Attacking the Migration Bottleneck of Mobile Agents

Peter Braun
Swinburne University of Technology
Faculty of Information and Communication Technologies
John St, Hawthorn, Victoria 3122, Australia
pbraun@ict.swin.edu.au

Steffen Kern
Friedrich Schiller University Jena
Department of Computer Science
Ernst-Abbe-Platz 2, 07743 Jena, Germany
steffen.kern@informatik.uni-jena.de

Ingo Müller
Swinburne University of Technology
Faculty of Information and Communication Technologies
John St, Hawthorn, Victoria 3122, Australia
imueller@ict.swin.edu.au

Ryszard Kowalczyk
Swinburne University of Technology
Faculty of Information and Communication Technologies
John St, Hawthorn, Victoria 3122, Australia
rkowalczyk@ict.swin.edu.au

ABSTRACT

Mobile agents were introduced as a design paradigm for distributed systems to reduce network traffic as compared to client-server based approaches, simply by moving code close to the data instead of moving large amount of data to the client. Although this thesis has been proved in many application scenarios, it was also shown that the performance of mobile agents suffers from too simple migration strategies in many other scenarios. In this paper we identify several reasons for mobile agents' poor performance, most of them related to the Java programming language, and adumbrate solutions to all of these problems.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications; C.2.4 [**Distributed Systems**]: Distributed Applications

General Terms

Experimentation, Performance

Keywords

Mobile agents, migration protocols, migration optimization

1. INTRODUCTION

Mobile agents have been introduced as a design paradigm for distributed applications. In this paper we solely focus on the problem of performance of mobile agents. One major argument in favor of mobile agents is *code-shipping versus data-shipping*. In a client-server based application, a single remote procedure call might result in a huge amount of data sent back to the client due to the lack of precision in the request. Instead of transferring data to the client where it will be processed and filtered and might cause a new request (data-shipping), this code can be transferred to the location of the data (code-shipping) by means of mobile agents. In the latter case, only the relevant data, i.e. the results after processing and filtering are sent back to the client.

This argument was examined by experiments in the last years for different application domains [6, 7] and it was shown that neither client/server nor mobile agents produce minimal network load in all situations. It is obvious to see that the *code-shipping versus data-shipping* argument is only valid if, simply speaking, the mobile agent's code that has to be transmitted is not larger than the amount of data that can be saved by the use of a mobile agent. It depends on the size of the request, the size of the reply, the code size and some other parameters to decide whether mobile agents perform better than client-server approaches.

We propose to supplement a static design decision between both paradigms [5] with techniques to reduce the migration overhead of mobile agents by increasing the *migration efficiency*. The *migration efficiency* of a mobile agent defines how many code units (on the level of statements, methods, or classes) and data units (variables or objects) of an agent are used (read or written) on remote agencies proportional to the number of code units and data units that have been transmitted. An agent has a low migration efficiency, if many code units or data units have been transmitted superfluously, i.e. they have not been used at remote agencies. A migration efficiency greater than 1 can make sense, since agents' code can be deployed or be cached in advance. It should be clear that the performance of a mobile agent is directly influenced by the migration efficiency.

2. THE KALONG MOBILITY MODEL

When a mobile agent decides to migrate to another host, the underlying agency is responsible to stop agent execution, serialize the agent, and to transfer the agent's state to the destination agency. The destination agency receives and deserializes the agent's state and then re-starts the agent by creating a new thread and invoking a specific method. This

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

framework can be considered as the least common denominator of all mobile agent toolkits. It does not describe how code transmission works, because this can be implemented in different ways. In the following, we use the term *migration strategy* to describe how code and data are relocated.

Today's agent toolkits (e.g. Aglets [4] or Jade [1]) only support very simple migration strategies either based on the *push strategy* (all code is transmitted to the next agency) or *pull strategy* (code is loaded per class from the home agency or a Web server). It is not surprising that neither strategiy leads to minimal network load and minimal application performance in all cases [2].

Our overall goal is to increase the migration efficiency, that is, to develop techniques so that we only transfer code and data that are needed on remote agencies with high probability. The following list summarizes all approaches that we have implemented so far.

1. Using *adaptive transmission of code and data* it is possible for the agent to decide during runtime which classes should be pushed to the next destination and which classes should be pulled at the destination agency later. The agent can decide which data items must be transferred to the next destination and which must be sent back to the agent's home agency.

2. Even if using a pull strategy, the Java class loading mechanism might load classes that are never used, which decreases the migration efficiency. We have implemented a code rewriting technique on the level of Java byte code to prevent *unnecessary class loading*.

3. *Code caching* can be a powerful technique to prevent code transmission and therefore to increase the migration efficiency. Due to security restrictions and problems in distinguishing different versions of the same class, many toolkits disable the built-in Java code cache. We present an alternative approach for code caching that goes beyond Java's capabilities and increases code migration efficiency significantly. Our code caching techniques not only prevents pulling classes but also pushing classes.

4. Finally, we propose to transmit code on the level of single methods rather than on complete class files. We learned that different methods of the same class might have various execution probabilities. Therefore, it makes sense to *split classes* into smaller transmission units on transfer code on the level of methods [3].

All these approaches have been implemented in our new mobility model, Kalong, which is also available as independent software component. Kalong is designed to be adaptable to all Java-based mobile agent toolkits and has been already integrated into Jade [1] and Tracy [2].

3. CONCLUSION AND OUTLOOK

The interest in mobile agents as a new design paradigm for distributed systems seems to have dwindled over the last years. The number of research groups working on mobile agent related research topics is becoming smaller. It is argued that mobile agents were not able to satisfy main expectations, for example regarding network traffic overhead. Vigna [8] states that mobile agents are very expensive and provide worse performance in the general case than other design paradigms, as for example remote procedure call or remote evaluation.

We agree with Vigna on the fact of poor performance, but we draw different conclusions. Instead of abandoning the concept of mobile agents, we propose to improve the migration process of mobile agents. In this paper, we have introduced the notion of *migration efficiency* to describe reasons for mobile agents' poor performance. We have presented several drawbacks of Java-based mobile agents, which all result in a low migration efficiency. We have shown our solutions to all these problems and first evaluations in small networks showed a tremendous performance speed-up of our improved mobility model as compared to simple push or pull-based migration strategies.

4. REFERENCES

[1] F. Bellifimine, G. Caire, A. Poggi, and G. Rimassa. Jade – A White Paper. *EXP in search of innovation*, 3(3):6–19, 2003.

[2] P. Braun and W. R. Rossak. *Mobile Agents–Basic Concept, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers, 2005.

[3] S. Kern, P. Braun, C. Fensch, and W. R. Rossak. Class splitting as a method to reduce the migration overhead of mobile agents. In R. Meersman, Z. Tari, and A. Corsaro, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004, Agia Napa (Cyprus), October 2004, Proceedings, Part II*, volume 3291 of *Lecture Notes in Computer Science*, pages 1358–1374. Springer Verlag, 2004.

[4] D. B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[5] G. P. Picco. *Understanding, Evaluating, Formalizing, and Exploiting Code Mobility*. PhD thesis, Politecnico di Torino (Italy), 1998.

[6] A. Puliafito, S. Riccobene, and M. Scarpa. Which paradigm should I use? An analytical comparison of the client-server, remote evaluation and mobile agent paradigms. *Concurrency and Computation: Practice and Experience*, 13(1):71–94, 2001.

[7] G. Samaras, M. D. Dikaiakos, C. Spyrou, and A. Liverdos. Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment. In D. S. Milojicic, editor, *Proceedings of the First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99), Palm Springs (USA), October 1999*, pages 50–64. IEEE Computer Society Press, 1999.

[8] G. Vigna. Mobile agents: Ten reasons for failure (panel). In A. Joshi and H. Lei, editors, *IEEE International Conference on Mobile Data Management (MDM'04), Berkeley (USA), January 2004*, pages 298–299. IEEE Computer Society Press, 2004.