Service Offloading in Adaptive Real-Time Systems
Luis Lino Ferreira, Guilherme Silva, Luis Miguel Pinho
CISTER Research Centre, ISEP, Portugal
www.cister.isep.ipp.pt

Presented by: Vincent Nelis

SOCNE 2011
5, September, 2011
Example (Asteroid storm)
Example (Asteroid storm)
Example (Asteroid storm)
Example (Asteroid storm)
Example (Asteroid storm)
Example - Requirements

Soft real-time requirement: **25 FPS !!!**
Example - Requirements

Soft real-time requirement: **25 FPS !!!**

Meaning: computations have to be done within $\frac{1}{25} = 40$ milliseconds
Example - Execution time

![Game screenshot with execution time](image)

**Example** - Execution time

<table>
<thead>
<tr>
<th>Number of Physics Objects</th>
<th>Physics Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>30</td>
<td>80</td>
</tr>
<tr>
<td>45</td>
<td>120</td>
</tr>
<tr>
<td>60</td>
<td>160</td>
</tr>
<tr>
<td>75</td>
<td>200</td>
</tr>
<tr>
<td>90</td>
<td>240</td>
</tr>
<tr>
<td>105</td>
<td>280</td>
</tr>
<tr>
<td>120</td>
<td>320</td>
</tr>
</tbody>
</table>

*Graph showing the relationship between the number of physics objects and execution time.*

**40 ms**
Example - Solution, main idea

- The difficulty of the game, i.e., the number of objects coming on the screen, generally grows linearly.
The difficulty of the game, i.e., the number of objects coming on the screen, generally *grows* linearly.

Thus, it is possible to **predict (estimate)** the time-instant at which the computation will take too much time.
The difficulty of the game, i.e., the number of objects coming on the screen, generally grows linearly.

Thus, it is possible to **predict (estimate) the time-instant** at which the computation will take too much time.

**Solution**: find some **reinforcing devices** before reaching this instant.
Example - Solution (offloading)
Example - Solution (offloading)

- Predict
- Current time
- Future instant

TIME
Example - Solution (offloading)
Example - Solution (offloading)

Communication delay...

Help!

Communication delay...

Communication delay...
Problem Characterization

We specifically address applications which:

○ Have soft real-time requirements

○ The real-time requirements of the application change over time

○ Application execution exhibits a periodic behavior with:
  - A period of $T_p$
  - A variable execution time of $t_{core}$
Problem Characterization

- Such kind of behaviour can be found applications like:
  - **Multimedia**
    - Real-time image analysis
    - Periodic execution
    - Image analysis times vary according to image complexity
  - **Real-time Games**
    - Most of these games are based on game engine which runs periodically
    - The game engine periodically runs modules for physics, rendering, artificial intelligence
    - Depending on the number of objects being calculated the execution time of these modules varies
Example

- The vertical axis represents the time required to run the module core services ($t_{core}$) on a mobile phone.
- The horizontal axis represents the number of objects being calculated on a physics simulation.
Considerations about the example

- Such an application would only be able to calculate the trajectories of around 30 objects and maintain the same frame rate.
  - One solution would be to downgrade the quality of the calculations.

- The solution we are proposing to offload some of those calculations to neighbor nodes, at the same time guaranteeing the real-time requirements of the application.
The main objective is to dynamically adapt to the varying execution times

- by offloading computation to surrogate nodes in a timely way.
  - By timely we mean that the user should not notice any disruption on the application behavior.
- To that purpose, the offloading algorithm tries to predict the forthcoming core execution times, based on past execution times and execute the costly offloading procedure in advance to any overload situation.
Real-time offloading of mobile services

![Graph showing core execution time comparison between local and distributed execution]

- Local execution
- Distributed execution

- Core Execution
- Offload Core Execution
- Linear Regression

$t_p$, $t_{MaxCap}$, $t_{mob}$, $t_{th}$, $t_{xMaxCap}$

SOCNE 2011, 5/9/2011
Real-time offloading of mobile services

- $t_{\text{MaxCap}}$: the maximum capacity which can be used during a core cycle ($T_p$). This capacity is reserved by the underlying OS.
  
  Such reservation can be assured by algorithms like the Constant Bandwidth Server.

- The algorithm predicts the future evolution of the core execution time using linear regression.
Offloading decision

- If the crossing point of the linear regression line crosses the maximum capacity line in a future time, then offloading should be executed if:

\[ t_{th} = \left( \frac{t_{xMaxCap}}{T_p} \right) \times T_p \leq \left( \frac{t_{mob}}{T_p} \right) \times T_p \]

- Thus insuring that offloading is:
  - Performed only when necessary
  - That the offloading procedure can be done without disrupting the application
Offloading algorithms

Function update() { // called every $T_p$ by the application
    If isOffloading()
        If requiresNewSurrogate()
            new Thread(addAdicionalDevices())
            runOffloaded()
        else {
            if needsToStopOffloading() {
                runLocally()
            } else
                runOffloaded()
        }
    else //if is not offloaded
        If requiresNewSurrogate() {
            new Thread(addAdicionalDevices())
            runLocally()
        } else
            runLocally()
Offloading algorithms

Function requiresNewSurrogate() {
  Output result: bool – determines if a new surrogate node is required.

  ForEach(dev in devices) {
    If eq (1) is true
      If !tryRebalance() Return true
    Return False
  }
}

Thread addAdicionalDevice() {
  newDev = DiscoveryManager.getDevice()
  If (newDevice != null)
    Devices.add(newDev)
    OffloadCode(newDev)
  Else
    signalError()
}
Implementation issues

- Offloading framework
  - An Application layer framework that supports code offloading applications, which builds upon a code mobility framework – MobFr
Implementation issues

Main Device

- Offloadable Service
- Communication Manager
- QoS Manager
- Package Manager
- Execution Manager

Surrogate Device

- Offloadable Service
- Communication Manager
- QoS Manager
- Discovery Manager
- Package Manager
Implementation issues

Main Device

Application

Offloadable Service

Communication Manager

Offloading Manager

MobFR

Execution Manager

Package Manager

QoS Manager

Discovery Manager

Surrogate Device

Application

Offloadable Service

Communication Manager

MobFR

Execution Manager

QoS Manager

Discovery Manager

Package Manager
MobFR

- distributed framework for Android
- implements mechanisms to support the mobility of Android services
- interacts with the underlying OS in order to obtain the desired QoS level for the applications.
Implementation issues

Main Device

Application
- Offloadable Service
- Communication Manager
- Offloading Manager
- Execution Manager
- Package Manager

Surrogate Device

Service
- Application
- Offloadable Service
- Communication Manager
- Discovery Manager
- Package Manager

QoS Manager
QoS Manager

- administers the system resources, either locally, on a node, or in a distributed environment
- encapsulates the functionalities of high level QoS control frameworks
Implementation issues

Main Device

Application
- Offloadable Service
- Communication Manager
- Offloading Manager
- QoS Manager
- Discovery Manager
- Execution Manager
- Package Manager

Surrogate Device

Application
- Offloadable Service
- Communication Manager
- Discovery Manager
- Package Manager

Service
Discovery Manager

- designed to discover neighbor devices on a local network
- advertise the host’s resource availability
- gather information about the resource availability on neighbor devices
Implementation issues

Main Device

Application

Offloadable Service

Communication Manager

Offloading Manager

MobFR

Execution Manager

QoS Manager

Package Manager

Discovery Manager

Surrogate Device

Application

Offloadable Service

Communication Manager

MobFR

Execution Manager

QoS Manager

Discovery Manager

Package Manager
Package Manager

Install, uninstall and transfer services. This module is also responsible for the interaction with QoS Manager in order to request specific QoS levels for the service being transferred.
Implementation issues

Main Device

- Offloadable Service
- Communication Manager
- Execution Manager
- Package Manager
- QoS Manager
- Discovery Manager

Surrogate Device

- Offloadable Service
- Communication Manager
- Execution Manager
- Discovery Manager
- Package Manager
- QoS Manager
Implementation issues

- Execution Manager allows executing services on a surrogate node through the exchange of Android intents, this allowing the development of transparent applications (in relation to its distribution)
Implementation issues

Main Device

Application

Offloadable Service

Communication Manager

QoS Manager

Discovery Manager

Execution Manager

Offloading Manager

MobFR

Surrogate Device

Application

Offloadable Service

Communication Manager

QoS Manager

Execution Manager

Discovery Manager

Package Manager

MobFR
Implementation issues

Communication Manager

- takes care of communications between the main and surrogate devices. Basically, this module implements the functionalities required to send and receive data and results aggregation.
Implementation issues

Main Device

- Offloadable Service
- Communication Manager
- Execution Manager
- Package Manager
- QoS Manager
- Offloading Manager

Surrogate Device

- Offloadable Service
- Communication Manager
- Discovery Manager
- Package Manager
- Execution Manager
- QoS Manager

MobFR

SOCNE 2011, 5/9/2011
Implementation issues

- Offloading Manager:
  - takes care of the initial configuration, monitoring the core execution times
  - control the creation of new surrogates
Implementation issues

Main Device

- Offloadable Service
- Communication Manager
- Execution Manager
- Package Manager
- QoS Manager

Surrogate Device

- Offloadable Service
- Communication Manager
- Discovery Manager
- QoS Manager
- Package Manager

MobFR

Application

Offloading Manager
Implementation issues

Main Device

- Offloadable Service
- Communication Manager
- Offloading Manager

Surrogate Device

- Application
- Communication Manager
- Offloadable Service

MobFR

Offloadable Manager

- represents the application modules which can offload to other nodes.
Conclusions

- We proposed the mechanisms for the support of real-time offloadable applications.
- The offloading framework exhibits a dynamic behaviour thus adapting to changing behaviour from the application.
Thank you!
Questions?