Using Adaptive Optimization Techniques To Teach Mobile Java Computing

Chandra Krintz
Computer Science Department
University of California, Santa Barbara

Abstract

Dynamic, adaptive optimization is quickly becoming vital to the future of high-performance, mobile computing using Java. These compilation environments have the potential to enable ubiquitous computing on resources that together represent greater computing power than that which can be extracted from existing supercomputers. As a result, we believe that mobile computing requires new curricular directions for compilers and the Java Programming Language that focuses on adaptive techniques, has a performance orientation, and is empirical. We describe such a course that we recently implemented at the University of California, Santa Barbara.

1 Introduction

The Internet is a constantly changing set of high-performance computational, communication, and storage devices. To extract the vast performance potential offered by the aggregation of these devices, programs move from where they are stored to the resources on which they execute. Hence, the programming methodology that enables these mobile programs to use the Internet to employ the computational power that is available is termed *Mobile Computing*. Mobile programs are transferred, in their entirety or in part, over a network in an architecture-independent *transfer format*. Once at the target site, the programs must be converted to native code and executed. Since target machines differ in architecture and capability, programming languages that support mobile computing must be able to facilitate portability without the need for modification. Once written, a program should be able to execute on any device that supports an *execution environment* for the programming language.

The Java Programming Language [7] is the most popular and best-suited mobile language for this environment. Programs are divided into class files (each is an object definition in the object-oriented paradigm) which are stored and transfered in an portable, architecture-independent format called bytecode. A Java execution environment, called a Java Virtual Machine (JVM), loads local or remote class files as they are used during program execution (on-demand) and dynamically compiles (or interprets) bytecode into native code. JVMs provide many other runtime services including code optimization, method dispatch, memory allocation, garbage collection, thread scheduling, exception handling, etc. In addition, JVMs can also implement security measures that ensure type-safe execution as well as adherence to system- and user-defined security policies.

However, portability, security, and a full-service runtime services come at a cost in mobile program performance. There are four primary sources of overhead that degrade mobile Java program performance:

- *Transfer delay:* the time to transfer the non-local portion of programs to the target site for execution. The widening gap between network and processor performance and the widespread use of the Internet make it increasingly difficult to maintain acceptable transfer times.
- *Verification delay:* the time to check that an untrusted program will not maliciously or unintentionally exploit the target system adversely.
- *Compilation delay:* the time to translate the portable transfer format to native code of the target architecture and possibly to optimize it.
- *Execution delay:* the opportunity cost (loss of optimization potential) introduced by the use of a portable transfer format to represent the program. The trade-off between optimization time and execution speed make it exceedingly difficult to do both efficiently.

Each of which occurs *while* the program is running, and as such, are experienced by user as long startup times, slow execution speeds, and intermittent interruption. We consider each of these sources of overhead as part of **overall mobile Java program performance**.

Improving Java performance is vital for the continued success of Java for mobile computing and doing so requires a keen understanding of the language mechanisms and services that enable mobile execution as well as the performance implications of each. **As such, we believe that mobile computing requires a new educational approach for compilers and the Java Programming Language that *focuses on adaptive techniques, has a performance orientation, and is empirical*.** These curricular directions, when interlaced, define a novel alternative to teaching these topics that enables greater impact not only on student learning but also on Java optimization research and the future of mobile computing.

We have developed such a course at the University of California, Santa Barbara that exposes the mechanisms and overheads of mobile Java computing. The course provides students with opportunities for hands-on experience with tools that can be used to exploit optimization opportunities. The availability of cutting-edge Java optimization tools as open source played a key role in the development and success of this course. Its implementation, described herein, has recently been selected for inclusion as a JikesRVM [1] [10] Teaching Resource [11] by Michael Hind [8], Dynamic Optimization Group [6] manager at IBM T.J. Watson Research Center.

In the following sections, we describe the course we developed as well as the adaptive infrastructures we used to teach students about mobile Java program performance. We also detail two projects that the students implemented as part of the course. Finally, we discuss the impact such a curricular implementation has had on students and Java research in general.

2 UCSB Graduate Course on Mobile Java Computing

We developed the course, entitled Implementation of Programming Languages (CS263), and taught it in the Winter quarter of 2002 at the University of California, Santa Barbara. We emphasized both

[1] JikesRVM stands for the Jikes Research Virtual Machine which was, until recently, called the Jalapeño Virtual Machine [2]. It is available as open source for x86 and PowerPC architectures [10].

Introduction (Class/Java)	Java Bytecode	Dynamic Class Loading
Dynamic Linking	Garbage Collection	Virtual Machines
JikesRVM	ORP	Dynamic Compilation
Adaptive Optimization	Annotation-based Optimization	Transfer Delay Optimization
Microsoft C# & CIL	Mobile Security	Mobility (strong/weak)

Table 1: Topics Covered in UCSB CS263 (in row-order). Transfer Delay optimizations includes compression, non-strict execution [16], and class file splitting and prefetching [15]. Topics in mobile security include cryptography, proof-carrying codes [17], and software-fault isolation [1].

Java and Internet-based mobile computing. We required that the students have some programming experience using the Java Programming Language [7]. All course materials can be found here [5].

The goal of our course was to provide students with a thorough understanding of the different services and mechanisms required for mobile Java execution. In addition, we wanted to emphasize performance. To this end, we developed the course as outlined in Table 1. We used each of class periods to discuss each of the different topics and the impact each has on mobile program performance. In particular, we identified how each the various mechanisms impacted transfer, verification, compilation, and execution delay.

To enable hands-on experimentation, we also articulated projects for which students identify and empirically evaluate the performance of various mobile computing JVM mechanisms and services. Both class discussions and projects incorporate cutting-edge Java research technologies that include dynamic and adaptive compilation [2, 4, 9], garbage collection, proof-carrying codes [17], dynamic compression format selection [14], and non-strict execution [16].

So often students are unclear about the applicability of what they are learning or what is required of them as part of a classroom experience. Likewise, graduate students are easily frustrated when they are unable to see how course material can be extended into viable research projects. By offering students clearly defined empirical experiments and opportunity to work with real, on-going mobile computing research tools and infrastructures, we hope to address both of these issues.

Recent release of open-source dynamic and adaptive compilation environments (Java Virtual Machines), e.g., ORP [4] and JikesRVM [10], is the primary motivation for our development of this novel curricular direction. Each of these systems implements a set of very different design decisions and offers many opportunities for empirical analysis, investigation, and improvement. The environments enable students to directly access and manipulate extensive Java runtime services, e.g., memory allocation and garbage collection, thread management, synchronization, exception handling, etc. Both implement an optimizing compiler with which we can investigate compiler optimizations and the trade-off between compilation and execution delay. In addition, both systems can be easily extended to investigate novel services, e.g., dynamic compression selection, encryption, non-strict execution etc., and interactions between new and existing services.

3 Mobile Java Computing Project Examples

In this section, we briefly describe two of the projects implemented by our students as part of CS263. The goal of each project is to provide students with hands-on experience with cutting-edge Java research techniques and empirical measurement and analysis. Our projects address all aspects of mobile program performance using Java: transfer, verification, compilation, and execution time.

The first project we describe addresses transfer delay; the second addresses compilation and execution delay. Descriptions of all of the projects developed for this class can be found on the course Web site [5]. Students implemented a single project throughout the duration of the quarter and worked in groups of one or two members.

3.1 Dynamic Compression Format Selection

One key factor in overall mobile program performance is transfer delay. Java class files include all of the information required to enable mobile execution and security. Since the transfer format of class files is architecture-independent, extra information must be included to enable compilation on different architectures. In addition, any information required for verification of type safety at the destination is also included in each Java class file.

The primary technique for the reduction of transfer delay is compression: Compactly encoding files so that redundancies are removed and the overall transfer (or storage) size is decreased. Using compression, Java class files are commonly archived together and compressed. Once at the destination the archive must be decompressed on-line and as such, this time must be considered part of mobile program performance.

Compression techniques offer the potential of significantly reducing transfer size. However, compression algorithms can introduce significant decompression time due to the inherent trade-off between compression ratio (compressed_size/original_size) and decompression rate. In addition, both compressed transfer and decompression time depend on the performance of the underlying resource, e.g., CPU speed and load, memory availability, network bandwidth and latency, etc.

The goal of this project is to analyze the trade-off between compression ratio and decompression time and the performance of various techniques in different Internet performance configurations. To implement the project, students measure the performance of various compression techniques (compression ratio and decompression time) that are available from the Web. The students then measure the transfer delay imposed by transfer of compressed files for a wide range of Internet links. The links include modems, broadband, and various cross-country common carrier. Students measure bandwidth and round trip time for these transfers which are performed at different times of the day (to capture different network performance characteristics). Lastly, the students introduce CPU load on a machine and repeat the decompression time study.

The analysis of data generated from this project indicates that different compression techniques offer the least transfer delay in different situations. That is, if we know (or could *predict*) the network and target machine performance, we can select a format that will result in the best transfer performance. The research project implemented from this class project uses dynamic resource prediction to dynamically select the best compression technique and reduce transfer delay [14].

3.2 Adaptive Optimization: Analyzing On-line and Off-line Profiling

Dynamic compilation and optimization of a mobile program can cause significant delays during execution since it must occur *while* the program is running. Most systems attempt to reduce this delay via adaptive optimization [9, 18, 3, 4]. The program is compiled first with the fast compiler, and then "hot" methods are compiled later with the optimizing compiler based on dynamic information gathered during execution. Such systems use on-line measurement or sampling of program execution to decide when a method is ”hot” [9, 4, 3].

Alternately, we can perform such measurement (profiling) off-line to reduce the overhead associated with doing so on-line. This type of compilation is called *annotation-based optimization* [13] and also substantially reduces compilation delay. It does so, not only by indicating the ”hotness” of methods but also by performing other types of analysis off-line and communicating the result to the compilation system. The compilation system can then bypass the analysis and proceed directly to straightforward, efficient optimization.

The goal of this project is to enable students to develop an understanding of on-line and off-line profiling and to empirically evaluation the limitations of both. For this project, the students measure the performance of on-line profiling for three different adaptive optimization systems: Jikes Research Virtual Machine (JikesRVM) [10] from IBM T.J. Watson Research Center, the Open Runtime Platform (ORP) [4], and HotSpot [9] from Sun Microsystems. Students time the execution of numerous benchmarks. These results are then analyzed and compared against an off-line profiling.

To enable evaluation of off-line profiling, we have implemented as part of ongoing research [12, 13], *annotation-aware* versions of both JikesRVM and ORP. The students use these systems to collect profile information for different benchmarks. The students decide what information is collected and how it is to be used during annotation-guided optimization. Initially, the students use JikesRVM and ORP on-line sampling systems to perform off-line profiling. This information (”hot” method identification) is then communicated to the optimization system (either integrated within class files or read from disk) and used to guide compiler selection (optimizing compiler for hot methods and the fast compiler for cold methods).

Since off-line profiling is effected by the choice of input used when gathering the profile, students must consider the impact of off-line profiling when inputs different from those used on-line are used off-line. This is commonly referred to as *cross-input* profile-guided performance. For example, if the profile from off-line execution for one input identifies hot methods that are not hot when a different input is used, poor performance may result. The students generate profiles using different inputs and use those to guide optimization. The students then analyze the differences and determine how improvements can be made. The research project implemented from this class project combines off-line annotation-based profiling with on-line sampling to achieve low on-line overhead and improved cross-input annotation-guided performance [12].

4 Conclusion

We believe that advances in mobile computing require new teaching directions for compilers and the Java Programming Language that *focus on adaptive techniques, has a performance orientation, and is empirical*. To this end, we have developed a course at the University of California, Santa Barbara called Implementation of Modern Programming Languages [5] that emphasizes Java and mobile program performance. The course introduces students to a wide range of topics that can potentially improve Java program performance and enable the continued success of Java as a viable and high-performance mobile computing platform. We detail two projects that enabled students to gain hands-on experience with cutting-edge Java research technologies and empirical evaluation. The technologies include the JikesRVM [10] and ORP [4] adaptive Java optimization environments, dynamic compression selection [14], and non-strict execution [16].

Overall the class has been successful and has had far reaching impact on student education as well as successful Java research. Class attendance was 45 at the beginning and 38 at the end. A total of 20 projects were completed, many of which were significantly beyond our expectations and the original project description. In addition, many students have inquired about extending their projects as directed studies or pursing research in this area.

References

[1] A. Adl-Tabatabai, G. Langdale, S. Lucco, and R. Wahbe. Efficient and language-independent mobile programs. In *Programming Language Design and Implementation*, May 1996.

[2] B. Alpern, C. R. Attanasio, J. J. Barton, M. G. Burke, P.Cheng, J.-D. Choi, A. Cocchi, S. J. Fink, D. Grove, M. Hind, S. F. Hummel, D. Lieber, V. Litvinov, M. F. Mergen, T. Ngo, J. R. Russell, V. Sarkar, M. J. Serrano, J. C. Shepherd, S. E. Smith, V. C. Sreedhar, H. Srinivasan, and J. Whaley. The Jalapeño virtual machine. *IBM Systems Journal*, 39(1), 2000.

[3] M. Arnold, S. Fink, D. Grove, M. Hind, and P. Sweeney. Adaptive optimization in the jalaepeño jvm. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Oct. 2000.

[4] M. Cierniak, G. Lueh, and J. Stichnoth. Practicing JUDO: Java Under Dynamic Optimizations. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, Oct. 2000.

[5] Cs263: UCSB graduate course - the Implementation of Modern Programming Languages. `http://www.cs.ucsb.edu/~ckrintz/classes/cs263/`.

[6] Dynamic optimization group at ibm t.j. watson research center. `http://www.research.ibm.com/dynamicopt/`.

[7] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[8] Michael hind. `http://www.research.ibm.com/people/h/hind/`.

[9] The Java Hotspot performance engine architecture.

[10] Jikes research virtual machine. `http://www-124.ibm.com/developerworks/oss/jikesrvm`.

[11] JikesRVM teaching resources. `http://www-124.ibm.com/developerworks/oss/jikesrvm/info/course-info.shtml`.

[12] C. Krintz. Combining off-line annotation with on-line adaptation. Technical report, University of California, Santa Barbara, 2002.

[13] C. Krintz and B. Calder. Using Annotation to Reduce Dynamic Optimization Time. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, Oct. 1998.

[14] C. Krintz and B. Calder. Reducing Transfer Delay with Dyanamic Selection of Wire-Transfer Formats. Technical Report UCSD CS00-650, University of California, San Diego, Apr. 2000.

[15] C. Krintz, B. Calder, and U. Hölzle. Reducing Transfer Delay Using Java Class File Splitting and Prefetching. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Nov. 1999.

[16] C. Krintz, B. Calder, H. Lee, and B. Zorn. Overlapping Execution with Transfer Using Non-Strict Execution for Mobile Programs. In *Eigth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998.

[17] G. C. Necula. Proof-carrying code. In *The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 106–119, 1997.

[18] M. Plezbert and R. Cytron. Does just in time = better late than never? In *Proceedings of the SIGPLAN'97 Conference on Programming Language Design and Implementation*, Jan. 1997.