Verteilte Systeme

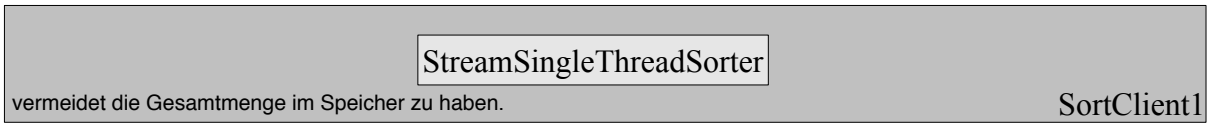
Übung C2 - Gruppenarbeit für 2er Gruppen

Bearbeitungszeit 2 Wochen

Die Übung adressiert die Verteilung von Algorithmen in Clustern und Supercomputern mittels Rekursion über Prozesse und Threads. Zur Demonstration wird ein MergeSort-Algorithmus verwendet und erweitert. Die Messungen müssen auf Mehrkernsystemen ausgeführt werden um sinnvolle Ergebnisse zu liefern, beachtet jedoch dabei ob es sich bei Euren Computern um echte Mehrkernsysteme, oder nur um virtuelle Barrel-Technologie bzw. Hyperthreading handelt! Interessante Quellen sind:

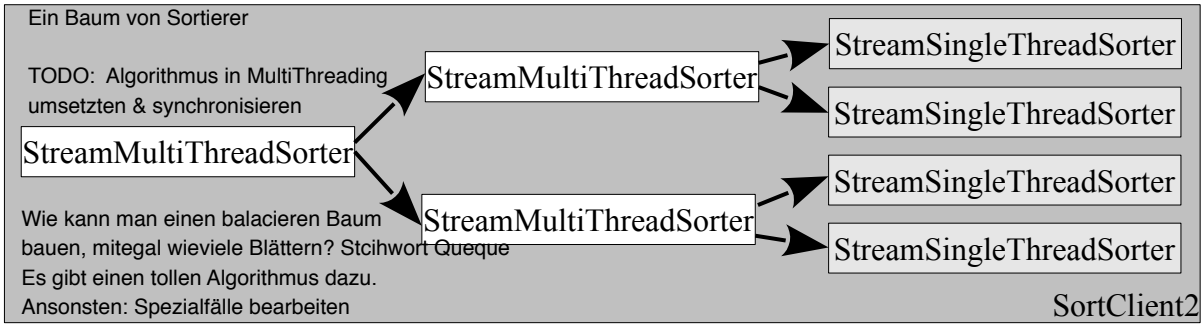
- Merge-Sort Verfahren: http://en.wikipedia.org/wiki/Merge_sort
- EBNF-Syntaxbeschreibungen: http://en.wikipedia.org/wiki/EBNF

Aufgabe 1: Einfache Rekursion



Verwendet die Tool-Klasse *de.htw.ds.sync.FileMultiplier* um die Datei "Goethe - Faust I.txt" nach "Goethe - Faust Ix50.txt" in 50-facher Länge zu replizieren. Bringt die Applikation *SortClient1* zum Laufen und messt aus wie lange es dauert diese Datei nach Wörtern zu sortieren. Macht Euch mit den Klassen *StreamSingleThreadSorter*, *SortClient1* sowie *SortClient* vertraut.

Aufgabe 2: Rekursion über multiple Threads eines Prozesses

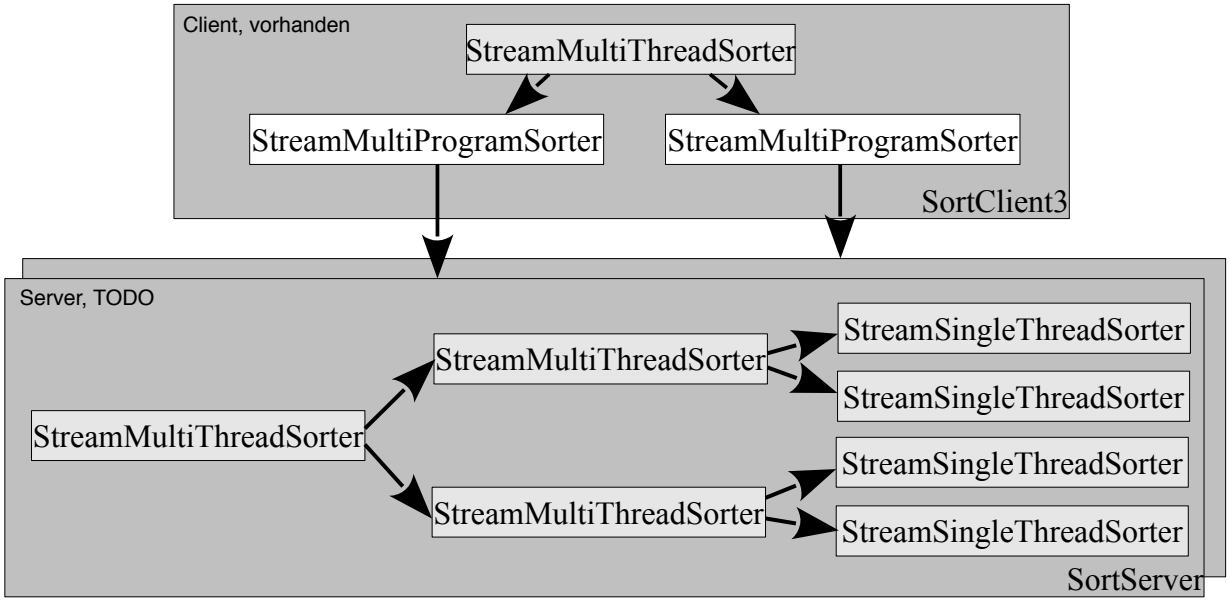


Benennt die Klasse *StreamMultiThreadSorterSkeleton* in *StreamMultiThreadSorter* um. Skaliert die Implementierung wie in Methode sort() unter TODO beschrieben vektorprozessorartig auf zwei Threads. Kopiert die Klasse *SortClient1* nun nach *SortClient2*, und schreibt sie so um dass nun eine *StreamMultiThreadSorter*-Instanz, welche auf zwei *StreamSingleThreadSorter*-Instanzen basiert, als Sortierer verwendet wird. Testet nun diesen Zwischenstand auf korrekte Funktion. Beachtet dass hier keine Rekursion im eigentlichen Sinne stattfindet; die Methoden des Wurzel-Sortierers rufen die Methoden seiner Bestandteile auf, während die Blätter des Sortierer-Baumes dann die eigentliche Sortierung verrichten.

Ein solcher Setup wäre für Zweikern-Systeme optimal, aber dummerweise soll Eure Software für eine beliebige Zahl von Kernen optimiert sein - inklusive ungerade Zahlen! Überlegt Euch wie mit Hilfe einer Instanz von *java.util.ArrayDeque* (implementiert *java.util.Queue*) sehr elegant ein möglichst balancierter Sortierer-Baum wie oben gezeigt aufgebaut werden kann, egal wie viele Kerne im System vorhanden sind. Falls dies nicht gelingt, unterstützt im Code stattdessen die Spezialfälle 1, 2, 4, und 8 Prozessorkerne durch manuell instantiierte Sortierer-Bäume. Verwendet dabei immer den Wurzel-Sortierer (oben links im Bild) zum sortieren in Eurem Client.

Messt aus wie lange es dauert die Datei "Goethe - Faust Ix50.txt" mittels *SortClient2* zu sortieren. Vergleicht die Laufzeit mit Aufgabe 1.

Aufgabe 3: Rekursion über multiple Prozesse und Threads



Analysiert die Klasse *StreamMultiProgramSorter*, vor allem die darin beschriebene EBNF-Protokollsyntax und deren Client-seitige Implementierung. Benennt die Klasse *SortServerSkeleton* nach *SortServer* um, und implementiert an der mit TODO markierten Stelle das Server-Gegenstück zu diesem Sortierer. Passt dabei die Struktur der intern verwendeten *StreamSorter*-Instanz ähnlich zur Aufgabe 2 dynamisch an die Zahl Eurer Prozessor-Kerne an. Beachtet besonders dass die gesamte Konversation zwischen Client und Server (also multiple Anfragen) ähnlich wie bei FTP innerhalb derselben TCP-Verbindung gehandhabt wird! Man spricht hier von einem zustandsbehafteten Protokoll, im Gegensatz z.B. zum zustandslosen HTTP.

Kopiert die Klasse *SortClient1* nach *SortClient3*, und schreibt sie so um dass eine beliebige Zahl von Socket-Adressen externer *Sort-Server* als zusätzliche Parameter übernommen werden – ihr könnt zum Parsen einzelner Socket-Adressen den Ausdruck *„**new** de.htw.ds.SocketAddress(text).toInetSocketAddress()“* verwenden. Realisiert zuerst den Spezialfall eines einzelnen externen Servers, indem ihr im Client eine einzelne *StreamMultiProgramSorter*-Instanz als Sortierer verwendet. Verwendet diesen Setup um die Funktion Eure Implementierung ausgiebig zu testen.

Sobald Eure *SortClient3*-Implementierung fehlerfrei läuft, versucht dort ähnlich wie in Aufgabe 2 die gegebenen Socket-Adressen in einen möglichst balancierten Baum aus *StreamMultiThreadSorter- sowie StreamMultiProgramSorter*-Instanzen umzusetzen. Beachtet dass es auch hier wichtig ist dass die vorgeschalteten Stream-Sorter eines Baumes die Last auf mehrere Threads verteilen, denn andernfalls würden die *StreamMultiProgramSorter* sequentiell statt parallel abgearbeitet, und damit keine zusätzliche Skalierung erreicht!

Startet nun je einen *Sort-Server* auf bis zu 4 separaten Maschinen (i.e. 4 Knoten eines weiten Clusters), und konfiguriert entsprechend auf einer weiteren Maschine eine Instanz von *SortClient3* - für die meisten von Euch sind so viele Computer nur im Übungsraum verfügbar. Messt wie lange es nun dauert die Datei "Goethe - Faust Ix50.txt" mittels dieses Clients zu sortieren. Vergleicht die Laufzeit mit Aufgabe 1 und 2.

Messt zudem mittels Einsatz von *de.htw.ds.sync.FileMultiplier* um welchen Faktor sich die maximal sortierbare Datengröße (unter Beibehaltung der Default-Einstellungen der JVMs) beim Client3 gegenüber den Clients 1+2 erhöhen lässt bevor *OutOfMemory*-Fehler auftreten.