Skalierbarkeit und Parallelisierung in Java

Herausforderungen und Tücken einer Banken-Handelssoftware

Full-Service-Dienstleistungen für die internationale Finanzwirtschaft

icubic

HighQ IT

Banken-IT

- stetiger Wandel durch Wettbewerb und Regulatorik
- hohe Anforderungen bzgl. Latenz und Verfügbarkeit
- Verarbeitung großer Produktpaletten und einfache Bedienbarkeit
- gekennzeichnet durch verteilte Systeme
- Kommunikation durch Nachrichten, Dateiaustausch, Datenbank-Updates

DZ BANK
Die Initiativbank

Was tut eine Handelssoftware

- verwaltet Finanzinstrumente
  - Aktien, Bonds, Währungen, Deposits, Fonds, Derivate
  - Händler betreut jeweils einen Pool von Instrumenten
- handelt Instrumente an elektronischen Märkten (B2B, B2C)
  - stellt Preise (Quotes, Kauf und/oder Verkauf)
  - sperrt/entsperrt Handel
  - verarbeitet Geschäftsabschlüsse (Verbuchung, Gegengeschäfte)
- berechnet Preise von Derivaten
  - beobachtet und reagiert auf Finanzmärkte
  - zahlreiche Stellschrauben für Händler
- überwacht Preisqualität
  - vergleicht mit Wettbewerbern, sperrt Instrumente mit unplausiblen Preisen

DZ BANK
Die Initiativbank

Anforderungen an eine Handelssoftware

- Automatisierung
  - große Instrumentmengen je Händler
  - leichte Verwaltung und Steuerung aller wichtigen Parameter
- Ausfallsicherheit
  - Hot-Standby oder kurze Startzeiten
- Schutz vor unbeabsichtigtem Handel
  - Sperren aller Quotes beim Abmelden oder Systemausfall
- kurze Reaktionszeiten
  - schnelle Preisanpassungen bei Marktbewegungen
  - schnelle Sperrung von Quotes als Reaktion auf ungewollte Ereignisse
- Reporting
  - Ausgabe wichtiger Informationen an weitere Systeme

DZ BANK
Die Initiativbank

Historie historischer Vergleich

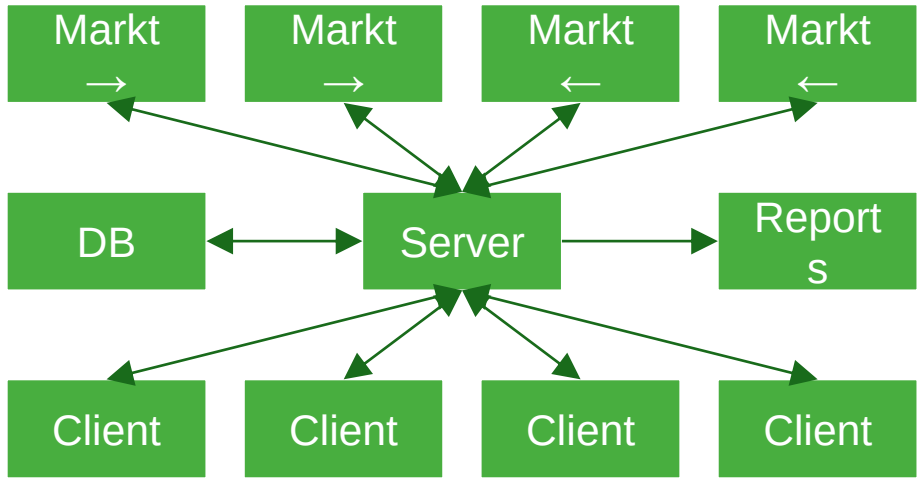
- Automatisierung
  - 2007: etwa 10.000 Instrumente, 70.000 Quotes auf 6 Märkten
  - 2017: etwa 200.000 Instrumente, > 1 Mio Quotes
- Ausfallsicherheit
  - 2007: Startzeit 5min
  - 2017: Startzeit trotz vielfacher Datenmenge kürzer als zu Beginn (< 4min)
- kurze Reaktionszeiten
  - Latenzen verkürzt durch konsequentes Multithreading
- Hardware
  - 2007: 4 Sockel, 16 Kerne, 16 GB RAM
  - 2017: 4 Sockel, 60 Kerne, 128 GB RAM

DZ BANK
Die Initiativbank

Architektur

- Server
  - verwaltet alle Daten
  - exklusiver DB-Zugriff
  - nimmt Marktdaten entgegen
  - berechnet Preise
  - sendet Quotes
- Clients
  - Händler-Frontend
  - wenig eigene Intelligenz

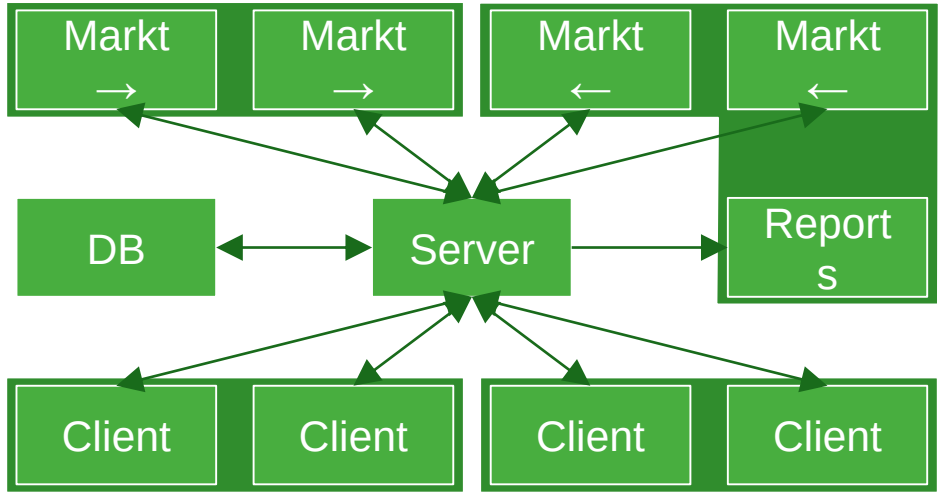
- Marktanbindungen
  - abstrahiert Handelslogik (Geschäftsarten, etc)
  - standardisierte Protokolle zum Server
  - marktabhängige Protokolle zum jeweiligen Markt



DZ BANK
Die Initiativbank

Architektur

- Server
  - eine physische Maschine (4 oder 2 Sockel, viele Kerne)
- Clients
  - 20+ auf mehrere Citrix-Hosts verteilt
- Marktanbindungen
  - können sich physische Maschinen teilen



DZ BANK
Die Initiativbank

Herausforderungen: Multithreading

- Ziele
  - optimale Auslastung, keine unnütze Arbeit
  - Sicherheit: Schutz der Datenstrukturen (Locks, synchronize, copy-on-write, immutable)
  - Kompromiss: Aufwand, Performance, Heap, GC
- Wie viele CPUs?
- Arbeit wie unterteilen? Was sind die Tasks/Jobs?
- Eingabe Verarbeitung Ausgabe
  - Nachrichten empfangen (Schnell annehmen, Puffer für folgende Nachrichten frei halten)
  - Verarbeitung später in Jobs
  - Nachrichten senden (Antworten und Updates, mindestens so schnell senden können, wie Gegenseite abnimmt; Gegenseite sollte schneller abnehmen können, als gesendet wird)

DZ BANK
Die Initiativbank

Job-Arten

- Dauerlast
  - Preise neu berechnen, Quotes senden
- Interaktiv
  - Felder lesen/setzen, persistieren
  - Instrumente erzeugen
  - Instrumente zuweisen
  - Statusänderungen (KO, Verfall, Sperrungen)
  - Plausibilität freigeben
- zeitkritisch vs Batchbetrieb
  - alle Jobs so schnell wie möglich abarbeiten (Feedback am Frontend)
  - Operationen, die sich am Markt auswirken
  - DB-Operationen weniger zeitkritisch (nur dann zu spät, wenn Server vorher abstürzt sich mit einer Fehlermeldung beendet)

Tücken: Frühe Fallen

- Synchronisation
  - “da kann eigentlich nichts passieren”
  - eigene Hardware (2 Threads) vs Zielhardware (16 Threads)
  - Loops über Collections
  - Sichtbarkeit
- GC
  - Default-GC auf Servern: Parallel GC - viel zu lange Pausen
  - Umstieg auf CMS: Full GCs: noch längere Pausen, extensives Tuning des Heap-Verbrauchs

DZ BANK
Die Initiativbank

Tücken: GC

- Heap-Regionen: Generationen
  - neue/junge Objekte, oft kurzlebig
  - alte Objekte, bleiben oft die gesamte Laufzeit erhalten
  - GC versucht, junge Objekte möglichst schnell und effizient zu bereinigen
- GC-Arten in Oracle Java 8
  - Seriell: Stoppt alles, kompaktiert durch Kopieren der Objekte, nur ein Thread, nur für kleine Heaps
  - Parallel Scavenge: parallele Implementation, Default in Java 8
  - Concurrent Mark Sweep: Anwendung nur noch für Young Collection gestoppt, OldGen größtenteils nebenläufig, neigt zu Fragmentierung

DZ BANK
Die Initiativbank

Tücken: GC

- GC-Arten in Oracle Java 8
  - G1: seit 1.7, Heap in Kacheln aufgeteilt, keine starren Generationen mehr, größerer Overhead als CMS (RAM und CPU), bislang keine Verbesserungen der GC-Pausen (in Java 9 wird alles besser)
  - Shenandoah: Open Source, RedHat-Entwicklung, benötigt OS-Unterstützung von Linux, verbesserte Pausenzeiten
  - Pauseless Garbage Collection (C4 Collector): Azul Zing, benötigt OS-Unterstützung von Linux, GC in < 30 ms bei 200 GB Heaps

DZ BANK
Die Initiativbank

Tücken: GC

- Allokationen vermeiden, Objekte wiederverwenden
  - Strings durch StringBuilder ersetzen
- häufige Allokationen finden
- Heap-Verbrauch im Auge behalten
  - String.intern() für langlebige Strings
  - Heap-Verbrauch durch fremden Code: JDK-Klassen, andere Bibliotheken
- Profiler für CPU und RAM
  - Optimierung des Heap-Verbrauchs

DZ BANK
Die Initiativbank

Optimierung

- “Verfrühte Optimierung ist die Ursache allen Übels”
  - nur im historischen Kontext richtig
- Performance-Metrik
  - Durchsatz oder Latenz? Beides!
- Werkzeuge
  - Visual VM: CPU Sampling zeigt Verbrauch der Methoden, erste Hinweise für Optimierungsziele
  - YourKit: CPU Sampling mit Aufrufstack, bessere Verfolgung der Aufrufe, viele andere Daten, komfortabel
  - Oracle Studio: Zugriff auf Hardware Counter, Cache Misses, IPC Werte

DZ BANK
Die Initiativbank

Beispiele: Profiler

Exc CPU Time %	Inc CPU Time %	Exc C/I	Exc Instr %	Exc Cyc %	Exc L3 Miss %	Name
100.00	100.00	1.372	100.00	100.00	100.00	<Total>
15.36	15.49	15.854	1.38	15.98	2.49	GenericTaskQueueSet<Padded<GenericTaskQueue<oopDesc*, (MemoryType)5, (unsigned int)131072>, (unsigned long)64>, (MemoryType)5>::steal_best_of_2(unsigned int, int*, oopDesc*&)
13.22	13.23	10.054	1.86	13.63	10.39	de.icubic.mm.server.instruments.data.Volatilities.getAdjustedVola(double, java.util.concurrent.atomic.AtomicReference)
6.45	9.39	0.385	23.71	6.65	0.02	<static>@0x13f50 (<librendite.so>)
3.87	3.87	0.727	6.98	3.70	0.03	sun.misc.Unsafe.park(boolean, long)
3.23	6.64	0.708	3.53	1.82	0.02	de.icubic.mmkf.MMKF4Java.ImplVolaBn(int, double, double, int, int, double, double, int, long, int, int, int, double, int)
3.07	3.07	0.706	5.63	2.90	0.03	java.lang.Object.wait(long)
1.89	1.89	1.369	1.97	1.96	0.01	__ieee754_exp_avx
1.54	19.91	8.679	0.29	1.83	0.95	ParEvacuateFollowersClosure::do_void()
1.13	1.13	0.747	1.76	0.96	0.01	java.lang.Thread.sleep(long)
1.03	2.36	0.581	2.31	0.98	0.61	InstanceKlass::oop_oop_iterate_nv_m(oopDesc*, FilteringClosure*, MemRegion)

Beispiele: Profiler

Exc CPU Time %	Inc CPU Time %	Exc C/I	Exc Instr %	Exc Cyc %	Exc L3 Miss %	Name
100.00	100.00	1.372	100.00	100.00	100.00	<Total>
0.00	65.15	0.	0.	0.	0.	java.lang.Thread.run()
0.01	58.94	3.194	0.00	0.01	0.01	java.util.concurrent.ThreadPoolExecutor.runWorker(java.util.concurrent.ThreadPoolExecutor$Worker)
0.	58.94	0.	0.	0.	0.	java.util.concurrent.ThreadPoolExecutor$Worker.run()
…	…	…	…	…	…	…
0.00	24.46	1.250	0.00	0.00	0.	GangWorker::loop()
0.00	24.40	0.	0.	0.00	0.00	ParNewGenTask::work(unsigned int)
1.54	19.91	8.679	0.29	1.83	0.95	ParEvacuateFollowersClosure::do_void()
0.00	17.55	2.500	0.00	0.00	0.00	de.icubic.mm.server.pricing.OptionPriceSource.calculatePrice(de.icubic.mm.server.pricing.IWarrant, boolean)
0.02	17.48	3.397	0.01	0.02	0.16	de.icubic.mm.server.pricing.CertificatePriceSource.doCalculate()
0.00	17.40	1.090	0.00	0.00	0.08	de.icubic.mm.server.pricing.CertificatePriceSource.calculateParts(de.icubic.mm.server.instruments.Warrant, java.lang.Iterable)

DZ BANK
Die Initiativbank

Beispiele: Profiler

Exc CPU Time %	Inc CPU Time %	Exc C/I	Exc Instr %	Exc Cyc %	Exc L3 Miss %	Name
100.00	100.00	1.372	100.00	100.00	100.00	<Total>
13.22	13.23	10.054	1.86	13.63	10.39	de.icubic.mm.server.instruments.data.Volatilities.getAdjustedVola(double, java.util.concurrent.atomic.AtomicReference)
0.97	0.97	1.389	0.99	1.00	4.61	java.util.concurrent.locks.AbstractQueuedSynchronizer.compareAndSetState(int, int)
0.45	2.41	5.387	0.11	0.45	2.78	de.icubic.mm.server.pricing.PricingEntity.invalidate(boolean, java.lang.Object)
15.36	15.49	15.854	1.38	15.98	2.49	GenericTaskQueueSet<Padded<GenericTaskQueue<oopDesc*, (MemoryType)5, (unsigned int)131072>, (unsigned long)64>, (MemoryType)5>::steal_best_of_2(unsigned int, int*, oopDesc*&)
0.74	0.74	6.734	0.16	0.77	2.36	java.util.concurrent.locks.AbstractQueuedSynchronizer.getState()
0.80	1.01	3.086	0.36	0.82	2.02	java.util.HashMap.getNode(int, java.lang.Object)
0.50	0.50	4.947	0.14	0.50	1.97	java.util.concurrent.atomic.AtomicReference.get()
0.26	0.53	10.665	0.04	0.28	1.80	de.icubic.mm.server.utils.Visitor.visit(de.icubic.mm.server.utils.COWArrayList, de.icubic.mm.server.utils.Visitor)
0.19	0.19	2.237	0.12	0.20	1.79	java.util.concurrent.atomic.AtomicReference.compareAndSet(java.lang.Object, java.lang.Object)
0.29	0.66	7.231	0.06	0.30	1.72	org.hibernate.collection.internal.PersistentSet.iterator()

DZ BANK
Die Initiativbank

Beispiele: Profiler

```
for ( int j = 0; j < cols; j++) {
        try {
                adjustParam =
priceMovesArray[ j];
        } catch (
ArrayIndexOutOfBoundsException e) {
                adjustParam = 0;
        } catch ( NullPointerException npe) {
                adjustParam = 0;
        }
        final double offset = delta * adjustParam;
        for ( int i = 0; i < rows; i++) {
                result[i][j] = volaArray[i][j] +
offset;
        }
}
```

```
double[] offsets = new double[ priceMovesArray.length];
for ( int j = 0; j < cols; j++) {
        try {
                adjustParam = priceMovesArray[ j];
        } catch ( ArrayIndexOutOfBoundsException e) {
                adjustParam = 0;
        } catch ( NullPointerException npe) {
                adjustParam = 0;
        }
        final double offset = delta * adjustParam;
        offsets[ j] = offset;
}
for ( int i = 0; i < rows; i++) {
        for ( int j = 0; j < cols; j++) {
                result[ i][ j] = volaArray[ i][ j] + offsets[ j];
        }
}
```

DZ BANK
Die Initiativbank

Beispiele: Profiler

Exc CPU Time %	Inc CPU Time %	Exc C/I	Exc Instr %	Exc Cyc %	Exc L3 Miss %	Name
100.00	100.00	1.372	100.00	100.00	100.00	<Total>
…	…	…	…	…	…	…
0.97	0.97	1.389	0.99	1.00	4.61	de.icubic.mm.server.instruments.data.Volatilities.getAdjustedVola(double, java.util.concurrent.atomic.AtomicReference)
…	…	…	…	…	…	…

Exc CPU Time %	Inc CPU Time %	Exc C/I	Exc Instr %	Exc Cyc %	Exc L3 Miss %	Name
100.00	100.00	1.372	100.00	100.00	100.00	<Total>
13.22	13.23	10.054	1.86	13.63	10.39	de.icubic.mm.server.instruments.data.Volatilities.getAdjustedVola(double, java.util.concurrent.atomic.AtomicReference)

DZ BANK
Die Initiativbank

Kontakt

Ralf Helbing

icubic AG

ralf.helbing@icubic.de

René Stäudte

DZ BANK AG

icubic AG
Headquarters
Mittelstraße 10
39114 Magdeburg

icubic AG
Firmensitz München
Bülowstr. 27
81679 München

icubic AG
Firmensitz Frankfurt am Main
Rahmhofstr. 2-4
60313 Frankfurt am Main