FAKULTÄT FÜR !NFORMATIK

Magisterstudium:
Software Engineering
& Internet Computing

Diplomarbeitspräsentationen der Fakultät für Informatik

Design und Implementierung einer grafischen Komponente für Monitoring von SEDA-Applikationen

Michael Lafite

Technische Universität Wien
Institut für Computersprachen
Arbeitsbereich Programmiersprachen und Übersetzer
Betreuerin: Ao. Univ-Prof. Dipl.-Ing. Dr. eva Kühn

Was ist SEDA?

Einführung

Internetdienste müssen heutzutage in der Lage sein, Anfragen tausender BenutzerInnen gleichzeitig zu verarbeiten. Dabei sollen ständige Verfügbarkeit und stets kurze Antwortzeiten garantiert werden. SEDA beschreibt eine Architektur zur Entwicklung solch hochgradig nebenläufiger Systeme. Diese vereint die Stärken von Thread- und Event-basierter Nebenläufigkeit.

Thread-basierte Nebenläufigkeit

Für jede Anfrage wird ein Thread erzeugt, der für die Verarbeitung der Anfrage zuständig ist. Dieses Konzept ist einfach zu implementieren, hat allerdings den Nachteil, dass die Produktivität ab einer gewissen Anzahl von Threads stark sinkt, da die Threadverwaltung, also das Erzeugen der Threads und das Wechseln zwischen den Threads, Zeit benötigt (siehe Abb. 1).

Event-basierte Nebenläufigkeit

Bei diesem Modell werden Anfragen durch Events repräsentiert. Diese werden zunächst in eine Event Queue gereiht und zu einem späteren Zeitpunkt in mehreren Teilschritten verarbeitet. Indem nach Ausführung eines Teilschrittes mit einem anderen Event fortgefahren wird, lässt sich Nebenläufigkeit bereits mit einem einzelnen Thread realisieren. Die Produktivität bleibt daher auch bei einer sehr hohen Anzahl an gleichzeitig zu bearbeitenden Anfragen stabil (siehe Abb. 1). Die Umsetzung Event-basierter Nebenläufigkeit ist allerdings komplex, u. a. weil die Bearbeitungsschritte so konzipiert werden müssen, dass sie nicht blockieren. Dies erschwert etwa Dateizugriffe.

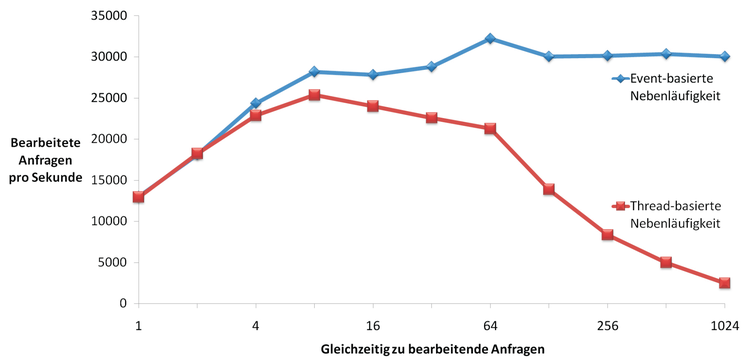

Abb. 1: Vergleich der Produktivität von Thread- und Event-basierter Nebenläufigkeit [1]

SEDA

Das SEDA-Modell vereint die Vorteile der oben angeführten Ansätze zur Realisierung von Nebenläufigkeit. Die Abkürzung steht für ***S**taged **E**vent-**D**riven **A**rchitecture. Event-driven* weist darauf hin, dass SEDA-Applikationen durch Events gesteuert werden. *Staged* deutet an, dass sich diese Applikationen aus mehreren Stages, welche unterschiedliche Aufgaben zu verrichten haben, zusammensetzen. Diese Stages kommunizieren asynchron durch Senden von Events. Jede Stage verfügt über eine Queue, in die empfangene Events eingereiht werden, und einen Thread Pool, dessen Threads für die Abarbeitung der Events zuständig sind. Die Größe des Thread Pools sowie die maximale Länge der Queue können dynamisch und individuell für jede Stage angepasst werden. Dadurch ist es möglich, die verschiedenen Systemkomponenten aufeinander abzustimmen und die Gesamtleistung zu optimieren.
SEDA wurde für hochgradig nebenläufige Systeme konzipiert und zielt auf Skalierbarkeit und Robustheit bei gleichzeitig relativ einfacher Implementierung ab.

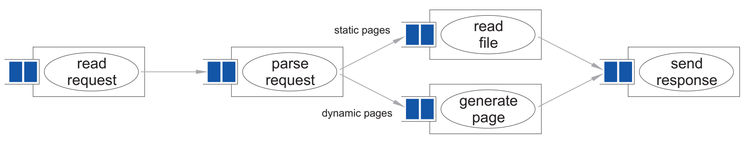

Abb. 2: Stagegraph eines einfachen HTTP-Servers

SedaMon

Ziele

Ziele dieser Arbeit waren die Konzeption und Umsetzung eines grafischen Monitoring-Tools, genannt *SedaMon* (*SEDA Monitor*), das EntwicklerInnen bei Überwachung, Analyse und Optimierung von SEDA-Applikationen unterstützt. Hierfür sollten folgende Hauptfunktionen implementiert werden:

- Visualisierung des Stagegraphen
- Überwachung der Event Queues und Thread Pools
- Anzeige beliebiger Monitoring-Daten
- Möglichkeit, spezielle Monitoring-Komponenten über die SedaMon-GUI hinzuzufügen

Visualisierung des Stagegraphen

SedaMon visualisiert den Stagegraph der überwachten Applikation, wodurch deren Struktur veranschaulicht wird und für jede Stage folgende Informationen abgelesen werden können:

- Name bzw. Funktion
- Aktuelle und maximale Größe des Thread Pools
- Aktuelle und maximale Anzahl von Events in der Event Queue
- Woher empfängt die Stage Events und wohin sendet sie welche?
- Verfügt die Stage über spezielle Subkomponenten? (Transformers, Interceptors)
- Beliebige sonstige Monitoring-Daten (sofern von der überwachten Applikation bereitgestellt)

BenutzerInnen können das Layout des Graphen ihren Wünschen entsprechend anpassen und dieses bei Bedarf auch speichern. Ebenso ist es möglich, den Graph stufenlos zu zoomen.

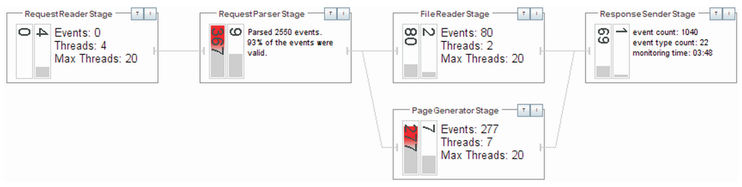

Abb. 3: Visualisierung des Stagegraphen der überwachten Applikation (vergleiche Abb. 2)

Monitoring-Komponenten

Aufgrund des modularen Aufbaus von SEDA-Applikationen bietet es sich an, Monitoring-Code nicht als Teil der Applikationslogik sondern in Form eigener Monitoring-Komponenten zu implementieren. Denkbar ist etwa der Einsatz einer speziellen Monitoring-Stage, um den Eventfluss zwischen zwei Stages zu überwachen und zu analysieren.
Über die SedaMon-GUI ist es möglich, solche Monitoring-Komponenten zur Laufzeit der überwachten Applikation hinzufügen. Dies versetzt EntwicklerInnen in die Lage, bestimmte Komponenten bei Bedarf – etwa im Falle einer Überlastung – gezielt zu überwachen.

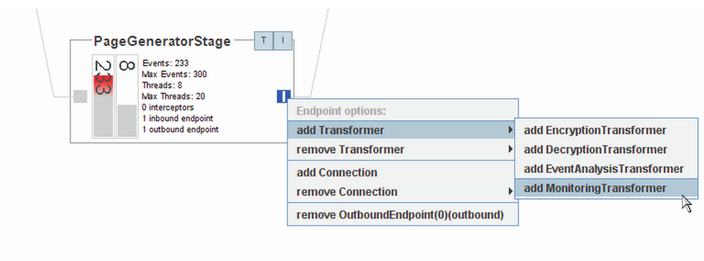

Abb. 4: Hinzufügen einer Monitoring-Komponente

Anzeige der Monitoring-Daten

Um Monitoring-Daten einzusehen, zeigt die BenutzerIn einfach mit dem Mauszeiger auf die gewünschte Komponente. Dies veranlasst SedaMon, die gesammelten Informationen anzuzeigen (unterhalb des Graphens, siehe Abb. 5).
Zusätzlich werden die Monitoring-Daten auch innerhalb der Stages dargestellt. Da hier jedoch weniger Platz für die Anzeige vorhanden ist, können die Informationen optional in verschiedenen Detailgraden zur Verfügung gestellt werden.

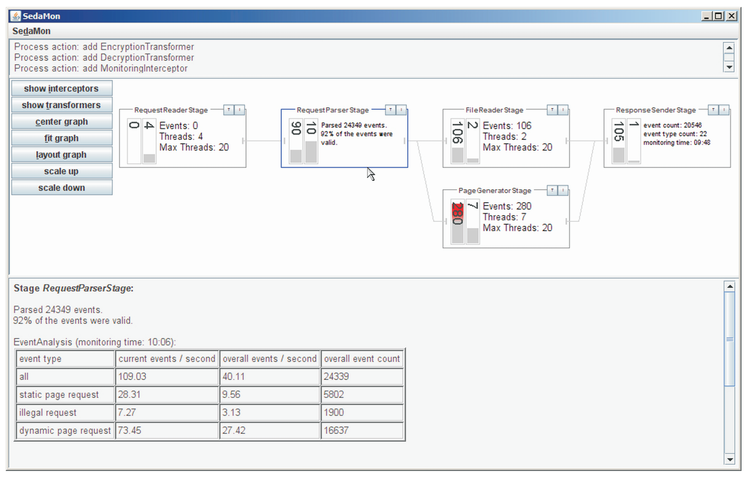

Abb. 5: Monitoring der Stage *RequestParserStage*

Design und Implementierung

SedaMon wurde als eigenständiges Java-Programm konzipiert, welches über eine Schnittstelle auf die zu überwachende Applikation zugreift. Die Kommunikation erfolgt mittels *Remote Method Invocation* (RMI). Ein Remote Interface beschreibt die Methoden, die SedaMon benötigt, um etwa Daten über die Stages auszulesen oder neue Monitoring-Komponenten hinzuzufügen. Die zu überwachende Applikation exportiert ein Remote Object, das dieses Interface implementiert, und stellt auf diese Weise alle erforderlichen Funktionen zur Verfügung.
SedaMon liest alle Informationen regelmäßig aus und aktualisiert die lokalen Daten, falls nötig. Dies gewährleistet eine aktuelle Sicht auf den Stagegraphen bzw. die Monitoring-Daten.
Die grafische Benutzeroberfläche und die Visualisierung des Graphen wurden mittels *Swing* implementiert.

Zusammenfassung und Evaluierung

Mit SedaMon wurde ein Programm entwickelt, das bei der Entwicklung von SEDA-Applikationen unterstützen soll. Alle Ziele wurden erfolgreich umgesetzt. Die Visualisierung des Stagegraphen veranschaulicht den Aufbau der überwachten Applikation und die Interaktion der verschiedenen Komponenten. Durch die Anzeige von Monitoring-Informationen innerhalb der Stages können die wichtigsten Daten auf einen Blick eingesehen werden. Die Überwachung der Event Queues bzw. Thread Pools erleichtert die Optimierung und die Lokalisierung überlasteter Komponenten. Durch den Einsatz von Monitoring-Komponenten ist es möglich, beliebige Vorgänge innerhalb der SEDA-Applikation bei Bedarf zu überwachen und die entsprechenden Daten in der SedaMon-GUI anzeigen zu lassen.
Die größte Schwäche des gewählten Konzepts ist, dass etwas Vorarbeit nötig ist, bevor eine SEDA-Applikation mit SedaMon interagieren kann. So muss zunächst das Remote Interface implementiert werden, welches SedaMon den Zugriff auf die zu überwachende Applikation ermöglicht. Im Gegenzug kann SedaMon jedoch für das Monitoring beliebiger SEDA-Applikationen eingesetzt werden.

Referenzen:
[1] Welsh, Matthew. *An Architecture for Highly Concurrent, Well-Conditioned Internet Services*. Dissertation, University of California, 2002.

Kontakt:
Michael Lafite
Gabelsbergerstraße 5/1/3/8
3100 St. Pölten
Michael.Lafite@gmx.at