Adaptive online deployment for resource constrained mobile smart clients

Tim Verbelen

Raf Hens, Tim Stevens, Filip De Turck, Bart Dhoedt

Ghent University - IBBT

Mobilware 2010

UNIVERSITEIT GENT

ibbt
connect.innovate.create

From dumbphone to smartphone

From mobile phone …

GPS

1973

2010

…to application platform

Still cope with mobile device constraints

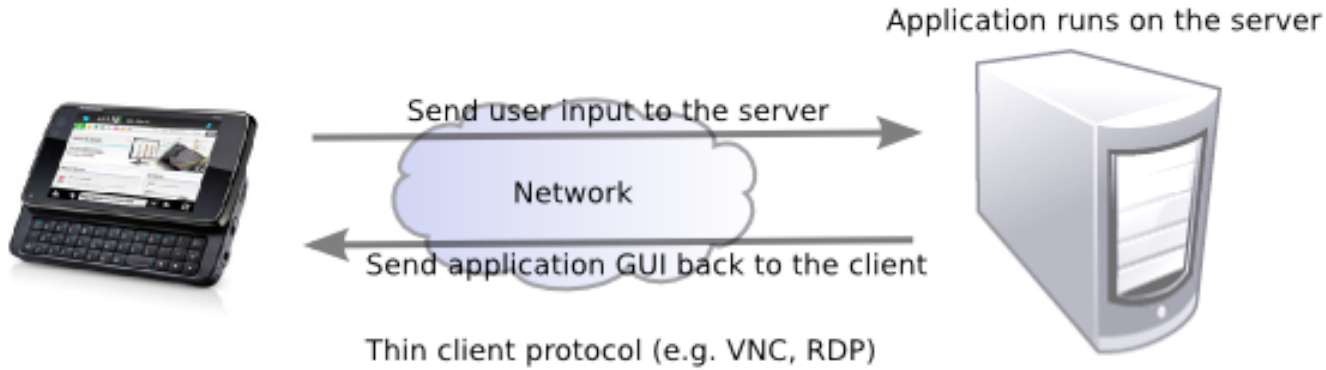
- Limited resources
  - CPU
  - Memory
  - Battery
- Changing device context
  - Network connectivity

Running advanced applications remains a challenge!

Move to the cloud

Thin client

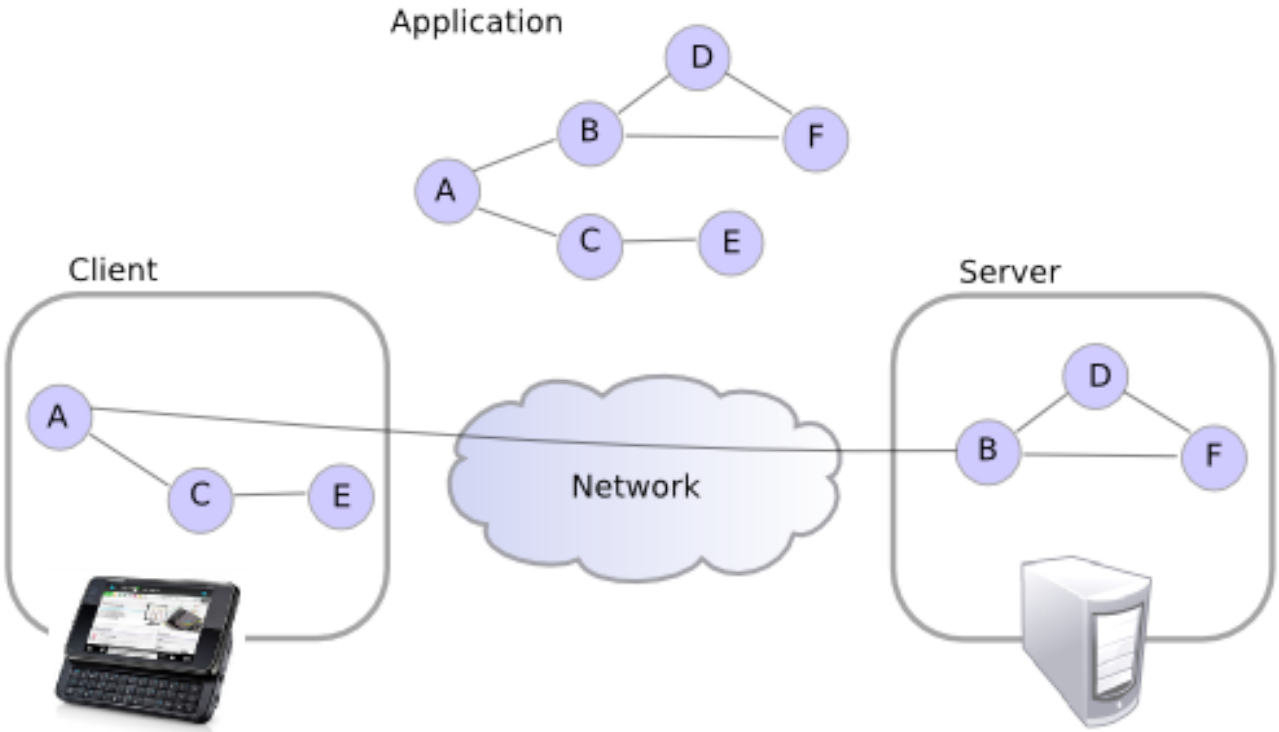
- Run the whole application on a server



- Disadvantages
  - Constant and high bandwidth needed
  - Always extra latency introduced
  - Capacity on the mobile device remains unused

Smart client

- Only offload parts of the software



Where to cut?

What is the optimal deployment?

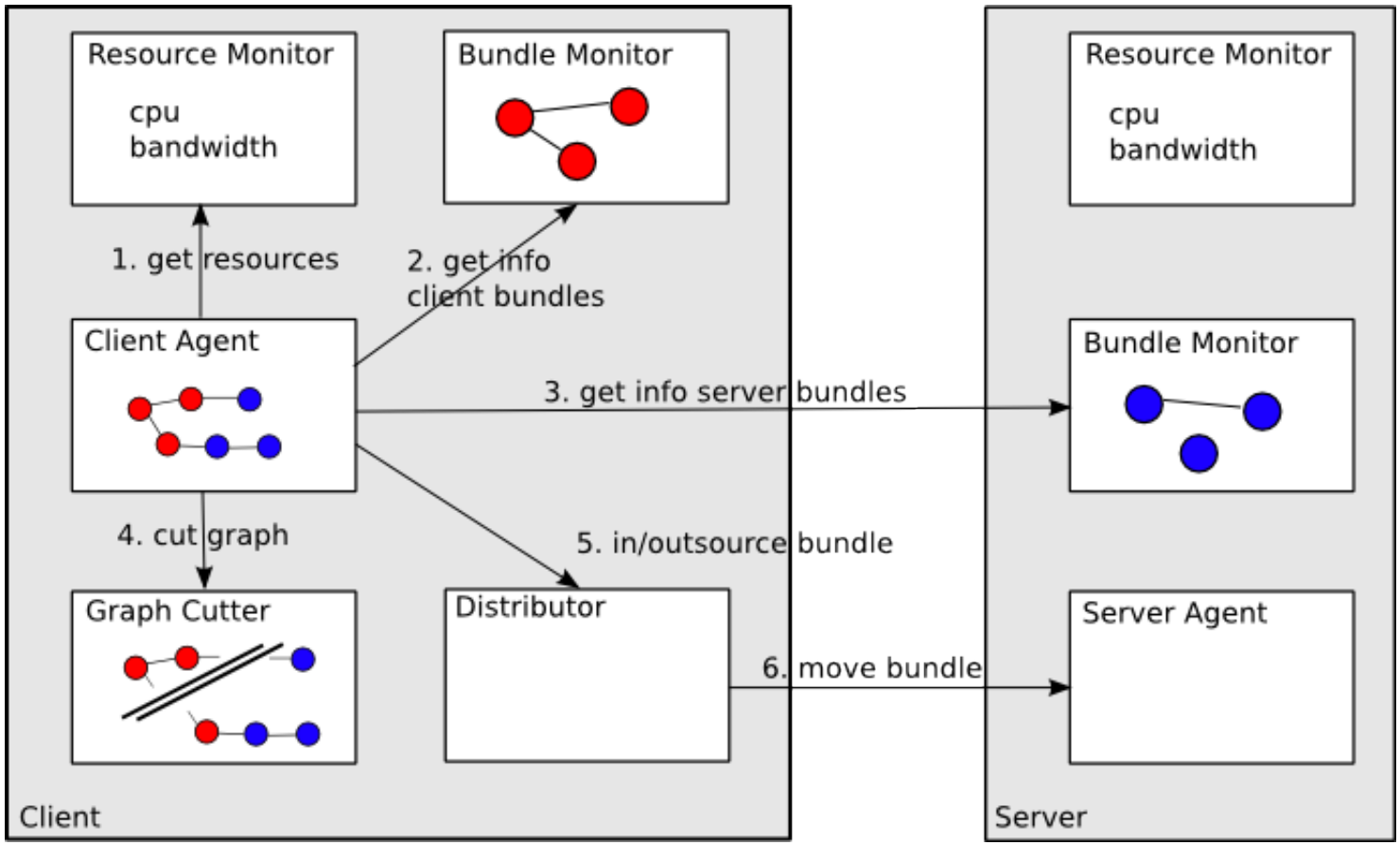
- The one that offers the end user the best user experience!
- But is this the deployment that:
  - offers best performance?
  - consumes minimal battery power?
  - offers minimal latency and maximal responsiveness?
  - minimizes the end user's data traffic bill?

It depends!

Middleware for adaptive deployment

- At runtime (re-)deploy the software
  - Migrate components from client to server and vice-versa
- Learn the structure of the software at runtime
  - Build up graph structure from monitor information
  - Node weights: resource utilization of a component
  - Edge weight: communication cost between components
- Decide which is the best cut
  - Choose cut algorithm based on context
  - In this paper: Augmented Reality use case
  - Restrict the CPU usage at the client side
  - Minimize the bandwidth between client and server

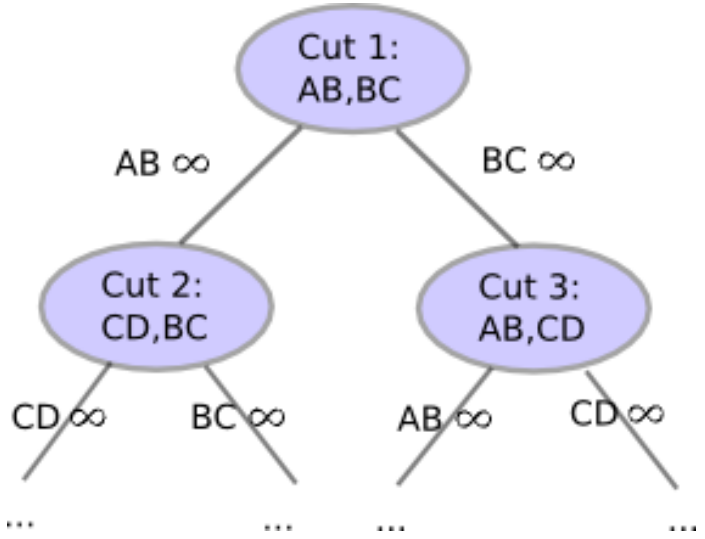
Architecture



Graph cut algorithm

- Client side weight cannot exceed threshold
- Minimize the bandwidth
- Iteratively execute Stoer & Wagner minimum cut algorithm

Example:

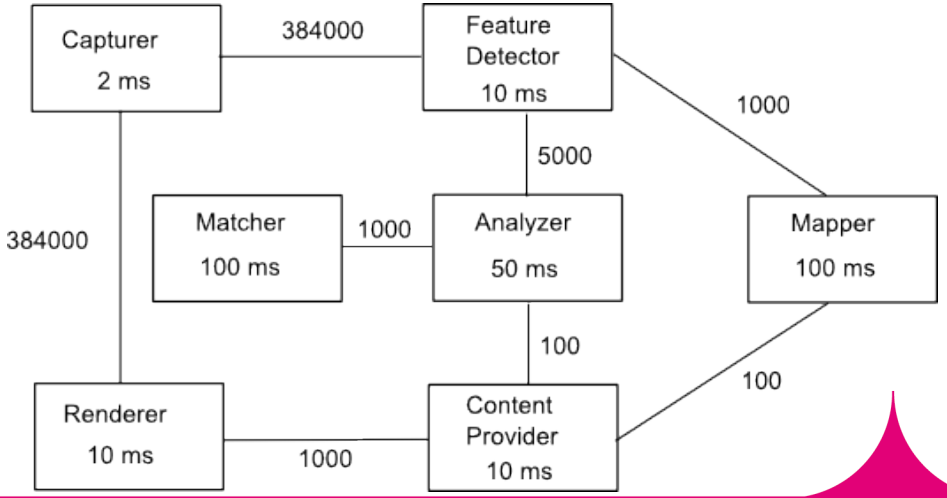


Implementation

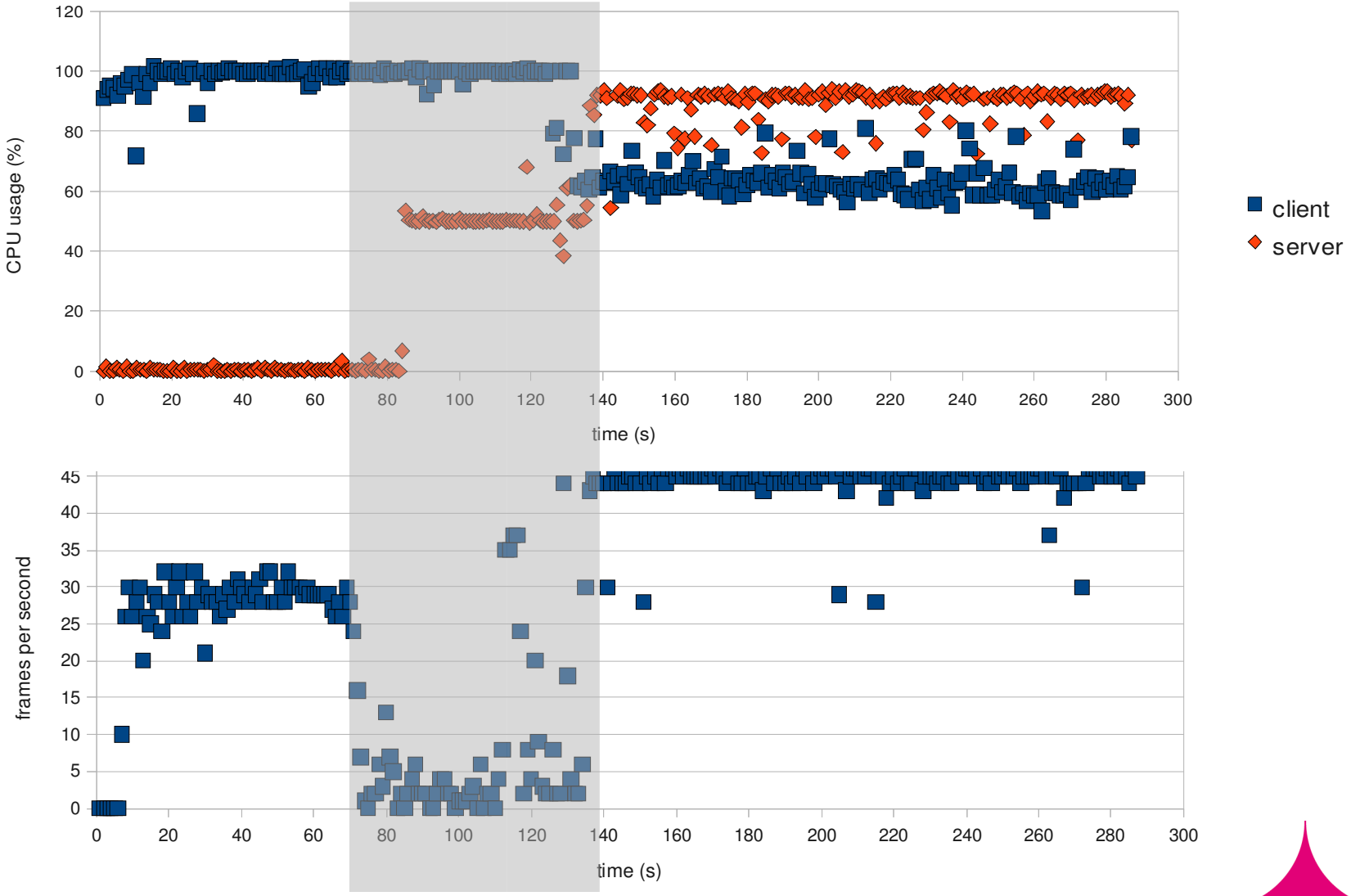
- Based on the OSGi framework
- OSGi bundles as unit of deployment
- On the fly monitoring to get weighted graph
  - System resource usage
  - CPU usage information of each OSGi bundle
  - Data exchanged between OSGi bundles
  - Low overhead (<2%)
- Bundle migration
  - Generate a local proxy that forwards calls to a remote instance (R-OSGi)
  - Send .jar to the server

Evaluation – Augmented reality use case

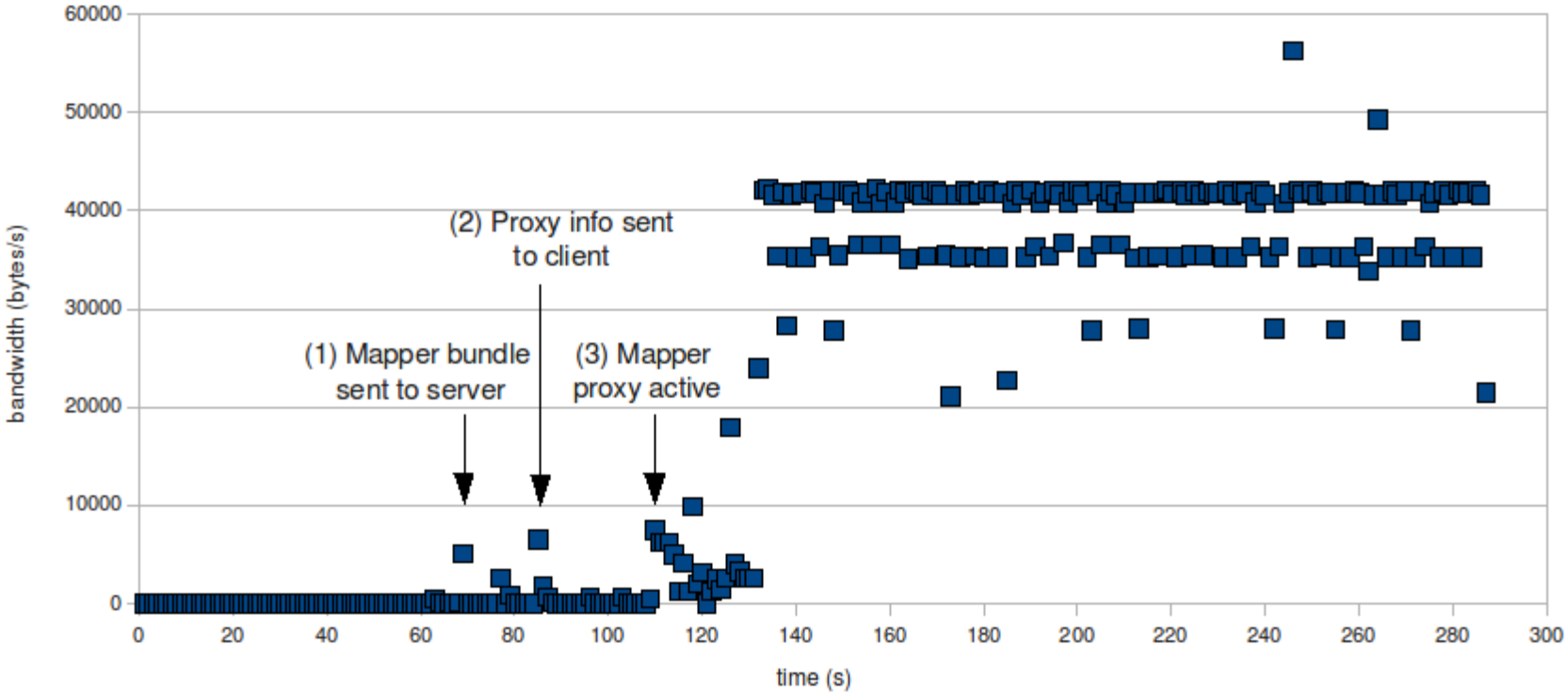
- Client: Nokia N900 with OMAP 3430 at 600 MHz
- Server: laptop with Intel Core 2 DUO at 2.26 GHz
- After 60s – start the ClientAgent, redeploy when CPU usage at the client is above threshold 80%
- To measure performance we count how many times our stub renderer component is called
- Stub AR application:



Results – CPU usage & perfomance



Results - Bandwidth



Conclusions

- Adaptive online redeployment gives opportunities for more demanding applications
- On the fly generation of proxy bundles on the mobile device is expensive → generate them in advance
- Offloading a bundle while the application is running strongly degrades application performance during some seconds → try to offload in advance

Future work

- Faster redeployment and less performance loss during migration
- Stateful migration of components
- Other policies for graph cutting and redeployment
  - Applications generating variable load
  - Different contexts (e.g. wireless connectivity)
  - Different optimization goals (e.g. energy)

Questions?