Performance Optimization for MapObjects – Java Edition Applications

Authors

Drew Saunders, QTSI Senior Software Engineer
Eric Miller, QTSI Staff Software Engineer
Greg Wilson, QTSI Staff Software Engineer
Tim Saults, QTSI Software Engineer

Abstract

The success of a web-based application is almost as dependent on performance as capability. Regardless of the extensive capability of a GIS application it will not be fully utilized if response time is slow. This paper focuses on optimizing the performance of a distributed application within the client software. Techniques to identify problem areas and engineering solutions will be discussed. These include techniques for memory allocation, rendering selection, and layer creation and manipulation and how to determine which technique is best suited for the application. The Java Virtual Machine (JVM) and the MapObjects - Java Edition will be discussed in detail.

Introduction

QTSI is designing a GIS to mine an extensive database of earthquakes. The database contains data for millions of events and grows daily. The primary users are analysts and seismologists who desire topography, geologic maps, and imagery as backgrounds for their event data. To be effective and therefore utilized, the software must have a reasonable response time.

Instantaneous response is not always obtainable, or reasonable. To achieve optimum performance there are many accepted software practices that can be utilized. These practices include defining a robust architecture, establishing benchmarks, identifying problem areas, and implementing enhancements.

Architecture

To efficiently serve this extensive dataset QTSI implemented a complete multi-tier architecture. The solution consists of an Oracle database to store and serve the seismic event data. ESRI's ArcSDE (Arc Spatial Database Engine) provides spatial queries and the ability to efficiently store data with a spatial component. ESRI's ArcIMS (Arc Internet Mapping Server) provides multiple

servers to serve specific types of data. The QTSI implementation currently utilizes the feature server (seismic event data) and the image server (imagery). ArcIMS manages these servers using an application server that provides performance, scalability, and reliability. The user interface is provided by a Java applet developed using MapObjects – Java. The applet executes within the Java Virtual Machine (JVM), which can be a plugin within a standard browser (Netscape, Internet Explorer, etc). The user interface can also be built as a stand-alone application.

While the distributed nature of this architecture can dramatically improve performance it is also vulnerable to many bottlenecks. The most obvious bottleneck is the bandwidth required to transmit data to the different components. The advantages of having a central data repository with distributed processing and access far outweigh the disadvantages.

Benchmarks

When starting a software task it is often hard to quantify performance requirements. Initial expectations can be stated, but they may be unrealistic. After developing a prototype application, benchmarks can be established and expectations more firmly established. These benchmarks should be a regular part of the development cycle. As new functionality is added the benchmarks should be regenerated to verify the new functionality does not unexpectedly affect performance.

Benchmarks can be easily implemented by instrumenting the code with simple print statements to display elapsed time. These time prints are placed after major sections of code or method calls. These timings are easily turned off when not generating benchmarks. The small amount of overhead incurred by these timings easily pays off in the development and maintenance of an optimized release.

Verifying the validity of a benchmark can be difficult, especially in a distributed environment. For instance, it is difficult to verify that something else generated substantial network traffic when a request or response is transmitted. Having a controlled test environment alleviates much of this concern. Even in this optimal case, it is wise to perform multiple timings and take the average to establish a valid benchmark.

Identifying Problem Areas

With the aid of benchmarks, analyze the performance of the software. Examine the benchmarks to determine the sections of code requiring the most processing time. These sections are where optimization should first be attempted. A 20% improvement to a section of code that executes 50% of the time results in a larger performance improvement than a 20% improvement in code that only executes 10% of the time (10% vs. 2% overall improvement). When a section is optimized, move on to the next time intensive section.

Once a section of code is selected for analysis there are a variety of tools available for use. There are many commercial tools available and each has its own niche. In essence, to analyze code thoroughly, a suite of tools is desired. In selecting tools, care should be taken to ensure that the measurement tools are as unobtrusive as possible. Verify that measurement tools do not alter the results of the benchmarks. The benchmarks are only useful when obtained and interpreted correctly.

There are also many free system utilities that can be utilized. For example, on the Unix platform tools such as top (process statistics), vmstat (virtual memory monitor), mpstat (CPU statistics), iostat and netstat (network statistics) provide useful insights into a process's resource utilization. The JVM and MapObjects - Java provide tools to monitor the internals of resource processing. The JVM has a console reporting errors and vital statistics plus a memory monitor reporting allocation and availability of memory. MapObjects – Java contains a thread summary display and a memory monitor via an included Sun package. On the server Oracle, ArcSDE, and ArcIMS provide detailed log files to record their processing states.

Implementing Enhancements

When QTSI's prototype applet was first implemented it was determined that the transfer of data from ArcIMS was extremely slow. The ArcIMS server was a Sun E250 and it was assumed that it was overburdened as a file server, database server, and now an ArcIMS server. However, in reality the CPU was 90% idle and the network traffic was low with very few collisions. The log files revealed that the network was incorrectly routed and that ArcIMS requests were being transmitted out through the ISP and back into the local server. Perhaps this would be acceptable if the uplink speed was greater that 256kbs!

Using ArcSDE QTSI spatially enabled the extensive event table. It is possible to transmit to the client (for geographic display) every attribute for an event. This is probably unnecessary and unwise when considering the required bandwidth. Choosing a subset of attributes (such as location and a few primary fields) to request from the server can drastically reduce the amount of transmitted data and significantly improve performance.

One of the primary concerns of a large vertical database is the ability to access discrete values efficiently. If a request is made for millions of events (features), transmission and display of the features may be prohibitively slow. If a small subset is requested, the response time should be much quicker. Using sqlplus, an Oracle tool, we can request a small number of events and have them displayed alpha-numerically almost instantaneously. ArcIMS should mimic this quick access with only a small amount of additional time to display the events geographically.

However, it appears accessing a subset of the data is slower than accessing the entire table or layer. The ArcIMS log files revealed that the MapObjects – Java selection set query, makes two queries from the ArcIMS server. The first query returns the entire table and the selection set returns only the desired data. This bug has been reported to ESRI and a work around has been implemented so that the entire layer is not returned.

Even though end users and developers do not have access to the MapObjects – Java internals it is important to analyze the code's performance. This allows verification that an implementation using the MapObjects – Java is optimum and also highlights problems in the API that need to be brought to the attention of the ESRI development team.

Another area where performance can be enhanced is in the calculation and display of large, complex polygons. Instead of storing the points of the polygon in a spatially enabled table and requesting via ArcIMS for display in the client, it may be faster to request the parameters required to generate the polygon and calculate the points for the polygon prior to insertion into a memory feature layer. The performance difference can be ascribed to the requirement of transmitting hundreds of points over the network versus transmitting a few values and calculating the shapes on the client.

Since the applet executes in the JVM, garbage collection is a potentially time consuming process to free memory for reuse. If an application works with large amounts of data, minimizing the number of times garbage collection is performed can increase performance. If sufficient RAM exists on the client machine, set the initial size of the JVM memory allocation to 512mb, instead of the default 64mb. There are additional startup arguments that are applicable to other applications.

When creating an applet, obfuscating the code not only protects proprietary code. It also helps with the download speed of the applet since obfuscated code is smaller and many obfuscators optimize code. To decrease initialization time of an applet, consider installing its jar files in a common location to be accessed by users (e.g. the JVM's lib/ext directory). This will prevent potentially slow downloads of the applet each time it is loaded.

Perception Enhancements

Another important consideration is the perception of an applet's performance. Users can become disenchanted with a tool when they have a misinterpretation of the tool's performance. Care should be taken that the user is always aware of what the tool is doing. Never let the user guess whether the tool is processing or it is sitting idle.

A potential area of misinterpretation for any web application and particularly a MapObjects –

Java applet is the effects of multi-threaded software. When retrieving a layer from a data store, MapObjects – Java spawns a new thread to handle the request and wait for the response. By using threads, MapObjects – Java allows multiple requests to be performed simultaneously. Obviously, this is much more efficient then requiring each request to be submitted and received before the next request is submitted.

Depending on the data requested there could appear to be a performance problem to the user. For example, imagine two datasets are retrieved from distributed sources. The first dataset is very large and will take approximately 5 seconds to transmit to the client. The second data set is small and will take approximately 1 second. Since the second data set is received first, the map may start displaying it first, even though it is the second data set. When data from the first dataset is received the drawing thread is interrupted and redrawing of the map begins again. This time the first dataset is drawn first, as expected, and then the second dataset is drawn.

While this behavior can be explained, it is disheartening to the user to watch the data start to appear and then be cleared, only to be redrawn again. In this case, it may be worthwhile to wait for data to be received before it is added to the map and available for the drawing thread. Forcing the drawing thread to wait until all data are received will probably not enhance performance. But it will give the user peace of mind that the tool is not wasting time performing needless tasks, instead of fulfilling their request.

Common Java Performance Hazards

There are many common Java performance hazards. Below is a short list of common hazards to avoid.

- Perform calculations once, if possible, and save the results for reuse.
- Any processing done in a loop, which can be pulled out, should be pulled out. All initialization should be done outside a loop whenever possible.
- Exception handling should not be used as a common control structure. It should only be used for exceptions. The overhead of exception handling is high because the call stack has to be rolled back.
- Limit memory allocation and object creation. Reuse objects whenever possible.

Conclusion

MapObjects – Java is a powerful toolbox of GIS tools. Applets or applications built with

MapObjects - Java can be powerful yet robust and efficient. The pursuit of optimum performance is a continuous, but will reward both users and developers.

Author Information

Drew Saunders
Senior Software Engineer
QTSI
Suite 707
1980 N. Atlantic Ave
Cocoa Beach, FL 32931
(321) 799-9655
dsaunders@qtsi.com