Multi-Layer Optimization Strategies for Enhanced Performance in Digital Editions: A Study on Database Queries, Caches, Java EE and JSF

Marco Galster

Universität in Hagen, Deutschland

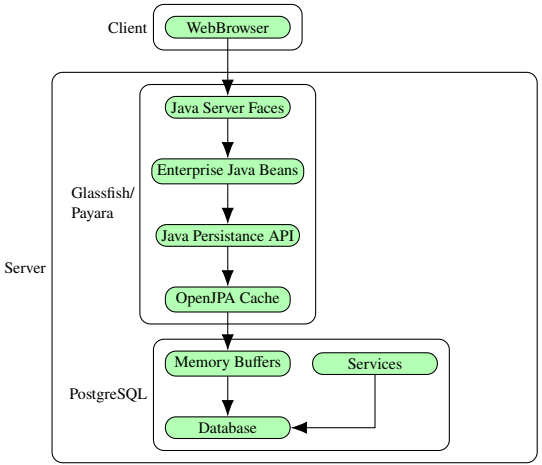
21. August 2024

Übersicht

Problemstellung

- Eine webbasierte Anwendung der Frank Wedekind Briefedition
- Performance Problem bei der Abfrage der Briefe sowie der Recherchen der Korrespondenzen
- entsprechend geringere Akzeptanz der Anwendung

Ablauf einer Web-Anfrage



Vorgehen

- Vor jeder Messung werden die Container neugestartet und die Startroutinen abgewartet
- Aufruf eines Bash-Script auf dem gleichen Rechner wie die Docker-Container um die Latenz des Netzwerkes auszuschließen
- Ermittlung des Speicherbedarfs vor und nach dem Webseitenaufrufen
- Messung der Aufruf der Index-Seite (ohne Datenbankaufrufe) und der Dokumentenliste, jeweils 10 mal
- Report über die SQL-Abfragen mit pgBadger erstellen

Voraussetzungen

- Verwendung von Docker, zur Performance-Limitierung
- Eigene Container für die Datenbank und den Webserver
- Für die Untersuchung wird nur die Dokumentenliste beobachtet

Erste Auffälligkeiten

- OutOfMemory-Ausnahme ausgelöst nach dem vierten Script Aufruf (~40 Webseitenaufrufe)
- Erhöhung des Java-Heapspeichers von 512MB auf 4096MB hat nur die Anzahl der Aufrufe bis zum Absturz verzögert

Ohne Cache

#	Aufrufzeit (ms)			Datenbankabfragen		Speicherverbrauch (MB)		
	min	avg	max	#-avg	avg (ms)	davor	danach	diff
1	360	623	2079	1224.0	30.3	872.8	914.1	41.3
2	331	372	430	1208.0	31.2	914.5	1008.0	93.5
3	291	428	815	1208.0	33.5	1030.0	1297.0	267.0
4	288	357	433	1208.0	33.7	1299.0	1461.0	162.0
5	294	404	499	1208.0	32.9	1462.0	1638.0	176.0

Tabelle 1: Messung ohne Caches

Ohne Cache

- Auffälliger Speicheranstieg, trotz deaktiviertem Cache
- Gleichmässige Anzahl an Datenbankabfragen
- Gleichmässige Laufzeit der Datenbankabfragen
- Laufzeitanteil der Datenbankabfragen unter 10%

	Aufrufzeit (ms)			Datenbankabfragen		Speicherverbrauch (MB)		
#	min	avg	max	#-avg	avg (ms)	davor	danach	diff
1	360	623	2079	1224.0	30.3	872.8	914.1	41.3
2	331	372	430	1208.0	31.2	914.5	1008.0	93.5
3	291	428	815	1208.0	33.5	1030.0	1297.0	267.0
4	288	357	433	1208.0	33.7	1299.0	1461.0	162.0
5	294	404	499	1208.0	32.9	1462.0	1638.0	176.0

Tabelle 1: Messung ohne Caches

Caching mit OpenJPA

#	Aufrufzeit (ms) min	avg	max	Datenbankabfragen #-avg	avg (ms)	Speicherverbrauch (MB) davor	danach	diff
ref-4	288	357	433	1208.0	33.7	1299.0	1461.0	162.0
1	338	567	1853	741.8	28.8	874.8	923.5	48.7
2	235	290	460	685.2	25.8	923.5	926.4	2.9
3	225	254	313	683.6	27.6	927.4	1018.0	90.6
4	235	289	403	683.9	27.6	1018.0	1018.0	0.0
5	193	265	359	687.9	27.6	1025.0	1140.0	115.0
1	151	368	1904	142.2	20.8	878.1	919.9	41.8
2	133	143	159	6.0	20.5	919.8	921.0	1.2
3	120	126	132	6.0	19.9	922.8	924.1	1.3
4	120	124	128	6.0	21.4	924.1	925.4	1.3
5	109	114	131	6.0	19.7	926.1	926.8	0.7

Tabelle 2: Messung mit OpenJPA-Cache und Größe auf 1000 bzw. 10000

Caching mit OpenJPA

- Erwartete Reduzierung der Datenbankabfragen
- Laufzeit in der Datenbank halbiert sich nicht, trotz halbierter Abfragen
- Speicheranstieg wurde reduziert

	Aufrufzeit (ms)			Datenbankabfragen		Speicherverbrauch (MB)		
#	min	avg	max	#-avg	avg (ms)	davor	danach	diff
ref-4	288	357	433	1208.0	33.7	1299.0	1461.0	162.0
1	338	567	1853	741.8	28.8	874.8	923.5	48.7
2	235	290	460	685.2	25.8	923.5	926.4	2.9
3	225	254	313	683.6	27.6	927.4	1018.0	90.6
4	235	289	403	683.9	27.6	1018.0	1018.0	0.0
5	193	265	359	687.9	27.6	1025.0	1140.0	115.0
1	151	368	1904	142.2	20.8	878.1	919.9	41.8
2	133	143	159	6.0	20.5	919.8	921.0	1.2
3	120	126	132	6.0	19.9	922.8	924.1	1.3
4	120	124	128	6.0	21.4	924.1	925.4	1.3
5	109	114	131	6.0	19.7	926.1	926.8	0.7

Tabelle 2: Messung mit OpenJPA-Cache und Größe auf 1000 bzw. 10000

Abfragen über materialized views

#	Aufrufzeit (ms)			Datenbankabfragen		Speicherverbrauch (MB)		
	min	avg	max	#-avg	avg (ms)	davor	danach	diff
ref	288	357	433	1208.0	33.7	1299.0	1461.0	162.0
1	203	315	808	17.8	3.0	851.4	883.9	32.5
2	154	172	187	9.0	2.2	883.2	887.0	3.8
3	145	151	163	9.0	2.8	887.7	895.3	7.6
4	132	143	152	9.0	2.8	896.0	900.0	4.0
5	121	125	132	9.0	2.4	900.6	901.0	0.4

Tabelle 3: Messung mit Materialized View

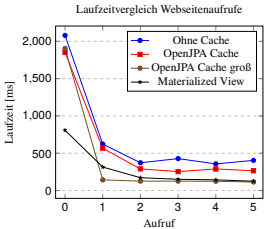
Abfragen über materialized views

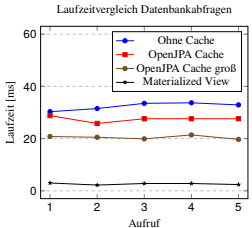
- Deutliche Reduzierung der Datenbankabfragen und Laufzeiten
- Unter-Abfragen werden als Json-Objekte direkt hinterlegt
- Teuer beim erstellen, aber selten notwendig
- Geringe Schwankung der Aufrufzeiten
- Anteil der Datenbank nochmals reduziert

	Aufrufzeit (ms)			Datenbankabfragen		Speicherverbrauch (MB)		
#	min	avg	max	#-avg	avg (ms)	davor	danach	diff
ref	288	357	433	1208.0	33.7	1299.0	1461.0	162.0
1	203	315	808	17.8	3.0	851.4	883.9	32.5
2	154	172	187	9.0	2.2	883.2	887.0	3.8
3	145	151	163	9.0	2.8	887.7	895.3	7.6
4	132	143	152	9.0	2.8	896.0	900.0	4.0
5	121	125	132	9.0	2.4	900.6	901.0	0.4

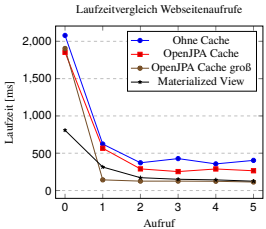
Tabelle 3: Messung mit Materialized View

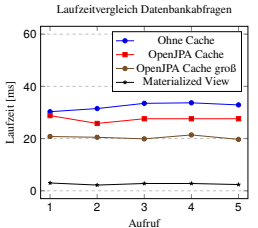
Vergleich


Laufzeitvergleich Webseitenaufrufe
Ohne Cache
OpenJPA Cache
OpenJPA Cache groß
Materialized View
Laufzeit [ms]
2,000
1,500
1,000
500
0
0
1
2
3
4
5
Aufruf


Laufzeitvergleich Datenbankabfragen
Ohne Cache
OpenJPA Cache
OpenJPA Cache groß
Materialized View
Laufzeit [ms]
60
40
20
0
1
2
3
4
5
Aufruf

Vergleich





- Keine Start-Phase auf der Datenbank zu erkennen, im Gegensatz zur Anwendung
- Die Datenbankabfragen nehmen den kleinsten Teil der Laufzeit ein
- Materialized View geringste Datenbankabfragezeiten, aber nicht schneller als OpenJPA-Cache mit größerem Cache

Ausblick

- Größte Optimierungspotenziale sind in der Anwendung vorhanden
- Ermittlung welcher Teil der Anwendung die meiste Zeit benötigt
- Weitere Prüfung der anderen Caches
- Suche des Speicherlecks

Referenzen

[Ibm] 2023. URL: https://www.ibm.com/docs/de/was/8.5.5?topic=applications-configuring-openjpa-caching-improve-performance (besucht am 24. 09. 2023).

[Pos] 2023. URL: https://postgrespro.com/docs/postgresql/14/runtime-config-resource (besucht am 27. 12. 2023).