Aufgabenblatt 02: Benchmark-Konstruktion

Gruppe: Hendrik Albers, Steffen Giersch, Maximilian Heinrich, Hector Smith, Jeremias Twele

1. Testobjekte für Benchmarks

- Streams verglichen mit foreach Schleifen
- Lambda Ausdrücke verglichen mit anonymen Klassen
- Lambda Ausdrücke verglichen mit Reflection
- Parallele Streams verglichen mit seriellen Streams

2. Testdaten

Wir werden hauptsächlich Referenztypen verwenden (anstatt von primitiven Typen) an denen z.B. auch Methoden aufgerufen werden im Zuge der Bearbeitung durch Streams.
Die vorgeschlagene Textdatei http://www-cs-faculty.stanford.edu/~uno/sgb-words.txt können wir verwenden um daraus Strings zu generieren. Nach kurzer Suche haben wir auch Dateien ähnlicher Art gefunden die wir in Betracht ziehen werden. Es gibt z.B. eine sehr lange Liste von Namen (Vor- und Nachname) von Facebook, Wörterbücher, usw.

3. Kriterien für Benchmarks

Wir messen
- Laufzeit
- Speicherbedarf (Heap)

Messungen von anderen Werten sind entweder schwierig durchzuführen oder von den verwendeten Frameworks nicht unterstützt.

4. Konstruktion und Durchführung von Benchmarks

Wir verwenden das zur Verfügung gestellte Micro Framework von Kent Beck. Die Messung der Laufzeit ist unterstützt. Für die Messung des Speicherbedarfs werden wir es etwas erweitern.

Generelles Vorgehen ist es die Zeit beim Aufruf einer leeren Methode (overhead) zu messen, danach die Methode mit dem Testobjekt aufzurufen, und deren Laufzeit zu messen, und dann die erste Zeit (overhead) davon abzuziehen. Vor dem Methodenaufruf wird der Garbage Collector ausgeführt. Nach Aufruf des Garbage Collectors gibt es eine Wartezeit von einer Sekunde. Über den Systemparameter zur Abschaltung der Class Garbage Collection werden wir uns informieren.

Wir haben einige unterstützende Software betrachtet wollen aber vorerst mit dem Micro Framework beginnen. Möglicherweise werden wir auch jmh einsetzen, für Laufzeitmessungen.
Bei der Verwendung von jmh ist es empfohlen Maven zu benutzen um ein extra Projekt zu erstellen das die .jar Datei des Testobjekts benutzt. Der Aufruf soll dann von der Kommandozeile und nicht aus einer IDE heraus erfolgen. Dies führt zu verlässlicheren Ergebnissen. Dieser Empfehlung würden wir folgen wenn wir jmh einsetzen. Für die Steuerung von jmh werden Annotations im

Testobjekt eingesetzt.

5. Evaluierte Java 8 Profiler

AppDynamics
http://www.appdynamics.com/
Kostenpflichtige Lösung, die weit mehr Funktionen bietet als ein gewöhnlicher Profiler.
Nicht geeignet für unsere Ansprüche, da nicht automatisierbar.

Eclipse Profiler Plugin
http://eclipsecolorer.sourceforge.net/index_profiler.html
Eclipse-Plugin für Profiling-Zwecke.
Nicht geeignet, da nicht automatisierbar und gebunden an Eclipse.

Eclipse Test & Performance Tools Platform Project
http://www.eclipse.org/tptp/
Beinhaltet Funktionalität für Profiling, Testing und Tracing.
Nicht geeignet, da Funktionsumfang zu groß (und komplex).

Eclipse MemoryAnalyzer
http://www.eclipse.org/mat/
Stand-alone Anwendung. Nutzbar für die Analyse von Speicherlecks.
Nicht geeignet, da auch Laufzeit und CPU-Zyklen gemessen werden sollen.

VisualVM
http://visualvm.java.net/download.html
Umfangreiche Open-Source-Anwendung für Profiling, (Remote-)Debugging und Monitoring von Java-Anwendungen. Bietet alle benötigten Features, scheint jedoch ebenfalls nicht automatisierbar zu sein. Außerdem könnte der Funktionsumfang zu umfangreich sein.

OpenJDK jmh
http://openjdk.java.net/projects/code-tools/jmh/
Ausgefeiltes Open-Source-Library/Framework für Micro-Benchmarks. Automatisierbar und leicht nutzbar. Leider lässt es sich nur für Laufzeitmessungen nutzen.