Runtime Deployment Adaptation for Resource Constrained Devices

Raf Hens, Bas Boone, Filip De Turck, Bart Dhoedt
Department of Information Technology
Ghent University - IBBT - IMEC
9050 Gent, Belgium
E-mail: Raf.Hens@intec.UGent.be

***Abstract*—This paper proposes a solution for the resource constraints of mobile devices. As an alternative for the thin client approach, a hybrid approach is presented, that is able to adapt dynamically to changes in the context in which the mobile application runs. By switching dynamically, at runtime, between local and remote execution of components of an application, the quality of experience of the end user is optimized.**
An ILP model to minimize the impact on end user experience, taking into account different context parameters, such network delay and available processing resources, is presented. Tests performed with this model have resulted in a simplified ILP model. The behaviour of this model under changing network conditions is presented.
Finally, the possibilities to incorporate this model into an earlier designed framework are discussed.

I. Introduction

The popularity of mobile devices, such as PDAs and mobile phones, their increasing networking capabilities and the growing number of applications and uses for those devices caused the thin client paradigm to regain interest. Although mobile devices have increasingly more resources, they are still inherently restricted, because of their mobility, if only because they are battery powered. Therefore the thin client approach is feasible in this context: offloading tasks to more powerful back end servers in the network, while using the mobile device only for input and output.

However, in order to reach an acceptable quality of experience, the network has to meet certain conditions. Network bandwidth has to be sufficient and network latency has to be low enough to guarantee a smooth user experience. Especially for mobile devices this is a significant problem, because they use wireless connections, which can fluctuate a lot in bandwidth and latency over time.

Thus, a pure thin client approach is highly dependent on the network conditions. Therefore we propose a hybrid solution, that can switch at runtime between local execution and offloading of components of an application, transparently towards the end user. Such an approach enables adaptation to the changes in the dynamic context in which mobile applications are executed.

This paper focuses on the question which composition of the application is best suited at a certain time, with regard to the location where the components are executed: on the hybrid thin client device or on the offloading server.

A. Related work

In [1] an architecture and design to support runtime implementation switching have been presented. The high level architecture is shown in Fig. 1. It is a component based application framework that generates a level of indirection (shaded in the figure) between the components an application is composed of, in order to choose the best suitable component at any time. By offering the switching functionality, the framework relieves application developers from the adaptation to the changing conditions.

Although it is designed to support runtime implementation switching between any component of a set of functionally equivalent components, the switching in this case is limited to a locally executing and a remotely executing component.

This paper focuses on the decision process of when to switch to another deployment of locally and remotely executing components. The focus is thus on the *Switching Mechanism*.

The concept of offloading tasks to nodes in the network has similarities to grid computing [2]. An important difference is the level of user interaction: tasks in grid computing are often computationally complex and/or data intensive processes without user interaction. A study of grid computing for interactive applications

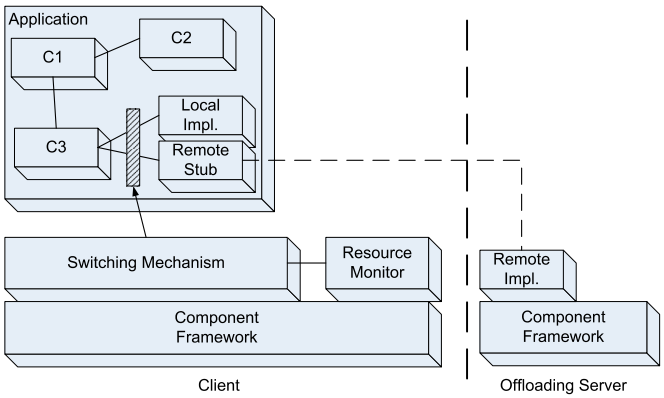

Fig. 1. Architecture of the framework

has been presented in [3]. Another important element is the mobility of the end user device and the associated dynamic context in which the applications run.

II. General ILP Model

In [4] an ILP model has been presented to find the best deployment of a set of components of a distributed system on a set of network nodes, with regard to the total delay for a call to go through the distributed system. This optimization is done at design time.

This model has been used as a basis for an ILP model that calculates the best deployment of locally and remotely executed components, given a set of conditions, such as available resources and network parameters.

The model takes network nodes and software components in consideration. Network nodes have a processing capacity and a memory capacity. Software components have a processor load and a memory usage. Network nodes can be connected through network links. A dependency between software components executing on different network nodes will introduce network traffic. The model will optimize the deployment of the components and the way the traffic is routed.

A. Parameters

Since the goal is to adapt to changing conditions at runtime, the parameters in the model should be interpreted as snapshots of the conditions at a certain time. The indication of a timestamp has been omitted for clarity, thus L_{it} is written as L_i (processor load of component i at time t).

In the model, the following parameters are used:

- I: the set of software components which are to be deployed
- R: the set of network nodes on which these software components can be deployed; this set consists of two subsets: R_c (hybrid thin clients) and R_s (offloading servers)
- N: the set of (uni-directional) network links between network nodes
- L_i, $i \in I$: the processor load of software component i. This is an indication of the processing load the component will cause on a node. It could be measured by profiling the component on a reference system.
- M_i, $i \in I$: the memory usage of software component i. Again, profiling may be used to determine this value.
- C_r, $r \in R$: the processing capacity of node r. This parameter is related to the processor load (L_i): the processor utilization for component i on node r will be L_i/C_r. This parameter could be measured by comparing the node with a reference system.
- Mem_r, $r \in R$: the memory capacity of node r. This parameter is related to the memory usage (M_i): the relative amount of memory used by component i on node r will be M_i/Mem_i.
- T_{ij}, $i, j \in I$: the amount of data that is exchanged between component i and j.
- B_l, $l \in N$: the capacity (available bandwidth) of network link l.
- δ_l, $l \in N$: an indication of the network latency of network link l.
- D_{ij}, $i, j \in I$: the latency dependency between component i and j. This is an indication of how much influence there is when network traffic between component i and j has a certain network latency.

B. Decision variables

The decision variables represent the deployment of the components and the routing of messages through the network. These variables are set by the ILP solver in order to optimize the objective function.

- S_{ir}: a binary decision variable. Its value is equal to 1 when component i is deployed on node r, and 0 otherwise.
- h_{lij}: a binary decision variable. Its value is equal to 1 if the network traffic from component i to component j is routed over network link l, and 0 otherwise.

C. Constraints

- $\sum_{r \in R} S_{ir} = 1, \forall i \in I$: this constraint ensures that each component i is deployed on exactly one node r.

- $\sum_{l \in Out(r)} h_{lij} - \sum_{l \in In(r)} h_{lij} = \begin{cases} S_{ir} - S_{jr}, & \text{if a call exists from i to j} \\ 0, & \text{otherwise} \end{cases}$, $\forall r \in R, i, j \in I$
  These constraints are the network flow constraints. They ensure that a path exists between components i and j, irrespective of where those components are deployed. $Out(r)$ and $In(r)$ represent the set of network links that start and end in r, respectively. If component i is deployed on r and j is deployed elsewhere, then the right hand side of the equation equals 1. If both components are deployed on r, then the right hand side of the equation equals 0: no network flow is required.
- $\sum_{i \in I} S_{ir} L_i < 0.9 \times C_r, \forall r \in R$: Processing capacity constraint: the total load caused by the deployment of components on node r can not exceed the processing capacity of the node. Note that the constraint limits the capacity at $0.9 \times C_r$ to avoid pushing the node to its maximum capacity and thus causing unwanted effects, such as queueing. The constant, 0.9, can of course be adjusted
- $\sum_{i \in I} S_{ir} M_i < 0.9 \times Mem_r, \forall r \in R$: Memory capacity constraint: the total memory usage can not exceed the available memory. Limitting the capacity to $0.9 \times Mem_r$ could be useful, for example, to avoid intensive memory swapping to hard disk.
- $\sum_{i,j \in I} h_{lij} T_{ij} < 0.9 \times B_l, \forall l \in N$: This constraint expresses the limit on the bandwidth for each network link l. As before, the limit on the bandwidth is at $0.9 \times B_l$, avoiding effects such as packet loss at near maximum capacity.
- $S_{ir} = 1$, if component i has a fixed deployment on node r. This is used, for example, to make sure that the component that accepts user input and presents output is always located on the hybrid thin client device.
- $S_{ir} = 0, \forall i \in Comp(r_1), r \in R_c \setminus \{r_1\}$: This constraint ensures that none of the components of the set of components of client r_1, $Comp(r_1)$, is deployed on other clients, only on offloading servers or on client r_1 itself.
  This constraint can be omitted when modelling an ad-hoc network, where an infrastructure containing offloading servers is limitted or non-existent.

D. Objective function

The objective function expresses the impact on user experience. A lower figure means less impact and thus a better user experience. The function is a weighted mean of parameters that affect user experience. In this case processing load, memory usage, network traffic and network latency are considered.

All those parameters have to be as low as possible on the hybrid thin client device, but since they are often in contradiction with each other, trade offs have to be made.

- processing load: the processor in the device is inherently slower than the processor in an offloading server causing the processing time to be longer
- memory usage: memory capacity of the device is inherently limited
- network traffic: wireless links have limited bandwidth; network traffic might have to paid for, per byte.
- network latency: the delay caused by sending user events to an offloading server and sending the results back decreases the user experience

$$
\begin{aligned}
impact = \\
& X_1 \sum_{r \in R} (\sum_{i \in I} S_{ir} L_i / C_r) \\
+ & X_2 \sum_{r \in R} (\sum_{i \in I} S_{ir} M_i / Mem_r) \\
+ & X_3 \sum_{l \in N} (\sum_{i,j \in I} h_{lij} T_{ij}) / B_l \\
+ & X_4 \sum_{l \in N} (\sum_{i,j \in I} h_{lij} D_{ij}) \delta_l
\end{aligned}
$$

Of course, it is difficult to quantify the different parts of the objective function in relation to each other. It is, for example, hard to say to what degree network delay has more (or less) impact on user experience than any other parameter. This is highly dependent on the application and the user.
Therefore X_1 to X_4 can be adjusted to determine the weight of each part, $\sum_{p=1...4} X_p = 1$.

E. Validation

The model described above has been tested with different input sets, regarding among others network topology, number of clients, number of software components and interconnection between software components. Fig. 3 shows the average impact per client, when the number of clients increases while the network conditions and the offloading servers are kept constant. All clients connect through the same access network (*AN*) (Fig. 2). The core network is modelled as a ring network and as a full mesh network.

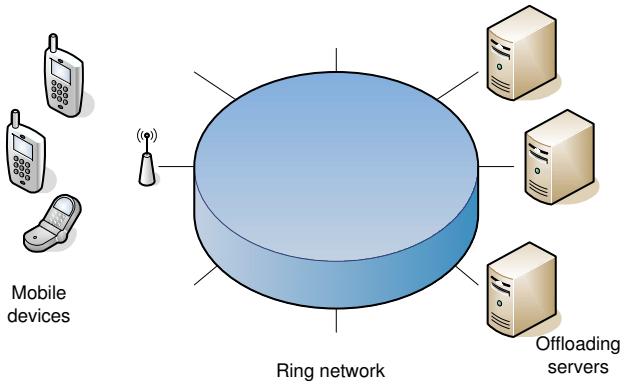

Fig. 2. Network topology (for ring network)

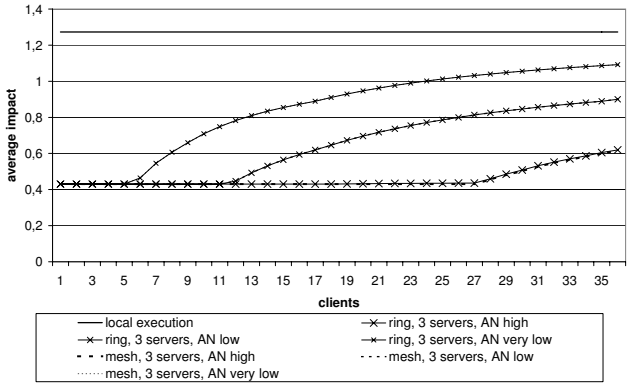

Fig. 3. Impact in relation to number of clients

The access network is varied from high capacity, over low capacity, to very low capacity. High capacity means the access network has the same bandwidth as the core network, low capacity is $1/10$ of the core network and very low is $1/20$. The number of offloading servers is 3 and the number of components per client device is 4.

The graph shows that the network topology in the core network is of little influence: the mesh and ring network give very close results. As long as the access network can cope with the load, all setups give the same result. Once the access network is saturated, more components have to be executed locally, thus increasing the average impact. This is consistent with the situation where a mobile device connects through a wireless link, which imposes the biggest limitations.

The impact when all components are executed locally (*local execution*) is the upper limit.

The same test has been repeated with software components that have more interconnections and that send more data to each other. The results are shown in Fig. 4. The overall trend is similar, but as the access link saturates, the impact increases more rapidly because of the extra incurred network traffic.

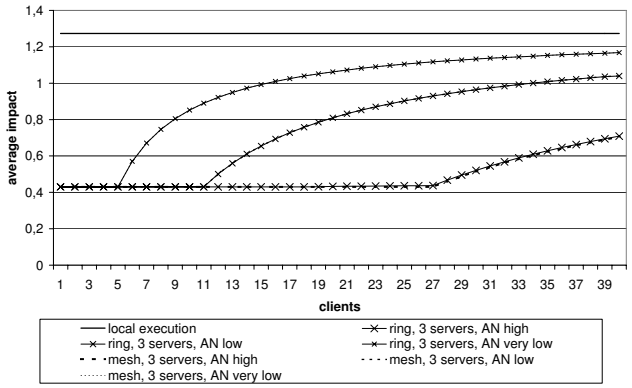

Fig. 4. Impact in relation to number of clients (more dependencies between components)

III. Simplified ILP Model

Since the topology of the core network is of little influence and the access link is the main factor, the ILP model can be simplified. Moreover, since the model tries to keep components, that send much data to each other, on the same node, it can be simplified further.

This simplification is very important when considering runtime deployment adaptation. Since the best deployment has to be generated while an application is running and the solution has to be re-evaluated when the context, in which the application runs, changes, the complexity of the model is an important factor. This is discussed later.

The changes compared to the general ILP model are:

- C_c, C_s, Mem_c, Mem_s: the processing and memory capacities of client and server, respectively.
- B: the capacity of the single access network link.
- δ: an indication of the network latency of the single access network link.

A. Decision variables

- S_i: a binary decision variable. Its value is equal to 1 when component i is deployed on the offloading server, and 0 when deployed on the client.
- h_{ij}: a binary decision variable. Its value is equal to 1 if component i and j are on different node, i.e. when their interaction goes over the network. It is 0 otherwise.

TABLE I
MODELLING A XOR FUNCTION IN ILP

S_i	S_j	$S_i - S_j$	$S_j - S_i$	h_{ij}	XOR
0	0	0	0	0 or 1	0
1	0	1	-1	1	1
0	1	-1	1	1	1
1	1	0	0	0 or 1	0

B. Constraints

- $$\begin{cases} h_{ij} \geq S_i - S_j \\ h_{ij} \geq S_j - S_i \\ h_{ij} \leq 1 \end{cases}$$
  $\forall i, j \in I$

  These constraints express the relation between the two types of decision variables. If S_i and S_j are equal, i.e. if component i and j are deployed on the same node, h_{ij} should be 0. Otherwise h_{ij} should be 1.

  Thus, $h_{ij} = S_i$ XOR S_j, which can not be expressed directly in an ILP model. The constraints above are equivalent to a XOR function (TABLE I) when minimizing the objective function, as the value of h_{ij} will be chosen as low as possible, i.e. 0.
- The capacity constraints are similar to the ones used in the general model.

C. Objective function

Thus, with the assumption that an offloading server is available for the client device, the objective function can be simplified. An offloading server could be elected in a login phase or a subscription based mechanism could be used. In this case the choice of deployment is limited to 2 nodes: the client device and the offloading server.

$$\begin{aligned} impact = & \\ & X_1 \left[\sum_{i \in I} (1 - S_i) L_i / C_c + S_i L_i / C_s \right] \\ & + X_2 \left[\sum_{i \in I} (1 - S_i) M_i / Mem_c + S_i M_i / Mem_s \right] \\ & + X_3 \sum_{i,j \in I} h_{ij} T_{ij} / B_l \\ & + X_4 \sum_{i,j \in I} h_{ij} D_{ij} \delta \end{aligned}$$

D. Validation

The simplified model has been tested with input sets representing different conditions. One result is shown in Fig. 5. It shows the associated impact when the network latency increases. As references, the impact for local execution, which is the upper limit, and the impact for a pure thin client approach are shown.

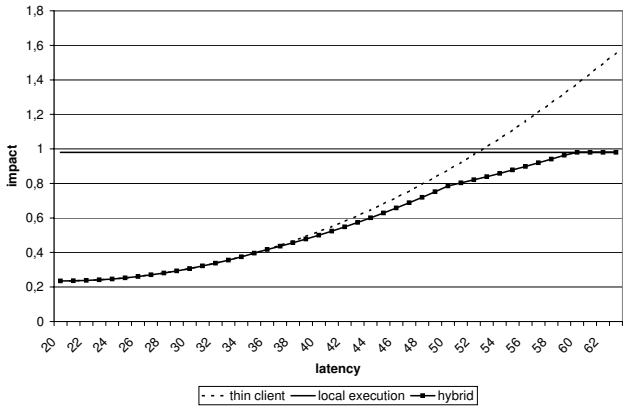

Fig. 5. Impact in relation to network latency

When the network latency is low enough, the model behaves as a pure thin client. All components, except for the component that handles the user interaction, are executed on the offloading server. As the network latency increases, the trade off changes: using the less powerful local resources becomes more feasible than accepting the delay added by the offloading. This causes some of the components to be executed locally to decrease network traffic. As network latency increases further, more components are executed on the hybrid thin client device, until finally everything runs locally, reaching the upper limit (*local execution*).

Determining the impact of network latency on user experience is ongoing research in our thin client research. This is done by subjective tests with different kinds of applications. Similar research is presented in [5], where the effect of network latency on interactive applications on thin clients is quantified.

In the case shown in Fig. 5, the following simplification is used. The indication of network latency, δ, is the square of the actual network latency. This assumes that the impact on user experience increases quadratically in relation to increasing latency.

IV. PRACTICAL USE AND PROTOTYPE

In order to incorporate the decision process in the prototype presented in [1], several approaches are possible. The easiest way is to just add the ILP model in the framework through the use of an ILP solver. This is usable for testing purposes and evaluation, but

proves counterproductive in practice. The resources on the device are too scarce to be used for a problem as complex as ILP solving.

Another possibility is to solve the ILP model on a more powerful server, possibly one of the offloading servers. In this setup, the framework on the hybrid thin client device just monitors the available resources and periodically sends them to the server, where the best suited deployment is determined by solving the ILP model.

Finally, a set of best deployments could be determined beforehand, by varying the different parameters, such as network latency, and solving the model according to those parameters. At runtime, the actual set of parameters could be mapped to the nearest set of known parameters and the associated deployment can be used. While this is less fine grained and less flexible, there is also a decreased overhead at runtime.

V. Future Work

An important piece of the puzzle is the quantification of the parameters describing the software components and their interconnections. This could be determined by profiling the components in an offline manner, in an environment designed for that purpose. A more ambitious approach is to profile the components at runtime by the framework itself, as a sort of self learning mechanism. However, this may prove to have an overhead that is too large.

An interesting alternative to the solving of the ILP model is the use of heuristic methods, that try to determine an acceptable, yet not necessarily optimal, deployment. Research will be done to develop heuristic methods, which will be compared to the ILP modelling.

VI. Conclusion

In order to optimize the quality of experience of users of applications on mobile devices, a hybrid thin client is proposed. This approach allows to dynamically switch between local and remote execution of components of the mobile application. An ILP model, that determines the best deployment of components given certain conditions, has been presented and validated using an intuitive test case.

A simplified ILP model shows that the hybrid thin client behaves as a classic thin client when network conditions are good enough. When network conditions are deteriorated, the deployment shifts to the other extreme: everything is executed on the client device. In between, the best suited composition of locally and remotely executing components is chosen.

The results obtained through the models can be incorporated in various ways in the existing component framework.

Acknowledgment

Raf Hens is a Ph. D. Fellow of the Research Foundation - Flanders (FWO). Filip De Turck is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO). They acknowledge the Research Foundation - Flanders for the funding of their research.

References

[1] R. Hens, F. D. Turck, and B. Dhoedt, “Runtime implementation switching for resource constrained devices,” in *Proc. IASTED International Conference on Software Engineering (SE 2007)*, Innsbruck, Austria, Feb. 13–15, 2007 (in print).

[2] I. Foster and C. Kesselman, Eds., *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[3] R. Wismüller, M. Bubak, W. Funika, and B. Baliś, “A performance analysis tool for interactive applications on the grid,” *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 3, pp. 305–316, 2004.

[4] B. Boone, T. Verdickt, B. Dhoedt, and F. D. Turck, “Design time deployment optimization for component based systems,” in *Proc. IASTED International Conference on Software Engineering (SE 2007)*, Innsbruck, Austria, Feb. 13–15, 2007 (in print).

[5] N. Tolia, D. G. Andersen, and M. Satyanarayanan, “Quantifying interactive user experience on thin clients,” *Computer*, vol. 39, no. 3, pp. 46–52, 2006.