

Master-Slave

Zweck

Unabhängige Teilaufgaben innerhalb einer Serviceimplementierung werden in separaten Threads ausgeführt (*divide and conquer*), um nicht-funktionale Anforderungen (i. d. R. Performance) besser zu erfüllen.

Szenario

Ihre Software enthält eine Statusbericht-Komponente, die jederzeit Informationen über den Zustand der verschiedenen Subsysteme, Warteschlangen, Ausführungseinheiten etc. zusammentragen und in Form eines Berichts zurückgeben kann. Dazu werden die verteilt ablaufenden Komponenten nacheinander abgefragt, um deren Statusinformationen anschließend aufzubereiten.

Sie stellen fest, dass die Performance problematisch ist, sobald viele Komponenten deployt auf mehreren Maschinen im Einsatz sind, weil sich die Latenzen für die Statusabfragen summieren. Einige Berechnungen (z. B. Median oder Quantile) können zudem erst durchgeführt werden, nachdem sämtliche Komponenten abgefragt wurden. Um die Anforderungen des Kunden zu erfüllen, müssen Sie die Antwortzeiten der Statusbericht-Komponente senken.

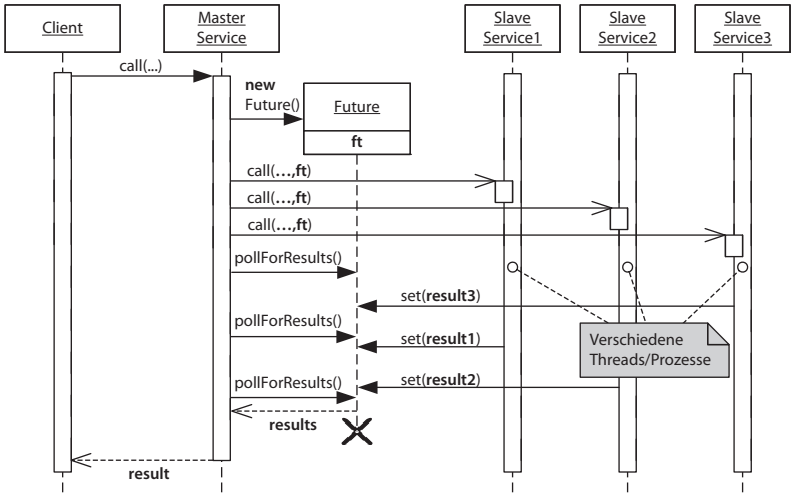
Problem/Kontext

Eine Komponente bietet eine komplexe Operation an, deren Verhalten Sie hinsichtlich nicht-funktionaler Anforderungen wie z. B. Performance, Fehlertoleranz oder Genauigkeit (vgl. [POSA-4]) verbessern möchten.

Lösung

Identifizieren Sie unabhängige Teilschritte und führen Sie diese parallel in unterschiedlichen Threads aus. Im Beispiel sind die Datenabfragen an den verschiedenen Komponenten voneinander unabhängig. Erst bei der anschließenden Auswertung wird ein Zusammenhang hergestellt.

Die Komponente implementiert einen Dienst (*MasterService*), der intern unabhängige Teilaufgaben (*SlaveServices*) *parallel* ausführt und aus den Rückgaben sein eigenes Ergebnis ermittelt. Die Kommunikation



zwischen Master und Slave erfolgt über ein sogenanntes *Future*, ein Objekt, das der Master dem Slave übergibt, damit dieser sein Ergebnis dort später ablegen kann. Der Master fragt das Future regelmäßig ab (*Polling*), bis alle Resultate vorliegen.

Vorteile

Das Muster kann die Performance in einer Umgebung mit mehreren Prozessoren (bzw. Kernen, Maschinen) erheblich steigern, falls sich eine Aufgabe in unabhängig ausführbare Teilaufgaben splitten lässt (vgl. [POSA-4]).

Nachteile

- Die Komplexität der Implementierung erhöht sich.
- Sie müssen auf die Teilergebnisse warten und die Threads sicher synchronisieren.

Varianten/Strategien

- Im Beispiel kümmern sich die SlaveServices selbst um die asynchrone Ausführung (vgl. →Active Object (96)). Alternativ dazu kann der MasterService mehrere Threads für die Aufrufe der Slave-Services verwalten.

Verteilung

- Ein Future ist nur eine Möglichkeit, die Teilergebnisse zu übergeben. Alternativ kann Messaging zum Einsatz kommen oder der Master als \rightarrow Observer (61) der Slaves implementiert werden.
- Statt einer Zerlegung der Aufgabe in Teilschritte können Sie auch eine Partitionierung der Daten vornehmen. In disjunkten Datenbereichen führen mehrere Threads parallel dieselbe Aufgabe aus (z. B. Bildverarbeitung, Suche). Vorhandene Hardware wird besser ausgelastet, was zu einer schnelleren Verarbeitung der Gesamtdatenmenge führt.
- Dieses Muster eignet sich für die Implementierung eines Mehrheitsentscheids (*Voting*). Dabei berechnet man die Lösung eines Problems mit unterschiedlichen Verfahren simultan. Ein Ergebnis gilt als verlässlich, wenn mindestens n von m Ergebnissen übereinstimmen (Motivation: Fehlertoleranz bei mehreren Hardwareknoten oder auch Senkung der Fehlerwahrscheinlichkeit beim Einsatz probabilistischer Algorithmen).
- Sie können ein deterministisches Verfahren mit einem Las-Vegas-Algorithmus kombinieren, der gegebenenfalls erfolglos aufgibt. Findet er (schnell) ein Ergebnis, wird es zurückgegeben; andernfalls wird auf die deterministische (aber in der Regel langsamere) Berechnung gewartet.
- Verschiedene Näherungsverfahren können parallel ausgeführt werden, um durch die Kombination der Einzelergebnisse eine höhere Genauigkeit zu erzielen.

Verweise

[POSA-4].

\rightarrow Combined Method (86)): Die Performance einer solchen Methode kann evtl. durch *Master-Slave* verbessert werden.