shown in Figure 4, where two devices share the same Mapper and Object Recognizer which are offloaded to the laptop. This way the map of the environment is generated twice as fast (as both devices send updates to the Mapper), and processing is only done once (instead of for each device).

Using multiple devices to add feature points to the map also leads to a more complete map of the environment. Each device now has a complete overview of the environment, whereas the map is limited to the device's own viewpoint in the non-collaborative case. In Figure 5 a map is shown from a desk environment generated with 3 devices. As each device looks to the desk from other viewpoints, they all map different (overlapping) parts of the desk.

VII. The Future of the Cloudlet

Due to the real-time constraints and the resource-intensive tasks of immersive applications, these represent very important scenarios for leveraging the cloudlet. However, a thin client approach on cyber foraging is not sufficient due to the huge bandwidth requirements, and offloading at a more fine grained level is necessary. We have proposed a component-based platform for cyber foraging, not only deciding on component deployment, but also taking into account application specific metrics and constraints provided by the developer. We have shown that our platform is able to optimize application quality and to adapt to changing network conditions using an implemented immersive application. Due to the component-based approach, our platform is able to share components between multiple devices, which opens doors to a whole new type of collaborative immersive applications.

Fig. 4. By sharing components multiple devices can track and expand the same map and recognize the same objects.



(a)

(b)

Fig. 5. The scene of the desk in (a) is explored by 3 devices resulting in the mapped features points and recognized object in (b). By sharing the detected feature points of all devices, more feature points from multiple viewpoints are added to the map, leading to robuster tracking and faster map expansion.

VIII. Related Work in Cyber Foraging

Cyber foraging has been a research topic for over a decade [12]. Early systems such as Spectra [13] offer an API to let programmers identify methods that should possibly be

TABLE I
OVERVIEW OF CYBER FORAGING SYSTEMS AND THEIR CHARACTERISTICS, BASED ON [12]

	Objective	Granularity	Decision Algorithm	Programming Model
Spectra [13]	Execution time, energy and fidelity	Method	Greedy heuristic	Use Spectra API
Chroma [14]	Execution time and fidelity	Method	Tactics-based selection	Define tactics
MAUI [5]	Energy	Method	Integer Linear Programming	Code annotations
Scavenger [7]	Execution time	Method	History-based profile	Code annotations
AIOLOS [8]	Execution time	OSGi component	History-based profile	Use OSGi or code annotations
Odessa [15]	Makespan and throughput	Sprout component	Incremental greedy algorithm	Use Sprout framework
CloneCloud [6]	Execution time or energy	Thread	Integer Linear Programming	None
COMET [16]	Execution time and energy	Thread	Threshold-based scheduler	None

executed remotely. At runtime the system selects the best option using a greedy heuristic optimizing an utility function taking into account execution time, energy usage and fidelity. Chroma [14] follows a similar approach, but uses a tactics-based decision engine optimizing a user-specific utility function.

Later systems often use a bytecode format running in an application VM in order to enable more transparent code offloading. MAUI [5] offloads methods in the Microsoft .Net framework aiming to optimize the energy usage. An ILP solver is used to decide whether or not offload a method call. Scavenger [7] offloads Python methods, minimizing the execution time depending on the method call arguments using history based profiles. Both systems require code annotations from the application developer to identify offloadable methods.

AIOLOS [8] is a cyber foraging framework on Android that takes a component-based approach, replicating OSGi components on the remote server. At runtime method calls are forwared either to the local or remote component instance using a profile built from monitoring information similar to Scavenger. Programmers can develop their applications as a number of OSGi components, or use code annotations to automate this process. Odessa [15] uses the Sprout component framework, employing an incremental greedy stragegy trying to exploit parallellism to optimize the throughput and makespan.

CloneCloud [6] operates at a lower level, migrating at thread granularity in the Dalvik VM. The CloneCloud partitioner automatically identifies offloadable parts of the application in an offline stage, using static and dynamic code analysis and without the need for programmer's annotations. At runtime the partition minimizing the execution time is selected using an ILP solver. COMET [16] also offloads threads in the Dalvik VM, but instead of rewriting the software offline, an online approach is presented using distributed shared memory (DSM). This allows to migrate threads at runtime using a threshold based scheduler, at the cost of more communication overhead for keeping the DSM synchronized.

Our framework uses an application-specific utility function similarly as [14] in order to optimize the application quality. We adopt a component-based approach as in [8], which gives a good granularity to offload, while keeping the monitoring overhead low. In contrast to existing systems, our framework not only adapts the deployment of parts of the application but also autonomically adapts configuration parameters of the components in order to adhere to constraints given by the developer and the device context. A heuristic search algorithm allows fast optimization of both the deployment and configuration for all connected devices in the vicinity. Second, the framework takes into account collaborative components, that can be shared among multiple users. This way, users can benefit from computations already done by others, thus reducing the overall computational load.

ACKNOWLEDGMENT

Tim Verbelen is funded by Ph.D grant of the Fund for Scientific Research, Flanders (FWO-V).

Part of this work was funded by the Ghent University GOA-grant "Autonomic networked multimedia systems".

REFERENCES

[1] T. Langlotz, D. Wagner, A. Mulloni, and D. Schmalstieg, "Online creation of panoramic augmented reality annotations on mobile phones," *IEEE Pervasive Computing*, vol. 11, no. 2, pp. 56 –63, feb. 2012.

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

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14 –23, oct.-dec. 2009.

[5] 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 *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62.

[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314.

[7] M. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, 29 2010-april 2 2010, pp. 217 –226.

[8] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Aiolos : middleware for improving mobile application performance through cyber foraging," *Journal of Systems and Software*, vol. 85, no. 11, pp. 2629–2639, 2012.

[9] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 6th International Symposium on Mixed and Augmented Reality*, ser. ISMAR '07, 2007, pp. 1–10.

[10] R. Castle and D. Murray, "Keyframe-based recognition and localization during video-rate parallel tracking and mapping," *Image and Vision Computing*, vol. 29, no. 8, pp. 524–532, 2011.

[11] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Adaptive application configuration and distribution in mobile cloudlet middleware," *Mobile Wireless Middleware, Operating Systems, and Applications*, pp. 178–192, 2012.

[12] J. Flinn, *Cyber Foraging: Bridging Mobile and Cloud Computing*, ser. Synthesis Lectures on Mobile and Pervasive Computing. Morgan & Claypool Publishers, 2012.

[13] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ser. ICDCS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 217–.

[14] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 273–286.

[15] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 43–56.

[16] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: code offload by migrating execution transparently," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, ser. OSDI'12. USENIX Association, 2012, pp. 93–106.