Self Management of a Mobile Thin Client Service

Lien Deboosere, Bert Vankeirsbilck, Pieter Simoens,
Filip De Turck, Bart Dhoedt and Piet Demeester

IBBT - Department of Information Technology (INTEC), Ghent University
Gaston Crommenlaan 8, bus 201, 9050 Gent, Belgium
Lien.Deboosere@intec.ugent.be

ABSTRACT

Mobile thin client computing is an enabler for the execution of demanding applications from mobile handhelds. In thin client computing, the application is executed on remote servers and the mobile handheld only has to display the graphical updates and send input from the user to the remote execution environment. To guarantee a high user experience in a mobile environment, a Service Management Framework is required to prevent users observing lower Quality of Experience due to changes in the available network, server and client resources. Therefore, the Service Management Framework monitors the environment and the Self Management component intervenes when necessary, e.g. by adapting the thin client protocol settings or moving a user session from one server to another. The design of the Self Management component is presented and the performance is evaluated.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Design, Management, Performance

1. INTRODUCTION

In the thin client computing paradigm, computation and storage are shifted from the client terminal to the network. User applications are executed on a remote server and the client device only deals with user interaction and rendering of the screen graphics (e.g. Virtual Network Computing (VNC) [1]). The thin client concept is very promising for mobile users: all applications can be executed without resorting to a restricted mobile version of the application. Furthermore, redundant hardware can be stripped from the device, resulting in potentially energy efficient, *thin* devices.

It is of the utmost importance that the thin client service is ubiquitously available and offers a high Quality of Experience (QoE). Ideally, every user should perceive the same application responsiveness as when running the application locally. To meet these challenges, a Service Management Framework (SMF) is required that aims for a global optimal state of the infrastructure while providing all users sufficient QoE. In Figure 1, three components of the proposed mobile thin client service can be distinguished: a management server, a thin client server and a client terminal. The SMF is distributed among the three components. By monitoring the environment, the SMF detects and adapts to variations in the state of the environment in which the mobile thin client service operates.

By adopting virtualization technologies (e.g. Xen [2]) on the thin client server, each user operates in his own dedicated Virtual Machine (VM). Furthermore, the online migration tool of current virtualization technologies enables moving VMs to other servers without the user noticing [3]. In the next sections, the design and evaluation of the Self Management as a part of the SMF will be discussed.

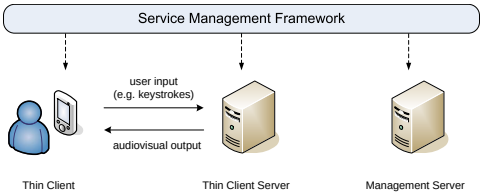


Figure 1: The Service Management Framework (SMF) is distributed among management servers, thin client servers and client terminals.

2. SELF MANAGEMENT

The Self Management component controls and optimizes the mobile thin client service in order to guarantee a high user experience. The Self Management component is distributed among the different levels of the service: the management server, the thin client server, the user's session and the mobile handheld.

When the Monitoring component reports a problem to the Self Management component at a certain level, the Self Management component first tries to solve the problem on its own level (e.g. by adapting thin client protocol settings, or by reserving more resources, etc.). When the problem cannot be solved locally, the component on the higher level is triggered. On the highest level, i.e. the management server, the Self Management component should be able to solve the problem. In the remainder of this article, the design and evaluation of the Self Management component located on the level of the management server is discussed.

The design of the Self Management component is based on the well-known MAPE (Monitor-Analyze-Plan-Execute) model [4] and is illustrated in Figure 2. To enable rapid and simplified implementation of the SMF, Java EJB3 has been used. An EJB is a managed component controlled by a JEE application container. The container controls the lifecycle of the EJBs and their resources.

The ***Monitor*** component sends standard Simple Network Management Protocol (SNMP) trap messages to notify the management

MobiHeld'09, August 17, 2009, Barcelona, Spain.
ACM 978-1-60558-444-7/09/08.

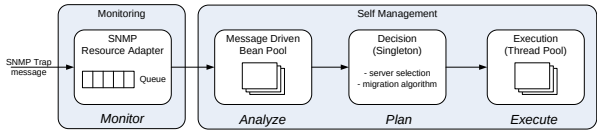


Figure 2: Design of the Self Management component

server of a problem. In order to redirect these SNMP trap messages to the Self Management component, a Java Connector Architecture (JCA) [5] resource adapter is required. The SNMP resource adapter forwards the SNMP trap messages to all registered JEE Message-Driven Beans (MDB).

The ***Analyze*** component, a registered MDB, receives the SNMP trap messages. In order to understand an SNMP trap message sent by the Monitor component, the Self Management component has to know what the object identifier (OID) field of the standard SNMP trap message defines. OID's are described in a Management Information Base (MIB). Based on the OID, the MDB analyzes the problem and for example looks up extra information that could be used by the decision-making component. When all desired information is gathered, the Self Management is ready to make a decision.

The ***Plan*** component, the decision-making component, is a singleton component in order to prevent simultaneous, conflicting decisions. Currently, this component contains two algorithms: *(i)* a server selection algorithm and *(ii)* a migration algorithm. The first algorithm selects a thin client server able to host the user's VM based on the current state of the environment. The migration algorithm selects at least one VM from a thin client server that should be migrated to another thin client server. The target thin client server is selected by means of the server selection algorithm.

The ***Execute*** component executes the decision in a separate thread.

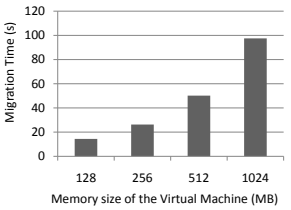
3. EXPERIMENTAL RESULTS

The testbed to evaluate the Self Management component consists of a management server (dual CPU Quad-core AMD Opteron 2350, 8 GB RAM) and 2 thin client servers (Dual CPU Dual-Core AMD Opteron 2212, 4 GB RAM). The adopted virtualization technology in the experiments is Xen 3.1.

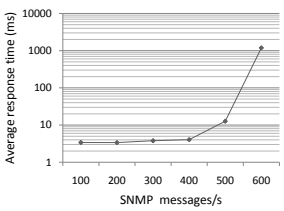
In the experiments, a thin client server becomes overloaded and an appropriate SNMP trap message is sent to the management server. The only solution is to migrate at least one of the VMs running on the overloaded thin client server to another one. The server selection algorithm selects in a round-robin way a server from the list of available thin client servers. The implemented migration algorithm selects a random VM to be migrated to another thin client server.

The time spent in each component of the MAPE-chain presented in Figure 2 is *(Monitor)* 0.58 ms, *(Analyze)* 0.02 ms, *(Plan)* 1.2 ms and *(Execute)* 0.008 ms + total migration time (see Figure 3(a)). As can be seen from Figure 3(a), the total time spent on the live migration of a VM depends on the size of the memory allocated for that VM. The larger the VM's memory, the more information has to be copied to the new server and thus the longer it takes to finalize the migration. It should be noted that at the end of the live migration, the VM is frozen in order to copy the last changed memory state and bring the VM up on the new server. This freezing time is referred to as *downtime*. In our experiments, downtimes up to 300 ms were measured.

From user perspective, the downtime should be kept as small as possible, while from infrastructure perspective, the total time of the online migration is the most important parameter. The faster resources can be released, the faster the server load can be balanced. From Figure 3(a), it can be seen that the migration time is



(a) Total time to live migrate a Xen VM.



(b) Average response time to an SNMP message.

Figure 3: Experimental results

linearly proportional to the memory size of the VM. When a thin client server is overloaded, the migration algorithm has to decide which VM(s) should be migrated based on the memory size of the VMs in order to optimize the thin client service. In Figure 3(b), an evaluation of the performance of the Self Management component under high load shows up to 700 SNMP trap messages per second can be handled within on average 3.5 ms.

4. CONCLUSION AND FUTURE WORK

A high quality mobile thin client service requires a Service Management Framework. When environmental changes are detected, the Self Management component of the Service Management Framework intervenes to optimize the mobile thin client service and guarantee high user experience all the time. The Self Management component currently comprises two decision algorithms: *(i)* a server selection algorithm that selects a thin client server able to run the user's VM and guarantee the desired quality and *(ii)* a migration algorithm that decides which VM(s) should be migrated to another thin client server. The time spent between detecting a problem and the execution of the problem is on average 1.8 ms, while the duration of the migration of a VM is linearly proportional to the memory size of the VM. It was shown that the implemented Self Management component can handle up to 700 reported problems per second. Future work encompasses extending the Self Management component with additional algorithms to further optimize the mobile thin client service.

5. ACKNOWLEDGEMENT

Part of this research was done for the MobiThin Project and has received funding from the European Community's Seventh Framework (FP7/2007-2013) under grant agreement nr 216946. Lien Deboosere and Bert Vankeirsbilck would like to thank IWT-Vlaanderen. Pieter Simoens and Filip De Turck would like to thank FWO-V.

6. REFERENCES

[1] T. Richardson, *et al.*, Virtual Network Computing. *IEEE Internet Computing*, 02(1):33–38, 1998.

[2] P. Barham, *et al.*, Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the 19th ACM symposium on Operating systems principles*, pages 164–177, New York, USA, 2003.

[3] C. Clark, *et al.*, Live Migration of Virtual Machines. In *NSDI '05: Proceedings of the 2nd ACM Symposium on Networked Systems Design and Implementation*, pages 273–286, Boston, USA, 2005.

[4] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[5] R. Sharma, B. Stearns, and T. Ng. *J2EE Connector Architecture and Enterprise Application Integration*. Addison-Wesley, Pearson Education, 2001.