Smarter Connected Logistics™

EPG® | Ehrhardt Partner Group

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen
Campus Jülich

Detecting abnormal backend behaviour

Ausarbeitung Seminararbeit (5. Semester, 95200)

Leon Ikinger

Matr. No.: 3276037

Fachbereich für

Medizintechnik und Technomathematik

Studiengang Angewandte Mathematik und Informatik

Aachen, November 2022

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema **Detecting abnormal backend behaviour** selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Leon Ikinger
Aachen, den 03.01.2023

_____

Unterschrift der Studentin / des Studenten

Diese Arbeit wurde betreut von:

1. Prüfer:	Prof. Dr. rer. nat. Martin Reißel
2. Prüfer und Betreuer:	Dipl. inform. Robert Schulte

Abstract

Die Firma topsystem GmbH entwickelt Software für tragbare Geräte, welche als sprachgesteuerte Assistenten in der Intralogistik dienen, um beispielsweise Lagerarbeiter bei der Kommissionierung zu unterstützen. Diese Geräte kommunizieren über verschiedene Kanäle bzw. Protokolle mit den Lagerverwaltungssystemen der Kunden um dort zum Beispiel neue Aufträge abzurufen oder Lagerbestände zu aktualisieren. Der Inhalt der Anfragen dieser Kommunikation wird (vornehmlich) zwecks einer Vorratsdatenspeicherung zusätzlich an eine hauseigenen Managementsoftware übermittelt, welche dem Benutzer dann verschiedene Statistiken, Informationen und Warnungen basierend auf Auswertungen dieser Daten anzeigt, in vielen Fällen sind aber Auswertungen über die Latenz von Anfragen nicht aussagekräftig ohne den Kontext der Anfrage zu kennen.

Der Ansatz, der in dieser Arbeit zur Lösung des Problems verfolgt wird, besteht im Kern daraus, die Anfragen zu klassifizieren. Für die beiden gängigsten Protokolle kann diese Klassifizierung statisch erfolgen, was für den Rahmen dieser Seminararbeit genügen soll. Die Idee ist es, für jede Anfrage einen Eintrag in einem dokumentenorientierten Datenbanksystem zu hinterlegen, sodass später im Bezug auf die Latenz interessante Quantile errechnet werden können, mit deren Hilfe entschieden werden kann, ob eine Anfrage eines gegebenen Anfragetypens außergewöhnlich schnell oder langsam bearbeitet wurde. Bei der bei dieser Strategie erwarteten Datenmenge ergeben sich unweigerlich Herausforderungen im Hinblick auf die erforderliche Rechenzeit, die benötigt wird, wenn man über alle Datenbankeinträge iteriert, um die oben erwähnten Quantile zu berechnen.

Zur Lösung dieser Herausforderung werden verschiedene Datenstrukturen respektive Algorithmen gegenübergestellt, welche bei Einbußen bezüglich Genauigkeit konstante oder logarithmische Zeitkomplexität ($O(1)$ oder $O(ln(n))$ und Speicherverbrauch versprechen. Außerdem wird auf Grund der unterschiedlichen verwendeten Protokollen zwischen Lagerverwaltungssystemen und mobilen Geräten eine Vorverarbeitung, deren genaue Art auf Grundlage von Mustern ausgewählt wird, von Nöten sein.

Inhalt

1 Einleitung

1.1 Motivation

Bei der Kommunikation zwischen verschiedenen IT-Systemen im Enterprise-Umfeld ist die Latenz ein Gütemaß von entscheidender Bedeutung. Nicht nur wird durch hohe Antwortzeiten das Empfinden der Anwender negativ beeinflusst, sie sind oft direkte Folge von einer rechenintensiven Operation, die Optimierung bedarf.
In jedem Fall ist es für das Unternehmen *topsystem GmbH*, mit dessen Zusammenarbeit diese Arbeit entsteht, interessant die Ursache einer lange andauernden Anfrage zu kennen. Diese kann jedoch nur – ob der Grund Netzwerkprobleme, hohe Systemauslastung oder problematische Parameter sind – sinnvoll untersucht werden, wenn genügend Anfragen samt Metadaten bekannt sind, um das Verhalten zuverlässig zu reproduzieren. Da Anwender eine unter bestimmten Gegebenheiten langsame Anfrage in den wenigsten Fällen so weit eskalieren, dass der Vorfall die Softwareentwicklungsabteilung erreicht, ist es von hohem Interesse solche Anfragen automatisiert zu detektieren. Bislang fehlt dem Endanwender auch eine Möglichkeit zielgerichtet Feedback zu geben und Informationsketten sind im Logistiksektor wegen branchentypischem Rückgriff auf Zeitarbeitskräfte oft unklar.
Im Rahmen dieser Arbeit soll untersucht werden, ob eine eingehende Server-Anfrage mit gegebenem Anfragetyp[1] und der Dauer der Anfrage (Latenz) als statistischer Ausreißer identifiziert werden kann. Die Menge der möglichen Anfragetypen ist in diesem Fall nicht nur gänzlich unbekannt, sondern auch variabel.

1.2 Ausgangssituation

Zwar gibt es in der Managementsoftware, mit der in dieser Arbeit gearbeitet wird, bereits eine Ansicht, in der eine allgemeine Übersicht über die Latenz von Anfragen, allerdings wird die Art und der Kontext der Anfragen nicht berücksichtigt, sodass es schwierig bis unmöglich zu beurteilen ist, ob eine Anfrage wirklich außergewöhnlich lange gedauert hat. Sämtliche Anfragen von den mobilen Geräten an eigene und Systeme Dritter werden bereits zusätzlich mit Metadaten wie dem Status und der Latenz angereichert und an eine Komponente der Managementsoftware von *topsystem GmbH* weitergeleitet, sodass die Datengrundlage für eine erste Erschließung dieses Gebietes bereits gegeben ist und leicht Testdaten erzeugt werden können.

[1] Anfragetyp meint hier eine Kategorisierung einer Anfrage unabhängig von Parametern

1.3 Problematik und Aufgabe

Das Identifizieren von anormalen Server-Anfragen im Bezug auf die Latenz lässt sich trivial auf das Problem des Erkennens von Ausreißern in einer Wahrscheinlichkeitsverteilung reduzieren, was eine häufig wiederkehrende Herausforderung in der angewandten Statistik darstellt [6]. In dem hier vorliegenden Problem, gibt es jedoch keine sinnvolle Wahrscheinlichkeitsverteilung, die man annehmen könnte. Dies liegt in der Natur des Problems, denn die Ausreißer sollen hinsichtlich des Typs der Anfrage aus einer unbekannten Menge von Anfragetypen, welche dementsprechend auch in der Funktionsweise der dahinterstehenden Operation unbekannt sind, gefunden werden.
Eine weitere Herausforderung besteht darin, dass es nicht praktikabel ist, die Daten für jede Anfrage zu speichern oder sogar für die Entscheidung, ob ein Ausreißer vorliegt, über den gesamten Datensatz zu iterieren. Diese Überlegung ist ein zentraler Punkt bei der Wahl des Ansatzes, weil die gängigen Verfahren ausscheiden, da sie eine Kenntnis über den gesamten Datensatz voraussetzen.
Um entsprechend auf dem oben genannten Datenbanksystem arbeiten zu können, muss zunächst ein Modul in die bestehende Managementsoftware integriert werden, das Informationen über alle Anfragen von den tragbaren Geräten zu anderen Systemen aggregiert und geeignet in der Datenbank persistiert. Die dafür nötigen Daten kennt die Managementsoftware bereits, sodass hier keine erheblichen Kosten und Aufwände entstehen.

Der in dieser Arbeit auf Tauglichkeit zu überprüfende Ansatz, um den genannten Anforderungen hinsichtlich Speicher- und Rechenzeitbedarf gerecht zu werden, ist der Einsatz von Datenstrukturen bzw. Algorithmen, welche es erlauben Quantile näherungsweise zu bestimmen, ohne alle Datenpunkte vorrätig zu halten. Da das bereits in die Managementsoftware integrierte Datenbanksystem *Elasticsearch* zwei solcher Funktionalitäten implementiert, T-Digest und HdrHistogram [7, 9, 10], sollen diese beiden in dieser Arbeit gegenübergestellt und auf ihre Eignung im Bezug auf die Problemstellung analysiert werden.
So sollen ohne Kenntnis über die vorliegende Verteilung Aussagen darüber getroffen werden ob es sich bei einer beliebigen Anfrage hinsichtlich der Latenzzeit um einen Ausreißer handelt, indem die Quantile an den Rändern betrachtet werden. Ob diese Grundidee im Bezug auf das illustrierte Problem wirklich sinnig ist, soll anhand einiger empirischer Untersuchungen auf den im Rahmen dieser Arbeit gesammelten Daten erforscht werden.

2 Stand des Wissens

2.1 Theoretische Grundlagen

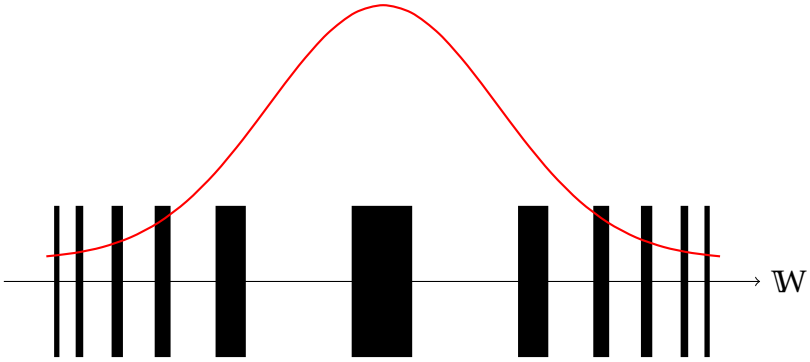
2.1.1 T-Digest

Als T-Digest bezeichnet man eine Datenstruktur, die durch Clusterung[1] von reellen Werten und Festhalten des arithmetischen Mittels und Anzahl an Werten jedes Clusters generiert wird. Diese Clusterung kann dann genutzt werden, um Statistiken, die in Verbindung mit Quantilen stehen, mit besonders hoher Genauigkeit an den Rändern der Verteilung abzuschätzen. Gundsätzlich gibt es zwei Varianten des Algorithmus, der die T-Digest Datenstruktur generiert: Die erste Variante fügt neue Werte einzeln dem nächstgelegenen Cluster hinzu, während die zweite Variante auf Zwischenspeicherung mit anschließendem Sortieren und Zusammenführen setzt. Auch wenn letztere Variante laut Benchmarks [11] und Dunning [7] klar überlegen ist, ist aktuell auf Grund von offenen Fragen hinsichtlich Fehlerfreiheit [14] die erste bei *Elasticsearch* im Einsatz [10]; Deshalb wird in dieser Arbeit nur die Variante ohne Zwischenspeicherung betrachtet.
Im Gegensatz zu etablierten Clustering-Methoden wie *k*-means arbeitet der T-Digest nur auf Werten aus $\mathbb{R}^1$ anstelle von beliebigen metrischen Räumen [8, 12]. Außerdem können die Daten aus verschiedenen Clustern überlappen und die einzelnen Aggregate werden durch einen Centroid[2]-Wert und einem aus der Anzahl an Werten im Aggregat bestimmten Gewichtes beschrieben, anstatt durch die Ober- und Untergrenze des Aggregates [6, 8]. Die Werte werden so aggregiert, dass Cluster, die Quantilen am äußersten Rand entsprechen nur wenige Werte enthalten, wodurch der Fehler beinahe konstant relativ zu $q(q-1)$ ist [7, 8, 9]. Bei vorherigen, vergleichbaren Methoden gibt es einen konstanten absoluten Fehler, unabhängig vom Quantil $q \in [0;1]$ [8].

Wie in Figur 2 zu sehen, ist die grundlegende Struktur des Algorithmus zum Einfügen neuer Daten einfach. Ein neuer Wert x_n mit Gewicht w_n wird einem Centroid c_n, aus der Menge aller Centroids C zugeordnet, indem zunächst eine Untermenge $z \subseteq C$ gebildet wird, in denen nur Centroids mit minimalem Abstand ihres arithmetischen Mittels zu x_n vorkommen. Es wird erneut eine Menge $S \subseteq z$ gebildet, bestehend nur aus Centroids, die nach Hinzufügen von w_n noch ein zulässiges Gesamtgewicht k_n (abhängig von zugehörigem Quantil) haben würden. Sollten noch Centroids in S verbleiben, so wird x_n mit Gewicht w_n dem Centroid mit dem höchsten Gewicht hinzugefügt. Andernfalls wird aus dem Tupel (x_n, w_n) ein neuer Centroid gebildet und der Menge C angehängt [8]. Die letzten Operationen verhindern ungebremstes Wachstum in Abhängigkeit von K, δ und der Mächtigkeit von C, indem der auf Zwischenspeicherung basierte Algorithmus aus Figur 10 aufgerufen wird. Die genaue

[1] Clusterung bedeutet hier Aggregation
[2] Centroid: Schwerpunkt

Funktionsweise ist hier irrelevant, wichtig ist nur, dass durch diesen Algorithmus mit der Skalierungsfunktion aus Figur 4 eventuell Aggregate zusammengefasst werden, wenn der aufzunehmende Wert in ein Intervall fällt, in das nur selten Werte fallen oder der Wert am Rand der Verteilung liegt. Das liegt logisch darin begründet, dass möglichst viele individuelle Cluster in diesen Intervallen liegen sollen, um dort die Präzision zu verbessern. Eine beispielhafte, schematische Visualisierung einer Bildung von Aggregaten mit dem T-Digest kann in Figur 1 betrachtet werden, die Stärke der Striche kodiert in dieser Abbildung das Gewicht eines Clusters. $\mathbb{W}$ sei in dieser Abbildung der Wertebereich, in dem der T-Digest Werte aufnimmt, zusätzlich wurde in Rot der Plot der Normalverteilung angedeutet. Es ist deutlich zu erkennen, dass nach Außen hin mehr Cluster mit weniger Werten vorkommen und sich weiter in der Mitte weniger, mehr Werte fassende Cluster befinden.



Figur 1: *Beispiel T-Digest Aggregate für eine Normalverteilung*

$|\mathcal{C}|_k$ sei für Algorithmus 2 definiert als $|\mathcal{C}|_k = k(q_{right}) - k(q_{left}) \leq 1$ [8] mit

$$q_{\text{left}} = \mathcal{W}_{\text{left}}(\mathcal{C})/n$$
$$q_{\text{right}} = q_{\text{left}} + |\mathcal{C}|/n$$

und [8]

$$\mathcal{W}_{\text{left}}(\mathcal{C}_i) = \sum_{j<i} |C_j| \qquad \text{bzw.} \qquad \mathcal{W}_{\text{right}}(\mathcal{C}_i) = \sum_{j>i} |C_j|$$

Input: Ordered set of weighted centroids $C = \{\}$, sequence of real-valued, weighted points $X = \{(x_1, w_1), \ldots (x_N, w_N)\}$, accuracy tolerance δ, and limit on excess growth K typically in the range $3 \ldots 10$

Output: final set $C = [c_1 \ldots c_m]$ of weighted centroids

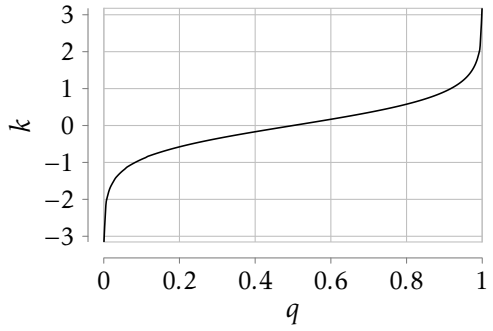
for $(x_n, w_n) \in X$ **:**
  $z = \min |c_i.\mathrm{mean} - x|$;
  $S = \{c_i : |c_i.\mathrm{mean} - x| = z \wedge |c_i + w_1|_k < 1\}$;
  if $|S| > 0$ **:**
    // sort by descending distance
    $S.\mathrm{sort}(key = \lambda(c)\{-c.\mathrm{sum}\})$;
    $c \leftarrow S.\mathrm{first}()$;
    $c.\mathrm{count} \leftarrow c.\mathrm{count} + w_n$;
    $c.\mathrm{mean} \leftarrow c.\mathrm{mean} + (x_n - c.\mathrm{mean})/c.\mathrm{sum}$;
  else:
    $C \leftarrow C + (x_n, w_n)$;
  if $|C| > K\delta$ **:**
    $C \leftarrow \mathrm{merge}(C, \{\})$;
$C \leftarrow \mathrm{merge}(C, \{\})$;
return C

Figur 2: *Construction of a t-Digest by clustering*

(a) *Quelle: Computing extremely accurate quantiles using t-digests [8]*

$$k_2(q) = \frac{\delta}{4 \log n/\delta + 24} \log \frac{q}{1-q}$$

Figur 4: *Elasticsearch relevante Skalierungsfunktion*



Figur 5: *Beispielplot von $k_2(q)$ für $n = \delta = 10$*

(a) *Quelle: Computing extremely accurate quantiles using t-digests [8]*

Man kann also abschließend für das Hinzufügen neuer Werte von einem Kompromiss sprechen, durch den die Laufzeitkomplexität zunimmt und die Präzision an weniger relevanten Stellen der Verteilung abnimmt, dafür aber eine hohe Genauigkeit an den Rändern der Verteilung durch sinnvolle Speicheraufteilung gewährleistet wird.

Abfragen von Quantilen an die Datenstruktur können intuitiver erläutert werden, ohne dass Informationen über den Rechenaufwand und die Speichernutzung relativ zur Genauigkeit verloren gehen. Es sei zunächst angemerkt, dass die von Elasticsearch genutzte Imple-

mentierung der AVLTreeDigest ist [1, 10]. Diese Variante benutzt einen AVL-Baum, sodass das Iterieren über die Aggregate auf das Traversieren eines solchen Baumes, und damit auf eine Binärsuche hinausläuft [1].

1. Zuerst bestimmen wir die Zahl c_q an aufgezeichneten Werten, welche in das Quantil q fallen, indem wir die Gesamtzahl der durch den T-Digest aufgenommenen Werte c multiplizieren und das Ergebnis aufrunden, also $c_q = \lceil q \cdot c \rceil$ [1].

2. Als nächstes werden einige Randfälle, wie bspw. solche, in denen Aggregate nur einzelne Werte enthalten. Für die Zeitkomplexität sind diese nicht relevant.

3. In der Binärbaumtraversion über alle Aggregate wird solange die Anzahl der jeweils aufgenommenen Werte aufsummiert, bis diese Summe den Wert von c_q erreicht oder überschreitet [1].

4. Zwischen den betreffenden Aggregaten wird je nach Gewicht beider Aggregate und um wie viel c_q überschritten wurde linear interpoliert [1].

Der wesentliche Faktor, der die Zeitkomplexität bestimmt, ist also hier die Traversion des Binärbaumes und lässt sich damit durch $O(ln(n))$ beschreiben, wobei mit n die Anzahl der gebildeten Aggregate gemeint ist.

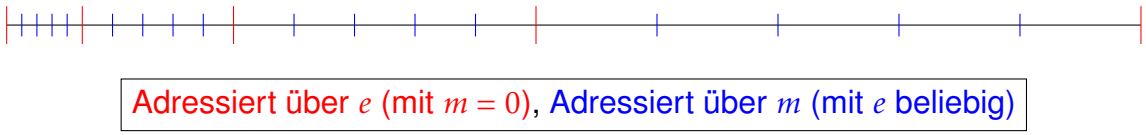
2.1.2 HdrHistogram

Ein HdrHistogram ist ebenfalls eine Datenstruktur, aus der man statistische Daten einer möglicherweise unendlichen Folge von Werten aus $\mathbb{R}^1$ bei konstantem Speicherverbrauch und Dauer des Einfügens neuer Werte, lesen kann [3, 7]. Zum Bilden der einzelnen Aggregate wird jedoch ein eher traditioneller Ansatz verfolgt, denn es gibt jeweils eine feste Ober- und Untergrenze [2, 5]. In der Praxis bedeutet dies, dass Histogramme (sofern kompatibel bspw. bzgl. Auflösung) einfach und genau mittels Vektoraddition zusammengefasst werden können [2, 5, 7]. Eine weitere Auswirkung der festen Aggregatsgrenzen ist, dass der relative Fehler über alle Quantile konstant ist [3].
Um ein HdrHistogram anzulegen benötigen wir x_{min} und x_{max}, welche den kleinsten bzw. größten aufnehmbaren Wert angeben. Dazu kommt der Parameter zur Konfiguration des maximalen Fehlers $s \in \{1 \dots 5\}$, mit welchem Präzision über die Anzahl signifikanter Stellen der Werte innerhalb des Intervalls $x_{min} < x < x_{max}$ kontrolliert wird [2]. Ein Aggregat a_i hat nur ein explizites Attribut, die Anzahl aufgenommener Werte, daher ist ein Aggregat in der offiziellen Implementierung ein Feld vom primitiven Java-Typ *long* [2]. Der Wertebereich, in den ein Wert x in ein Aggregat fällt, wird implizit über die Adressierung der Cluster bestimmt, dies wird vor allem weiter unten deutlich, wo der Aufzeichnungsprozess erläutert wird.
Der Aufbau der Datenstruktur aus den Clustern wird deutlich, wenn wir die Adressierung der

einzelnen Aggregate betrachten: Wir benötigen die zwei Werte e und m, welche *exponentiell* bzw. *linear* adressieren [2]. Figur 7 verdeutlicht dies graphisch, mathematisch können wir die Untergrenze eines jeden Aggregats als $b(e,m) = 2^e(m+O)$ definieren, O sei hierbei definiert als $O = \lfloor log_2(x_{min}) \rfloor$ [2]. Da Elasticsearch die *DoubleHistogram*[3]-Implementierung nutzt, bei der immer $x_{min} = 1$ [2, 10] und damit $O = log_2(1) = 0$ gilt, reicht es für die Zwecke dieser Arbeit anzumerken, dass O eine Art von Offset darstellt, durch welchen die aufnehmbaren Zahlenwerte des ersten adressierbaren Aggregats bei $e = m = 0$ bestimmt werden. Das erste Aggregat $a_0 \in \mathbb{N}$ nimmt (hier mit $O = 0$)

$$\forall x: \quad 2^0 \cdot (0+O) = b(0,0) = \ 0 \ < \ x \ \leq \ 1 \ = b(0,1) = 2^0 \cdot (1+O)$$

auf. Es ist trivial zu erkennen, wie O diesen Wertebereich beeinflusst und so Speicher für nicht vorkommende Werte eingespart werden kann.



Figur 7: *Schematische Visualisierung der Adressierung bzw. Aggregatsaufteilung*

Der Grund für die Wahl dieses Adressierungsschemas wird ebenfalls deutlich, wenn das Aufnehmen eines neuen Wertes erläutert wird. Auffällig ist außerdem, dass sich der relative Fehler beim Erhöhen von e um 1 verdoppelt, weil das Intervall, das durch ein Aggregat repräsentiert wird, sich ebenfalls in der Größe verdoppelt – ein weiterer aus Figur 7 zu entnehmender Zusammenhang. Da der bei der Berechnung eines Quantils q verwendete Wert eines Aggregats immer durch die Obergrenze, von der 1 abgezogen wird, bestimmt ist [2], kann der absolute Fehler Δx durch

$$\begin{aligned} \Delta x &\leq b(e,m+2) - b(e,m+1) - 1 \\ \Delta x &\leq 2^e \cdot (m+2) - 2^e \cdot (m+1) - 1 \\ \Delta x &\leq 2^e - 1 \end{aligned}$$

beschränkt werden.
Wenn wir das größte mögliche e kennen, lässt sich also der absolute Fehler berechnen. Die Anzahl der Aggregate wird in zwei Schritten bestimmt. Sei dafür m_{max} die Anzahl der über m bei festem e adressierbaren Aggregate, und e_{max} invers analog.

1. Um die Genauigkeit von s signifikanten Stellen zu gewährleisten, wird zuerst m_{max} in Abhängigkeit von s bestimmt. Mit $m_{max} = 2^{\lceil log_2(2 \cdot 10^s) \rceil} = 2^{\lceil log_2(10^s) \rceil + 1}$ wird eine Zweier-

[3] Eine Implementierung des HdrHistogram, die double-Werte in long-Feldern speichert

potenz verwendet, sodass mindestens doppelt so viele[4] Aggregate für $e = 0$ existieren wie benötigt [2]. Die Präzision ist hinreichend, da für $e = 0$, also die ersten Aggregate, $\Delta x \leq 2^e = 2^0 = 1 \quad \forall a_i\,, \quad i \in \{1, \ldots, m_{max}\}$ gilt.

2. e_{max} erhalten wir, indem wir in einer Schleife zählen, wie oft m_{max} verdoppelt werden müsste, um den größten aufnehmbaren Wert x_{max} zu erreichen [2].

Der maximale absolute Fehler beträgt dann also $\Delta x = 2^{e_{max}}$, und befindet sich am Ende der Verteilung.
Nun soll dem HdrHistogram ein neuer Wert x hinzufügt werden. Sei dazu $\varphi = m_{max} - 1$ eine Maske, die genutzt wird, um kleine Zahlen Aggregaten mit Indexierung über $e = 0$ zuzuweisen [2].

1. Den Wert $z_b = 64 - O - log_2(m_{max})$ berechnen [2].
2. Die führenden Nullen in der binären Repräsentation von $x|\varphi$ zählen und das Ergebnis von z_b subtrahieren [2]. Das Endergebnis ist bezogen auf das hinzuzufügende x das passende e.

Um den zweiten Index m zu errechnen, wird auf x einen arithmetischen Bitshift nach rechts um $e + O$ ausgeführt [2], dies entspricht dem Ergebnis der Division $\frac{x}{2^{e+O}}$. Der Zähler für die Anzahl an Elementen im so adressierten Aggregat wird um eins erhöht und der Aufzeichnungsprozess für den Wert x ist damit abgeschlossen.
Da für diesen Prozess nur hardwarenahe Operationen wie Bitshift, binären Veroderung und Addition benötigt werden, ist das Aufnehmen neuer Werte in ein HdrHistogram sehr effizient, die Laufzeitkomplexität ist $O(1)$ und der Kompromiss hier besteht darin, dass alle Aggregate fest über den abgedeckten Wertebereich verteilt sind, und kein Speicher an nicht relevanten Stellen eingespart wird.

Das Abfragen eines des Wertes x_q an einem gegebenen Quantil $q \in [0;1]$ ist weniger komplex und kann mit wenigen Schritten erklärt werden.

1. Zuerst bestimmen wir die Zahl c_q an aufgezeichneten Werten, welche in das Quantil q fallen, indem wir die Gesamtzahl der durch das HdrHistogram aufgenommenen Werte c multiplizieren und das Ergebnis aufrunden, also $c_q = \lceil q \cdot c \rceil$ [2].
2. Als nächstes stellen wir sicher, dass c_c mindestens 1 ist, damit im nächsten Schritt mindestens ein aufgezeichneter Wert erreicht wird: $c_q = max(1, q_c)$ [2].
3. In einer Schleife über alle Aggregate wird solange die Anzahl der jeweils aufgenommenen Werte aufsummiert, bis diese Summe den Wert von c_q erreicht oder überschreitet [2].

[4] Laut Kommentaren im Code ist dies durch die erwartete ± 1 Genauigkeit bei Werten über 1000 begründet, ein relativer Fehler von ± 1 für Werte knapp unter 1000 ist demnach nicht akzeptabel.

4. Von dem Aggregat, bei dem wir in der Schleife waren, als der c_q überschritten bzw. erreicht wurde, ermitteln wir ähnlich wie zuvor bei den Überlegungen zum absoluten Fehler die Obergrenze und ziehen von dieser 1 ab [2].

Das Ergebnis ist das gesuchte x_q [2].
Die Laufzeitkomplexität beträgt $O(n)$, wobei $n = e_{max} \cdot m_{max}$ hier die Mächtigkeit der Menge aller Aggregate bezeichnet.

3 Umsetzung

Ziel dieses Kapitels ist es, neben der abschließenden Bewertung über die Eignung der Datenstrukturen und sie – gesetzt den Fall, dass beide geeignet sind – gegenüber zustellen, die Umsetzung der Indexierung neuer Datenpunkte als JVM-Implementierung und beispielhafte Queries[1] an Elasticsearch zu demonstrieren.

3.1 Evaluation der Datenstrukturen

Mit den bisher angestellten Überlegungen bezüglich der Einsatzmöglichkeiten und Funktionsweise der beiden Datenstrukturen und den dahinterstehenden Algorithmen lässt sich bereits sagen, dass beide Optionen den grundsätzlichen Anforderungen gerecht werden, die in Sektion 1.3 aufgeführt wurden. Dass beide im Kern statistische Daten über einen Datensatz liefern können, ohne unbegrenzt in Speicher oder Rechenzeit zu wachsen, haben wir in Kapitel 2 durch direkte Aussagen aus verschiedenen Quellen belegt und konnten diese Aussagen plausibilisieren, indem die Funktionsweise wiedergegeben wurde. Für den T-Digest wird die Anzahl der Aggregate (und damit der Speicher und die Abfragezeit) durch eine Skalierungsfunktion limitiert, und bei einem HdrHistogram ist sie über seine komplette Lebensdauer konstant. Im Folgenden geht es nun also darum, die geeignetere Wahl im Bezug auf die Problemstellung herauszuarbeiten.

3.1.1 Geschwindigkeit und Speicher

Grundsätzlich kann man sagen, dass ein HdrHistogram schneller ist als ein T-Digest. Das dynamische Design der Aggregate kann nicht mit der einfachen Adressierung des HdrHistograms beim Einfügen und Abfragen von Daten mithalten, das können wir aus unseren theoretischen Überlegungen aus Kapitel 2 verallgemeinert schlussfolgern und in Benchmarks der beiden Datenstrukturen ablesen [1, 11]. Bei der Speichernutzung ist der T-Digest im Vorteil, da Aggregate dynamisch gebildet werden, sodass in der Nähe von Häufungspunkten von Datenpunkten und nahe der Mitte des aufzuzeichnenden Wertebereichs nur wenige Aggregate angelegt werden.
Es ist zu bedenken, dass wir die Speichernutzung bzw. Kompression bei beiden Datenstrukturen konfigurieren können und somit nur eine Betrachtung in Relation zu der erwarteten Genauigkeit nahe den für uns interessanten Quantilen sinnvoll ist. Da uns außergewöhnlich lange andauernde Anfragen interessieren, kommt uns die dynamische Aggregatsbildung des T-Digests nur bedingt entgegen. Zwar mag es eine sinnige Annahme sein, dass die Wahrscheinlichkeitsverteilung der Latenzen von Anfragen zum Rand hin abflacht[2], aber

[1] Anfragen
[2] Empirisch für Standardanfragen an die Managementsoftware bestätigt

dennoch wird so potenziell Speicher für Aggregate mit großer Distanz zum Rand verschwendet, wenn Latenzen selten auf gewisse Intervalle tief innerhalb der Verteilung fallen. Dieses Phänomen wird allerdings durch die Skalierungsfunktion k stark abgeschwächt, sodass an den Rändern auch für solche Verteilungen eine hohe Präzision gewährleistet werden kann.

Beide Aspekte sind jedoch für uns nur von geringer Relevanz, da in beiden Fällen Speicher- und Zeitkomplexität konstant sind oder nur logarithmisch wachsen. Insbesondere im Hinblick auf die naive Implementierung dieser Funktionalität, in der alle Daten gespeichert und zum Abfragen der Quantile über sie iteriert wird ($O(n)$), sind beide Möglichkeiten für den in Kapitel 1 skizzierten Anwendungsfall mehr als hinnehmbar. Die beschriebene Managementsoftware läuft ausschließlich auf modernen Systemen, sodass geringfügige Performanz- und Speichereinbußen in Kauf genommen werden können, wenn dadurch die Qualität der Vorhersagen verbessert wird.

3.1.2 Genauigkeit

Da, wie in Abschnitt 3.1.1 deduziert, Unterschiede in Speicher- und Zeitkomplexität für unseren Anwendungsfall zu vernachlässigen sind, können wir unsere Entscheidung alleine auf der Grundlage der Genauigkeit im Verhältnis zu dem benötigten Speicher/Rechenzeit fällen. Die Präzision kann nicht alleine untersucht werden, denn bei beiden Datenstrukturen ist die Kompression konfigurierbar, sodass nur eine Betrachtung der Genauigkeit in für uns interessanten Quantilen relativ zu dafür aufgewendeten Ressourcen sinnvoll ist.
In Sektion 1.3 wurde bereits begründet, warum keine sinnvolle Annahme über die Wahrscheinlichkeitsverteilung der Dauer der Anfragen getroffen werden kann. Dieser Aspekt kommt zum Tragen, wenn man die beiden grundlegenden Strategien der Bildung von Aggregaten vergleicht. Bei einem HdrHistogram nimmt die Genauigkeit mit steigenden zu indexierenden Werten ab, weil die Aggregatsabstände statisch sind und nach oben hin größer werden. HdrHistograms eignen sich aus diesem Grund vor allem für Situationen, in denen die Wahrscheinlichkeitsverteilung und die minimalen und maximalen indexierbaren Werte bekannt sind. Das auszeichnende Merkmal des T-Digests ist, dass die Aggregatsabstände und -gewichte dynamisch sind. Dies ermöglicht eine Aussage über die Genauigkeit nur in Abhängigkeit vom betrachteten Quantil, was im Bezug auf unsere Überlegungen hinsichtlich des besonderen Augenmerks auf außergewöhnlich lange Latenzzeiten in Sektion 1.1 von besonderem Interesse ist.

3.1.3 Ergebnis

Die Wahl der Datenstruktur fällt mit dem in Sektion 3.1.1 und 3.1.2 angestellten direkten Vergleich der beiden Datenstrukturen auf den T-Digest, dessen Anwendung auf das vor-

liegende Problem in der Abwägung zwischen den Kernattributen Geschwindigkeit, Speicher und Genauigkeit, überlegen ist. Eine zwar untergeordnete, aber dennoch erwähnenswerte Entscheidungsgrundlage, stellt die bessere wissenschaftliche Erforschtheit des T-Digests dar. Auch wenn der Einsatz des HdrHistograms bei namhaften Organisationen wie Elastic NV, dem Entwickler des in dieser Arbeit vorgestellten Datenbanksystems Elasticsearch, durchaus eine Vertrauenswürdigkeit suggeriert, fehlen unter anderem Publikationen mit Beweisen für die Genauigkeitsgarantien, die durch Peer-Reviews verifiziert wurden. Insgesamt ist die Zahl der Quellen, die sich mit dem HdrHistogram befassen, gering – auch der Projektverantwortliche hat hier keine wissenschaftlichen Schriften herausgegeben. Der Autor des T-Digests hingegen liefert gleich mehrere Publikationen zu seiner Implementierung und dem theoretischen Hintergrund. Diese Tatsache hat die Wahl noch einmal vereinfacht.

3.2 Implementierung des Managementsoftware-Moduls

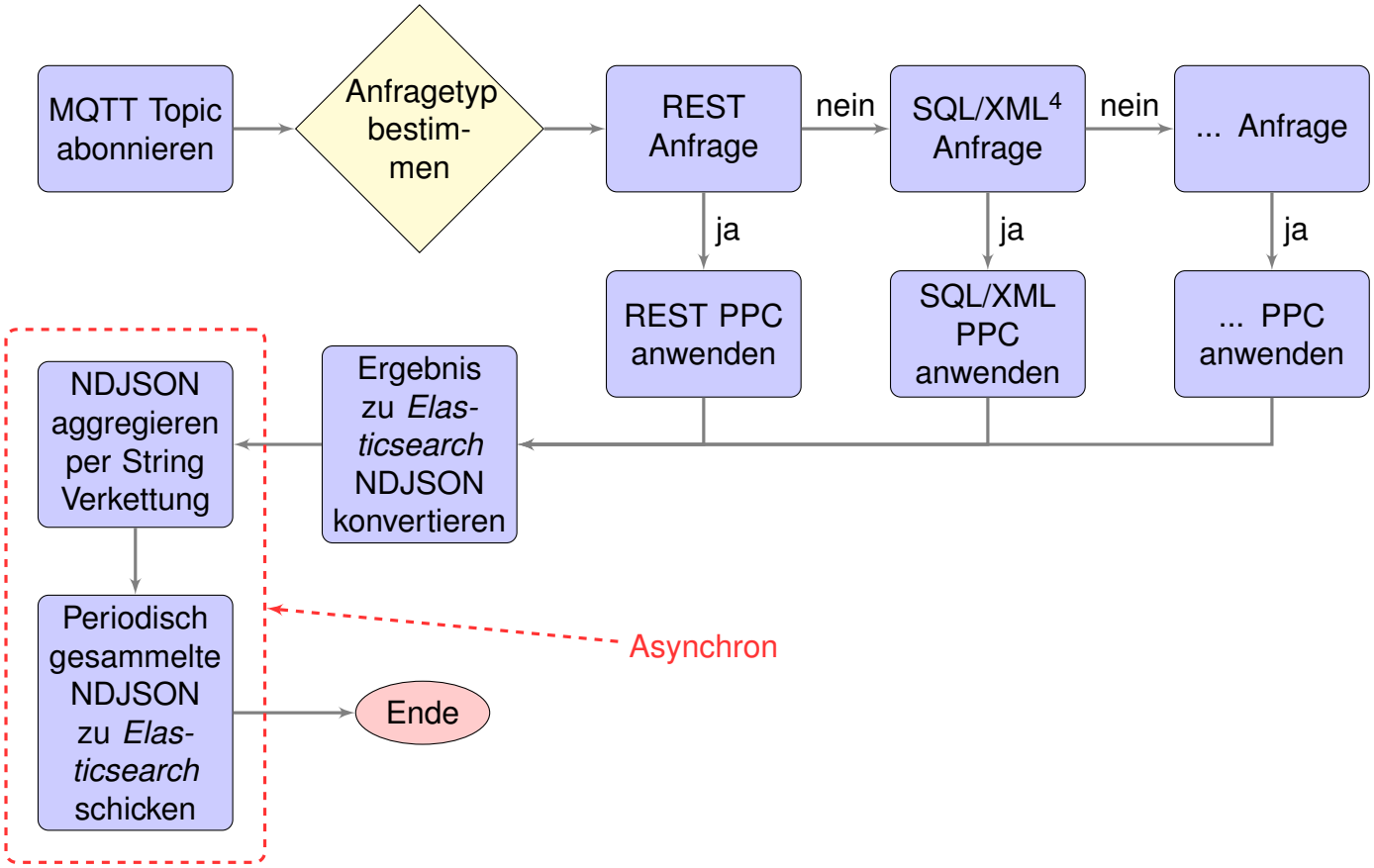
Um die Anfragelatenzen in Elasticsearch-Dokumenten zu speichern, soll ein Modul für die Managementsoftware von *topsystem GmbH* angefertigt werden.
Für die eigentliche JVM-Implementierung, die im Rahmen dieser Arbeit angefertigt werden soll, ist die Wahl der Datenstruktur zunächst nicht relevant. Diese kommt nur beim Abfragen von statistischen Daten zum Tragen, und wird in der nächsten Sektion mit einfachen Datenbankabfragen gezeigt, welche über die REST-Schnittstelle von *Elasticsearch* demonstriert werden können.
Zuerst gilt es aber eine geeignete Struktur für die Dokumente, die in der Datenbank abgelegt werden sollen, festzulegen. Da im Kapitel **Ausblick** noch empirische Untersuchungen zur Weiterentwicklung und potenziellen Verbesserung der Methode angestellt werden sollen, wird hier nicht nur die Anfragetypidentifikation und die gemessene Latenz aufgezeichnet (was für die beschriebenen Datenstrukturen ausreichend wäre, da sie nur auf $\mathbb{R}^1$ arbeiten), sondern noch weitere Features[3] mit aufgenommen.
Die Datengrundlage ermöglicht die folgenden Felder für die Dokumente (hier mit beispielhaften Werten angegeben):

```
{
  "_doc": {
      "id": "cxf/udm/voiceLogin/bulk",
      "size": 18001,
      "timestamp": 1669716026438,
      "duration": 4035
  }
}
```

Zusätzlich zu den beiden in diesem Kapitel benötigten Feldern, `"id"` und `"duration"`, sollen die Payload-Größe (`"size"`) und die Zeit der Anfrage (`"timestamp"`) persistiert wer-

[3] Features sind im Data Science-Umfeld messbare Eigenschaften des zu untersuchenden Phänomens

den.
Der grobe Programmablauf, um dies zu verwirklichen, kann Figur 8 entnommen werden.



Figur 8: *Skizze des Programmablaufs*

Ein essenzieller Bestandteil des Programms ist die Vorverarbeitung durch einen PPC (Preprocessor), durch den die Informationen, die zur Bestückung aller Felder im oben erläuterten Elasticsearch Dokumenten-Layout benötigt werden, extrahiert werden. Die Herausforderung hierbei besteht darin, dass verschiedene Anfragen je nach Kanal unterschiedliche Formen haben. Zum Beispiel kann als `"id"` einer Anfrage an die REST-Schnittstelle die URL verwendet werden, bei der SQL/XML Schnittstelle muss die Identifikation aus dem XML des Anfrage-Payloads gelesen werden.
Bei der Implementierung für die JVM habe ich mich für den Einsatz des Frameworks *Apache Camel* [4] entschieden, mit welchem man verschiedene Fremdsysteme (unter anderem MQTT Broker) an Java Applikationen anbinden kann. Zudem hat man die Möglichkeit große Teile der Nachrichtenverarbeitung deklarativ in XML auszudrücken, was den Ablauf des Programms übersichtlicher und weniger fehleranfällig macht. Es ist jedoch zu jedem Zeitpunkt in der Verarbeitung wieder möglich, die Nachricht in ihrem aktuellen Zustand mit allen währenddessen angehängten Zusatzinformationen durch eine Java-Bean zu leiten, wo dann wieder imperativ gearbeitet werden kann, was für algorithmische Zwischenschritte

[4] SQL/XML meint die Kommunikation mit einem SQL-Server über XML als Query-Sprache

von Vorteil sein kann. Ein Beispiel für solch einen Zwischenschritt ist die bereits erwähnte Vorverarbeitung der Anfragen. Das Framework ist bereits in die Managementsoftware integriert und ist bei *topsystem GmbH* für die Verarbeitung von MQTT Nachrichten in Enterprise-Umgebungen gut erprobt.
Für das weitere Verständnis reichen diese Details der Implementierung, genaueres ist im Anhang unter Quellcode der JVM-Implementierung nachzulesen.

3.3 Anfragen an Elasticsearch

Da die Wahl der Datenstruktur getroffen und die Implementierung des Moduls für die Managementsoftware abgeschlossen ist, können nun die Quantile über entsprechende Queries an Elasticsearch abgefragt werden. Die Daten werden in der JVM Implementierung in dem Elasticsearch Index *wms_requests* gespeichert, an den es nun die Anfrage zu stellen gilt. In der Dokumentation von Elasticsearch ist nachzulesen, dass Anfragen die den T-Digest zur Schätzung von Quantilen über den sogenannte *percentiles aggregation* Queries gestellt werden, die Syntax für eine solche, exemplarische, Anfrage sieht wie folgt aus [9]:

```
{
    "size": 0,
    "aggs": {
        "load_time_outlier": {
            "percentiles": {
                "field": "duration",
                "tdigest": {
                    "compression": 200
                }
            }
        }
    }
}
```

Auffällig ist zunächst das Feld `"size"`, welches auf den Wert `0` gesetzt wird. Der Nutzen besteht darin, dass wir in der Antwort von Elasticsearch nicht zusätzlich zu dem Ergebnis der *percentiles aggregation* Query den Inhalt von (standardmäßig `10`) Elasticsearch Dokumenten auf dem Index *wms_requests* erhalten [10]. Das `"aggs"`-Feld zeigt an, dass es sich um eine Anfrage vom Typ *aggregation* handelt, der Typ der Aggregation seinerseits wird als Schlüssel der `"load_time_outlier"`-Map angegeben, in unserem Fall `"percentiles"`. Semantisch ist `"load_time_outlier"` nur ein frei wählbarer Name für unsere Aggregation, echte Relevanz hat dieser nur in Fällen, in denen mehrere Aggregationen in einer Anfrage vorgenommen werden sollen. Um Elasticsearch mitzuteilen, auf welchem Feld der Dokumente im Index die *percentiles aggregation* durchgeführt werden soll, setzen wir das das Attribut `"field"`, und um T-Digest zu verwenden zusätzlich `"tdigest"`, welches wieder eine Map ist in der wir das Feld `"compression"` auf `200` setzen. Dieser Wert wurde in der

Sektion Theoretische Grundlagen aus Kaptitel 2 mit δ bezeichnet und kontrolliert wie dort beschrieben die Speichernutzung und damit auch indirekt den Rechenaufwand, `200` hat sich hier für den Anwendungsfall als ein guter vorläufiger Kompromiss zwischen Genauigkeit und Performanz herausgestellt.

Um eine Antwort von Elasticsearch zu der oben stehenden Anfrage zu bekommen, muss der JSON Text als Body[5] an die REST-Schnittstelle von Elasticsearch geschickt werden. Der Pfad und die Methode zum Endpunkt der API für die Anfrage an den entsprechenden Index lauten `GET /wms_requests/_search`, der gekürzten Antwort entnehmen wir einige von Elasticsearch vorgegebene Standardquantile, die vollständige Antwort enthält noch einige Metainformationen und ist im Anhang zu finden [13].

```
{
  "took" : 21,
  "aggregations" : {
    "load_time_outlier" : {
      "values" : {
        "1.0" : 25.810000000000002,
        "5.0" : 109.00000000000001,
        "25.0" : 1365.5,
        "50.0" : 3164.0,
        "75.0" : 4763.75,
        "95.0" : 9885.749999999996,
        "99.0" : 11947.209999999985
      }
    }
  }
}
```

Eine weitere Information in dieser gekürzten Fassung dem Feld `"took"` zu entnehmen, und gibt die interne Bearbeitungszeit der Anfrage von Elasticsearch an. Der Wert von 21ms wird im nächsten Kapitel bewertet, an dieser Stelle sei jedoch angemerkt, dass dieser natürlich auch etlichen Mehraufwand beinhaltet, den eine Elasticsearch Anfrage mit sich bringt.

[5] Hier: Payload, Anfrageinhalt

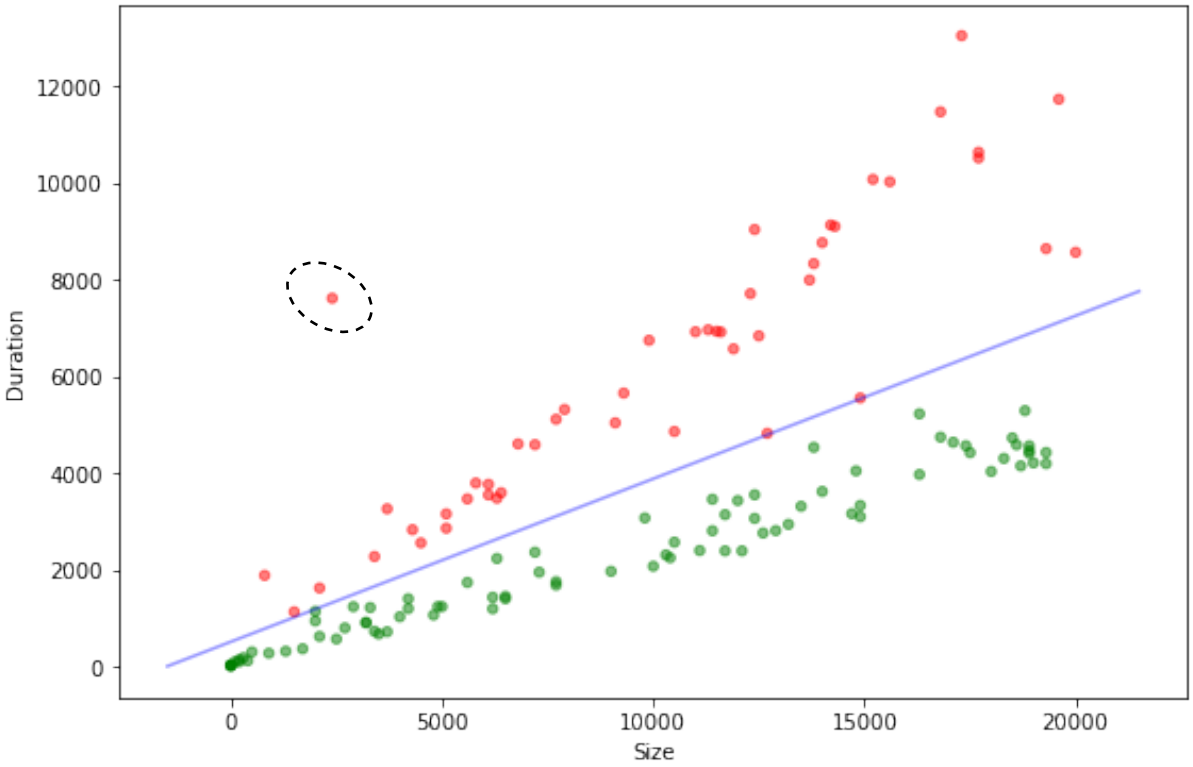
4 Fazit

Die Anfragen an Elasticsearch dauerten, wie dem Feld `"took"` der Antwort zu entnehmen, für 21173 Dokumente etwa 21ms (in 50 Versuchen immer zwischen 13 und 33 ms), für 52401 Dokumente konnte eine Zeit von ca. $19 - 37$ms bei der gleichen Anzahl von Versuchen beobachtet werden. Diese Beobachtung scheint die theoretischen Überlegungen zum logarithmischen Wachstum der Zeitkomplexität zu bewahrheiten, was somit auch die grundsätzliche Tauglichkeit der Methode bestätigt. Es gibt allerdings trotzdem einen schwerwiegenden Kritikpunkt an der Methode.

4.1 Kritik an der Methode

Während der Anfertigung dieser Arbeit kamen verschiedene Bedenken hinsichtlich der Adäquanz des verfolgten Ansatzes auf. Eine der größten Befürchtungen war, dass für viele Anfragentypen ohne weiteren Kontext zusätzlich zur Dauer der Anfrage keine Entscheidung über die Anormalität getroffen werden kann. Bereits in Kapitel **Stand des Wissens** ist im Rahmen der Darlegung beider untersuchter Datenstrukturen erwähnt worden, dass sie jeweils auf $\mathbb{R}^1$ arbeiten, und somit nur ein Feature-Vektor der Dimension 1 unterstützt wird. In Kapitel 3.2, in dem die JVM Implementierung vorgestellt wird, ist beiläufig erwähnt, dass noch weitere Features mit in die Elasticsearch-Dokumente aufgenommen wurden, um eben diese These in diesem Kapitel empirisch zu überprüfen.
Exemplarisch für einen solchen Fall zeigt Figur 9 einen Plot, in dem die Latenz gegen die Payload-Größe eines Anfragetyps aufgetragen worden, der in der Managementsoftware mehrere Benutzer auf einmal anlegt. In blau wurde zusätzlich eine Ausgleichsgrade nach der Methode der kleinsten Quadrate eingezeichnet und die Farbe der eingezeichneten Datenpunkte zeigt an, ob sich ein Punkt unter- oder oberhalb der Gerade befindet. Das Jupyter Notebook, dass diese Grafik generiert hat befindet sich im Anhang als Figur 14. Die Anfrageinhalt (dessen Länge in Figur 9 als ”Size” bezeichnet wird) besteht aus einem JSON String, in dem Informationen über die anzulegenenden Benutzer kodiert sind. Es ist leicht zu erkennen, dass die Payload-Größe proportional zu der Latenz ist, was mutmaßlich[1] an einem höheren Rechenaufwand durch eine größere Anzahl an zu erstellenden Benutzern liegt.
Dass die Betrachtung von 1-dimensionalen Quantilen über die Latenz in diesem und ähnlichen Fällen keinen Sinn macht, wird besonders deutlich wenn der in Figur 9 schwarz eingekreiste Datenpunkt betrachtet wird. Offensichtlich liegt dieser nämlich bezogen auf die Verteilung in der Zeitdimension nicht in einem extremen (extrem soll hier einmal als $q \geq 0,99$ definiert sein) Quantil, relativ zur Länge des Anfrage-Payloads betrachtet kann jedoch trivial abgelesen werden, dass der Wert das Maximum darstellt.

[1] Observationen in der JConsole scheinen diese Annahme zu bestätigen.



Figur 9: *Latenz und Payload-Größe eines Anfragetyps dessen Bearbeitungsdauer augenscheinlich von der Payload-Größe abhängt*

4.2 Abschließende Bewertung

Für eine Vielzahl an Anfragetypen mag die hier untersuchte Methode in ihrer aktuellen Form sinnvoll sein, doch da sie keine allgemeingültige Lösung anbieten kann, ist sie für die in Kapitel 1.3 beschriebene Problematik ungeeignet. Vor allem im Bezug auf die unbekannte Grundgesamtheit der Anfragetypen ist diese Eigenschaft der geschilderten Herangehensweise bedenklich.
Im nächsten Kapitel wird noch ein Vorschlag gelistet, um den Ansatz so zu verbessern, dass die Kernidee der Methode für den Anwendungsfall dennoch brauchbar ist. Die genaue Analyse dieser Möglichkeit geht jedoch über den Rahmen dieser Arbeit hinaus, weswegen dies Teil eines zukünftigen Projekts sein wird.

5 Ausblick

5.1 Automatisiertes Erkennen von anormalen Anfragen

Dass Quantile grundsätzlich über Elasticsearch mit Hilfe der T-Digest Datenstruktur geschätzt werden können, wurde in Kapitel 3.3 praktisch demonstriert. Bei einer produktiv genutzten Implementierung, die automatisch Queries an Elasticsearch stellt und ultimativ Latenzzeiten bewertet, sollte der Kompressionsfaktor konfigurierbar sein, da die Kundensysteme zwar immer Mindestanforderungen genügen, sodass der im Rahmen der empirischen Untersuchungen dieser Arbeit festgelegte Wert von `200` keine Probleme verursacht, jedoch eine höhere Präzision natürlich in jedem Fall wünschenswert ist.
Den zu Beginn der Arbeit festgelegten Ansatz in seiner ursprünglichen Form weiter zu verfolgen ist allerdings wie in Abschnitt 4.1 beschrieben für einige Anfragetypen nicht sinnvoll, auch ist es fraglich, ob Elasticsearch für diesen Zweck überhaupt tauglich ist, oder ob man durch eine eigene Implementierung so viel mehr Flexibilität bei der Wahl von Datenstrukturen und Algorithmen gewinnt, dass sich der Mehraufwand lohnt.

5.2 Weiteres Vorgehen

Zur Verbesserung hinsichtlich der im Kapitel **Fazit** beschriebenen Kritikpunkte bleibt nur die Recherche und Evaluierung weiterer Datenstrukturen, die auch multivariante Daten unterstützen, Elasticsearch weißt keine weiteren solcher Funktionalitäten auf.
Ein alternativer Ansatz dazu wäre es, ein Dimensionsreduktionsverfahren, wie zum Beispiel PCA, einzusetzen und dann Elasticsearch und den T-Digest beizubehalten. PCA benötigt in seiner Standardvariante jedoch eine Kenntnis über den gesamten Datensatz (oder zumindest große Teile), was diese Option für unseren Anwendungsfall unattraktiv macht. Es gibt allerdings auch neuere Varianten des bekannten PCA-Algorithmus, die in ihrer Speichernutzung begrenzt [13] und für Zeitreihen geeignet sind [13].
In Zukunft könnte auch ein Wechsel zu HdrHistogram in Betracht gezogen werden, wenn es darum geht, auch den Benutzern der Managementsoftware einen Überblick über statistische Eckdaten wie den Median oder andere ausgewählte Quantile zu gewähren. Die Genauigkeit ist in solchen Fällen nicht so hoch zu Gewichten wie in der Urspünglichen Fragestellung, da der Fokus nicht mehr auf Ausreißern aus einer Verteilung, sondern mehr einer generellen Übersicht über die Latenz von Anfragen liegt, um eventuell Verbesserungen beispielsweise der eigenen Infrastuktur, Datenbankverbindungen oder WMS Konfiguration vorzunehmen.

Quellen

[1] T-digest quellcode, 2018, accessed 2022-11-30. URL https://github.com/tdunning/t-digest/.

[2] Hdrhistogram quellcode, accessed 2022-12-01. URL https://github.com/HdrHistogram/HdrHistogram.

[3] Hdrhistogram homepage, accessed 2022-12-01. URL http://hdrhistogram.org/.

[4] Apache camel website, accessed 2022-12-06. URL https://camel.apache.org/.

[5] David Andrzejewski. Analysis of hdrhistogram, June 2016, accessed 2022-11-30. URL http://www.david-andrzejewski.com/publications/hdr.pdf.

[6] Bertrand Candelon and Norbert Metiu. A distribution-free test for outliers. Technical Report 02/2013, Frankfurt a. M., 2013.

[7] Ted Dunning. The t-digest: Efficient estimates of distributions. *Software Impacts*, 2021. ISSN 2665-9638.

[8] Ted Dunning and Otmar Ertl. Computing extremely accurate quantiles using t-digests. 2019.

[9] NV Elastic. Elasticsearch documentation - percentiles aggregation, December 2021, accessed 2022-11-29. URL https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-percentile-aggregation.html.

[10] NV Elastic. Elasticsearch source code, accessed 2022-11-30. URL https://github.com/elastic/elasticsearch.

[11] Colin Goodheart-Smithe. Elasticsearch development discussion - uses mergingdigest instead of avldigest in percentiles agg, February 2018, accessed 2022-11-30. URL https://github.com/elastic/elasticsearch/pull/28702.

[12] Mark Last and Abraham Kandel. Automated detection of outliers in real-world data. 03 2002.

[13] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. Memory limited, streaming pca, 2013.

[14] Mark Tozzi. Elasticsearch development discussion - feature/replace avl digest with merging digest, November 2018, accessed 2022-11-30. URL https://github.com/elastic/elasticsearch/pull/35182.

Liste von Abkürzungen

PPC	Preprocessor/Präprozessor
$REST$	Representational State Transfer
AVL	Adelson-Velsky and Landis
SQL	Structured Query Language
XML	Extensible Markup Language
$NDJSON$	Newline-delimited JSON
$JSON$	JavaScript Object Notation
JVM	Java Virtual Machine
URL	Uniform Resource Locator
WMS	Warehouse Management System
API	Application Programming Interface
PCA	Principal Component Analysis

List of Figures

Appendix

Input: Sequence $C = [c_1 \dots c_m]$ a t-digest containing real-valued, weighted centroids with components `sum` and `count` arranged in ascending order by mean, data buffer $X = x_1, \dots x_n$ of real-valued, weighted points, and compression factor δ

Output: New ordered set C' of weighted centroids forming a t-digest

$X \leftarrow \mathtt{sort}(C \cup X)$;
$S = \sum_i x_i.\mathtt{count}$;
$C' = [\,], q_0 = 0$;
$q_{limit} = k^{-1}(k(q_0, \delta) + 1, \delta)$;
$\sigma = x_1$;
for $i \in 2 \dots (m+n)$ **:**
    $q = q_0 + (\sigma.\mathtt{count} + x_i.\mathtt{count})/S$;
    if $q \leq q_{limit}$ **:**
        $\sigma \leftarrow \sigma + x_i$;
    else:
        $C'.\mathrm{append}(\sigma)$;
        $q_0 \leftarrow q_0 + \sigma.\mathtt{count}/S$;
        $q_{limit} \leftarrow k^{-1}(k(q_0, \delta) + 1, \delta)$;
        $\sigma \leftarrow x_i$;
$C'.\mathrm{append}(\sigma)$;
return C'

Figur 10: *Merging new data into a t-digest*

(a) *Quelle: Computing extremely accurate quantiles using t-digests [8]*

https://git.fh-aachen.de/li7509s/codeseminararbeit/

Figur 12: *Quellcode der JVM-Implementierung*


```
{
  "took" : 21,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 21173,
    "max_score" : 0.0,
    "hits" : [ ]
  },
  "aggregations" : {
    "load_time_outlier" : {
      "values" : {
        "1.0" : 25.810000000000002,
        "5.0" : 109.00000000000001,
        "25.0" : 1365.5,
        "50.0" : 3164.0,
        "75.0" : 4763.75,
        "95.0" : 9885.749999999996,
        "99.0" : 11947.209999999985
      }
    }
  }
}
```


Figur 13: *Ungekürzte Antwort auf Elasticsearch Anfrage*

In [422]:

```
from elasticsearch import Elasticsearch
es = Elasticsearch("http://localhost:9200")
```

In [423]:

```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt
```

In [432]:

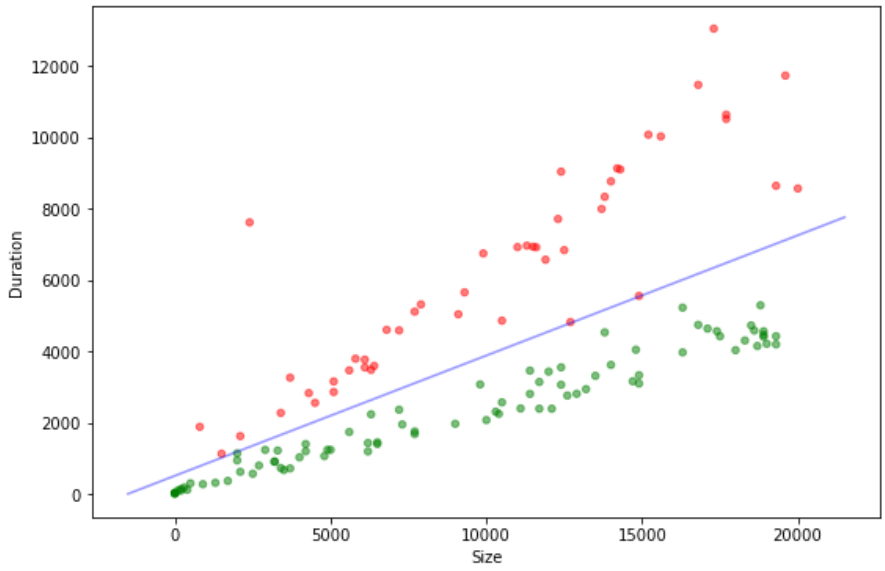
```
es.indices.refresh(index="wms_requests")
query = {
  "query": {"match": {
    "id": "http://localhost:8181/cxf/udm/voiceLogin/bulk?parentSpeechProfile.identifier=D
EFAULT_ENU"}}
}
resp = es.search(index="wms_requests", body=query, size=1000)
print("Got %d Hits" % resp['hits']['total'])

X, Y = map(list, zip(*[(hit['_source']['size'], hit['_source']['duration']) for hit in r
esp['hits']['hits']]))
fig, ax = plt.subplots(figsize = (9, 6))
b, a = np.polyfit(X, Y, deg=1)
color = np.where(Y < a + b * np.array(X), 'g', 'r')
ax.scatter(X, Y, s=20, c=color, alpha=0.5)
print(np.amax(Y))
xseq = np.linspace(-1500, np.amax(X)+1500, num=500)
ax.plot(xseq, a + b * xseq, color="blue", alpha=0.4, lw=1.5)
ax.set_xlabel("Size")
ax.set_ylabel("Duration")
#ax.set_xscale('log')
```

```
Got 133 Hits
13037
```

Out[432]:

```
Text(0, 0.5, 'Duration')
```



Figur 14: *Jupyter Notebook, welches zum erstellen von Figur 9 verwendet wurde*