UNIQUE*SOFT
1530 E. Dundee Road, Suite 100
Palatine, Illinois 60074
(847) 963-1777
info@uniquesoft.com

Performance of Java Code Translated from COBOL

Version 1.0

1 Overview

The purpose of this white paper is to discuss issues with the performance of Java code that has been translated from COBOL. When migrating from COBOL to Java, which here we will take as also meaning migrating from the mainframe to Linux, there is a natural question about whether the performance of the new system matches that of the original. In some system, especially ones that interact heavily with the user, the overall performance may not be a major concern. However, for other systems that are heavily based on data and transaction processing, the performance may be a major concern. This paper describes some of the underlying issues, discusses options for addressing the issues, and provides some benchmark performance data.

2 Translation Challenges

Mainframe computers are powerful processors and, in terms of performance, can excel at operations on large-scale data bases, high-throughput transaction processing, and bulk data processing. However, this power has traditionally come at the cost of using a legacy language such as COBOL. It is common for organizations to plan to move from a mainframe/COBOL environment to a more modern one such as Linux/Java. However, Linux has very different performance characteristics than the mainframe, and Java applications have different performance than COBOL applications. For performance-critical applications, these differences can prove to be a challenge.

There are multiple hardware and software choices that can be made for the new system that affect performance. For example, a project typically has some choice over which database is used, the data pipeline for the suite of applications, amount of physical memory, the speed of the disks, the number of processors/cores, the Java virtual machine, the quality of the COBOL-to-Java translation, the algorithms used in the applications themselves, and the ability to leverage data parallelism in the input.

From the perspective of the COBOL-to-Java translator, only the quality of the translation is directly controllable. With customization, the algorithms can also be affected, but this requires advanced analysis and transformation capabilities. Both possibilities are discussed in the rest of this section.

2.1 Quality of the COBOL-to-Java Translation

COBOL itself has characteristics that make it amenable to high-performance execution. For example, COBOL is a compiled language with stack allocation, pointers, unions with no run-time cost of conversion between types, and no run-time dispatching or type inference.

Java, on the other hand, runs on a virtual machine, has heap allocation, does not have unions, incurs overhead when converting between types, and uses dynamic dispatching and run-time type inferencing. While it is possible to minimize the amount that these features are used, that usually comes at the cost of having code that is not very “Java like”. A common complaint with translators is that the resulting Java code is unreadable and unmaintainable, which defeats much of the purpose of the migration.

COBOL programs frequently use data type redefinition, which makes translation to efficient and maintainable Java difficult. COBOL’s lack of enumerated types or built-in primitive types such as Booleans also causes problems. The common work around is to use strings of characters and 88-level definitions. If these are translated into Java strings and string comparisons, the performance penalty can be quite large. However, if this data is used in a redefinition, written to external storage, used as global data, or passed to an external program, no other representation may be feasible.

2.2 Translation at the Algorithm Level

Given the high performance of the mainframes and the relative difficulty of encoding more complex algorithms and data structures in COBOL, a direct translation to Java often leads to inefficient code. For example, it is common in COBOL to use simple bubble sorts, linear searches across data sets, and multiple short connections to external resources. In a more modern architecture, one would expect to use efficient sorting (usually from a predefined method), algorithms and structures for efficient searches and lookups (such as hash tables), and optimized database access.

3 UniqueSoft Solutions

UniqueSoft has a customizable tool suite called D*Code that is used for migrating an application from one platform and/or language to another. The translations from COBOL to Java can be tailored to specific project needs. For example, the code can be structured similar to the original COBOL or can be translated into a full OO application, and the same data declaration can be translated different ways depending on the usage.

D*Code can also be used to replace code at the algorithm level by making systematic changes to the code. At the simplest level, calls to programs or performs of paragraphs can be replaced with calls to faster alternatives. More complicated transformations can also be done, though. For example, the code can be adjusted to take into account changes to the target environment, such as accounting for breaking up the input into multiple data sets for distributed processing or changing from a linear search through a VSAM file to a SELECT call to an external database server.

4 Performance Comparisons

It is difficult to get a direct comparison for the performance of code on the mainframe and code on PCs. Many factors, such as memory access latency, disk speeds, etc., affect the overall performance and skew a one-to-one comparison of execution times. However, COBOL and C have roughly the same performance characteristics in terms of language elements (stack allocation, pointers, etc.), so using a translation of COBOL to C will give a reasonable benchmark for applications that do not rely heavily on user interaction or access to external resources.

Using the GNU cobc compiler, we measured the performance of various data-intensive COBOL programs and set this as the baseline. By this benchmark, the Java code produced from the COBOL performed approximately seven times slower than the C version, and the code from a popular commercial translator performed at approximately 160 times slower. Of course, the actual numbers depend strongly on the COBOL program, but the disparity in the numbers highlights the benefits of advanced optimizations.

5 Summary

It is unlikely that any Java application on a Linux server will match the performance of the original COBOL application on a mainframe. However, except where performance is not an issue at all, a naïve and literal translation of the COBOL code will typically lead to unacceptable performance. D*Code can provide significantly better performance by performing deep analysis on the code to translate the COBOL into efficient native Java. D*Code can also be customized to provide more large-scale and systematic transformations, such as changing how sorts are done, replacing data access with more efficient mechanisms, and accommodating cloud-based implementations.

D*Code has additional capabilities that are often useful for COBOL migration projects. For example, D*Code can extract business rules from code, separate the code into use cases, identify dead or replicated code, perform detailed metrics analysis, and create test cases for the system. Please contact UniqueSoft for more information.