Technical Report

A Framework for Offloading Real-Time Applications in a Distributed Environment

Cláudio Maia
Guilherme Silva
Luis Lino Ferreira
Luís Miguel Pinho
Luís Nogueira
Joel Gonçalves

HURRAY-TR-111204
Version:
Date: 12-02-2011

A Framework for Offloading Real-Time Applications in a Distributed Environment

Cláudio Maia, Guilherme Silva, Luis Lino Ferreira, Luís Miguel Pinho, Luís Nogueira, Joel Gonçalves

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

This work focuses on highly dynamic distributed systems with Quality of Service (QoS) constraints (most importantly real-time constraints). To that purpose, real-time applications may benefit from code offloading techniques, so that parts of the application can be offloaded and executed, as services, by neighbour nodes, which are willing to cooperate in such computations. These applications explicitly state their QoS requirements, which are translated into resource requirements, in order to evaluate the feasibility of accepting other applications in the system.

A Framework for Offloading Real-Time Applications in a Distributed Environment

Cláudio Maia, Guilherme Silva, Luis Lino Ferreira, Luís Miguel Pinho, Luís Nogueira, Joel Gonçalves
CISTER Research Centre
School of Engineering of the Polytechnic Institute of Porto
Porto, Portugal
{crrm, grss, llf, lmp, lmn, vjmg}@isep.ipp.pt

Abstract— This work focuses on highly dynamic distributed systems with Quality of Service (QoS) constraints (most importantly real-time constraints). To that purpose, real-time applications may benefit from code offloading techniques, so that parts of the application can be offloaded and executed, as services, by neighbour nodes, which are willing to cooperate in such computations. These applications explicitly state their QoS requirements, which are translated into resource requirements, in order to evaluate the feasibility of accepting other applications in the system.

Keywords- Code Offloading, Cooperative Distributed Systems, Real-Time

I. INTRODUCTION

In the context of cooperative distributed systems, a full-fledged framework was developed with the objective of integrating code offloading techniques [1], on top of a middleware framework that provides QoS [2], and real-time guarantees to the applications [3].

The application presented in this paper is a proof-of-concept of the offloading algorithms proposed in [1], which rely on the services provided by a code mobility library, named MobFr [4]. Additionally, the formation of a coalition to run the application is based on the CooperatES framework [2]. This framework is capable of finding a proper coalition to run a distributed application, based on the resource requirements (CPU, memory, display, etc.) of the services constituting the application. Furthermore, a real-time scheduler implemented in the Linux kernel of the Android OS – the Capacity Sharing and Stealing (CSS), proposed in [3], supports the above framework.

CSS integrates and extends recent advances in dynamic deadline scheduling with resource reservation. CSS proposes the coexistence of the traditional isolated servers with a novel non-isolated type of servers, combining an efficient reclamation of residual capacities with a controlled isolation loss. It is proven that CSS achieves a better system's performance when compared to other available server-based solutions and has a lower overhead.

II. OFFLOADING FRAMEWORK

In order to demonstrate the framework capabilities, we are using a physics simulation application that replicates the fall of different kinds of geometrical figures, and its bouncing properties, when hitting other objects or the bounds of the physics world. Figure 1 depicts a screenshot of the application's execution.

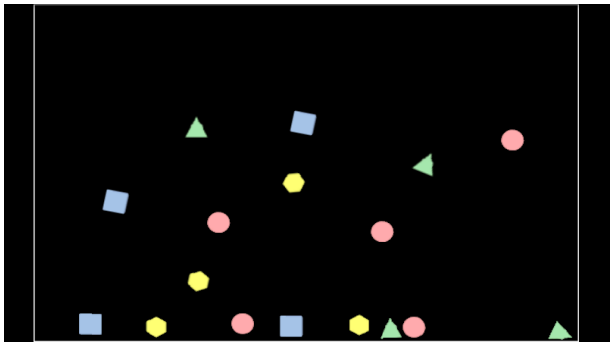

Figure 1. Physics simulation application

An object with a random shape appears on screen every time the user presses the display area; the objects' trajectories are drawn with a frequency of 30 frames per second (the update period – *Tp*), thus requiring the execution of the physics engine with the same frequency.

Importantly, to our application, is that due to the collision detection and path calculation algorithms used, the application's execution time increases with the number of objects, as shown in [1].

CPU resources assigned to this application are guaranteed by the CSS scheduler, which guarantees that the physics simulation task is able to execute for a time equal to *tMaxCap* on every *Tp* interval length.

The main objective of the offloading algorithm is to dynamically adapt to the varying execution times by offloading computations to surrogate nodes in a timely fashion. By timely it is meant that the user should not notice any disruption on the application's behaviour, meaning that local or remote execution (including communications) should be completed prior to *Tp*.

To that purpose, the offloading algorithm tries to predict the forthcoming physics simulation execution times, based on a linear regression calculated from past execution times. When a device needs to offload code, the physics engine distributes the

physics world by surrogate nodes. Each surrogate node is then responsible for calculating a set of objects and returning its results to the main node, where the simulation results are displayed.

The Android OS was chosen to demonstrate the feasibility the framework. The architecture, depicted in Figure 2, is composed by two libraries, Offloading Library and MobFr, placed at the Applications layer, and the CooperatES framework implemented on top of the Linux kernel and Android Runtime.

Concerning the Offloading library, it provides support to the offloading operations required by an application. These operations involve handling the exchange of data among the devices that are part of the offloading coalition, which is constituted by one main node and several surrogate nodes, as well as to monitor application execution, so that application's execution times are read, and then used to perform calculations.

Underneath the Offloading library is MobFr, a service-based QoS-enabled library capable of handling code mobility. Among the provided features, MobFr is designed to: (i) detect neighbour devices, by using the System Manager component from the CooperatES framework; (ii) determine the best candidate to run the offloaded code, according to the QoS requirements of the application and the available resources on the surrogate nodes. This can be achieved by using the CooperatES framework components running within the Android Runtime; (iii) migrate the code and initial state; (iv) remotely control the code execution; and finally, (v) handle the transfer of data between nodes.

III. THE COOPERATES FRAMEWORK

In the core of the Android platform there is the CooperatES framework. This framework is responsible for providing QoS guarantees to the upper layers' applications by evaluating the resource requirements of each particular application, and evaluating if the current node has enough resources to provide to that particular application. If this is the case, the request is handled locally; otherwise the framework is responsible for executing the application in a coalition of neighbour nodes.

Neighbour nodes may cooperate either because they cannot deal alone with the resource allocation demands imposed by users and applications, or because they can reduce the associated cost of execution by working together.

Application requests are handled by the QoS Provider, which is composed by the Local Provider and Coalition Organiser components. The Local Provider is responsible for determining if a local execution of the new service is possible within the user's accepted QoS range, e.g. our physics simulation application has been designed to support different frame rates {30, 20, 10}.

Rather than reserving local resources directly, the Local Provider contacts the Resource Managers to grant the specific resource amounts requested by the service. If the resource demand imposed by the user's QoS preferences cannot be locally satisfied, the Coalition Organiser starts the coalition formation process. The Coalition Organiser interacts directly with the System Manager.

The System Manager is responsible for maintaining the overall system configuration, detecting nodes entering and leaving the network, and managing the coalition's operations and dissolution.

Finally, the framework relies on a modified version of the Linux kernel that incorporates CSS, in order to provide the real-time guarantees required by the applications.

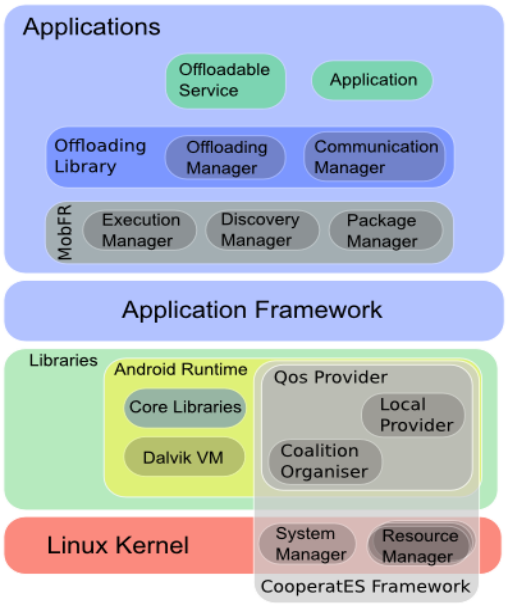

Figure 2. System Architecture

ACKNOWLEDGMENT

This work was supported by the CISTER Research Unit (608FCT) and CooperatES project (PTDC/EIA/ 71624/2006), both funded by FEDER funds through COMPETE (POFC-Operational Programme 'Thematic Factors of Competitiveness) and by National Funds (PT), through the FCT-Portuguese Foundation for Science and Technology; and the ArtistDesign NoE ref. ICT-FP7-214373.

REFERENCES

[1] L. Ferreira, G. Silva, L. Miguel Pinho. Service Offloading in Adaptive Real-Time Systems. In proceedings of the 6th IEEE International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE), Toulouse, France, 2011.

[2] L. Nogueira, L. Miguel Pinho. Time-bounded Distributed QoS-aware Service Configuration in Heterogeneous Cooperative Environments. Journal of Parallel and Distributed Computing, 69(6):491–507, June 2009.

[3] L. Nogueira, L. Miguel Pinho. A Capacity Sharing and Stealing Strategy for Open Real-Time Systems. Journal of Systems Architecture, Volume 56, Issues 4-6, April-June 2010.

[4] J. Gonçalves, L. Ferreira, L. Miguel Pinho, G. Silva, Handling Mobility on a QoS-Aware Service-based Framework for Mobile Systems. In proceedings of the Intl. Conf. on Embedded and Ubiquitous Computing (EUC 2010), pages 97-104, Hong Kong, China, 2010.