**Security and Performance Differences Between Java and C++**

Jared Mercado

CSC450: Programming III

Prof. Haseltine

February 11, 2024

**Security and Performance Differences Between Java and C++**

Multi-threaded applications or programs with concurrency are almost essential in high performance solutions. Two of the most popular and most used languages, Java and C++, both provide libraries which facilitate the implementation of the concurrent functions. However, the work needed to optimize the performance of concurrency or the risk toward data security differs between the two languages. Lack of performance optimization can lead to crashes and instability of concurrent program. Likewise, the lack of security measures centered on threads leads to unwanted changes in data structures and circumvention of data access protocols. It is important to realize the differences in the concurrency libraries of a language and analyzing the two languages will help us understand the shortfalls we need to avoid. Java and C++ both require the developer to code to avoid the weaknesses of both languages. Without the understanding of how concurrency operates in C++ and Java programs written in these languages may suffer from performance issues and data insecurity.

When analyzing the problems of performance in Java and C++ functions that allow parallelism it is critical to understand how each language compiles the code. Java and C++ use different compilers to turn the written human-understandable code into machine language. All C++ compilers will compile the code directly to machine language so that the execution of the code runs independently on the platform it is made for. Contrarily, Java’s compiler breaks the code down to byte code and executes the code in a supplemental Java Virtual Machine; the byte code is executed in chunks by the Java Virtual Machine making it useful for code that is not platform specific. It should be noted that Java can still have better load or processing times depending on the application and the programmer, but it is still on average slower than C++ (Prechelt, 1999). This bytecode interpolation causes slower execution since each line of Java code is evaluated. Especially in threaded applications, Java’s Garbage Collector will pause every thread during cleanups thus making the execution time exponentially slower. C++ creates the faster applications since the code written for a given platform runs solely on the hardware of that platform.

Also, the memory management of both languages differs in how the developer has to implement object retention and memory allocation. C++ requires the user to define their own memory management protocols which can make the program more memory optimized than other languages with garbage collection but can lead to memory leak and overflow issues. The developer usually creates pointers in C++ to help keep track of objects and to help deallocate memory when objects are no longer needed; the use of the smart or unique pointers help mitigate the risks of handling memory allocation on the side of the developer but it still relies on them to set the objects scope limits. Meanwhile, Java has an entity in the Java Virtual Machine called Garbage Collection which deletes objects or deallocates memory once the objects leave their respective scopes. Developers do not need to setup Garbage Collection as it automatically runs during execution. (However, the Garbage Collection configuration can be adjusted in some cases, such as using the -XX:ParallelGCThreads flag to set how many threads are set to handle Garbage Collection.) Objects are deallocated when they leave their scope in Java so it is important for the developer to understand the length of class and method scopes so objects in their applications do not persist longer than they should.

Lastly, on performance it is generally more difficult to code with C++ on different platforms optimally. Java is often a better choice for development on different platforms since the Java Virtual Machine handles the discrepancies between platforms and the development time that would have been spent on coding for different platforms could be used for optimizing the Java solution. However, C++ can give the developer more of a performance boost compared to Java since engine running Java bytecode and baseline memory management features of Java makes the language inherently slower than the developer managing it themselves.

Handling potential security vulnerabilities is also critical to software design and analyzing how vulnerabilities are avoided in C++ and Java is great for understanding this concept, especially with designs utilizing concurrency. The data safety of a language is usually defined in a few concepts: type safety, memory safety, and built in safety features. Java is a newer language with newer development features and is often considered the less vulnerable to security threats when compared to C++. Java is a strongly typed language and checks the type of data being passed between functions; the use of strong type checking makes it much more difficult to pass data that is not meant to be manipulated between functions. As opposed to C++ which allows for generic types that are only checked during runtime Java can restrict access to data structures from being passed to functions during compilation.

Java, in addition to its type management, manages memory to prevent common allocation mistakes. Overflows and other memory issues are avoided in Java since the use of a Garbage Collection system manages the memory allocation and deallocation during execution. At the cost of “stop-the-world” pauses for the Garbage Collector memory issues are often not a problem in Java applications (Li, Wu, & Chen, 2018). In C++, the developer must handle the allocation and deallocation of all objects manually which can make the application prone to memory issues; this proves especially difficult to manage in multi-threaded applications since the data accessing and writing of different threads can lead to data races or deadlocks in the program. Finally, Java has some built in features which can help to fix security vulnerabilities before they happen. Java offers a sandbox environment to test out code before actual deployment which makes performance testing and bug catching significantly easier. C++ has no native features and usually relies on how the developer implements safety checking—either through the use of best practices or by implementing external libraries. Considering the advantages of native Java compared to C++ in regards to security vulnerabilities many developers recognize Java as the more secure language.

Java and C++ both are still great languages for development but it is critical to understand the weakness of both languages. C++ is often considered the more performance optimizable language when compared to Java. Java has the Java Virtual Machine which takes the compiled bytecode and runs it in blocks during execution: the step-by-step execution of the bytecode and the automatic memory management leads to a slower execution time of the Java program when compared to fully compiled languages like C++. The Java Virtual Machine, however, does improve on development time since the code written can be ran on multiple platforms. If performance optimization is critical to the application, then C++ is the more suitable language to develop in when compared to Java. The inverse is true when considering the security vulnerabilities of the languages. Java offers more security features—like how the language being strongly typed checks against type errors or how the Garbage Collection system guards against memory mistakes. Additionally, Java has some security features built into the language, like sandboxing, which C++ lacks. Java is then considered the more secure language as opposed to C++ being considered the more optimizable language. Knowing these aspects of these two languages can help us as developers decide which language to use in our applications and how a language can affect the performance or security of the application.

**References**

Prechelt, L. (1999). Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences. *Communications of the ACM, 42*(10), 109–112. <https://doi.org/10.1145/317665.317683>

Li, H., Wu, M., & Chen, H. (2018). Analysis and Optimizations of Java Full Garbage Collection. *Proceedings of the 9th Asia-Pacific Workshop on Systems*, 1–7. <https://doi.org/10.1145/3265723.3265735>