Prof. Dr. Bernhard Hollunder
Fakultät Informatik
bernhard.hollunder@hs-furtwangen.de

Hochschule Furtwangen
Robert-Gerwig-Platz 1
D-78120 Furtwangen

Wintersemester 2023/24

Streaming Systems

Praktikum

Bearbeitungszeitraum: 18. Oktober 2023 – 24. Januar 2024

Rahmenbedingungen

In dem Praktikum "Streaming Systems" geht es vor allem um das praktische Arbeiten mit den in der Vorlesung vorgestellten Konzepten, Technologien und Frameworks. Neben der praktischen Umsetzung der gestellten Aufgaben sollen Sie Ihre Lösungen auch in schriftlicher Form dokumentieren. Erstellen Sie hierzu ein Dokument, in dem Sie die gewählten Lösungsansätze und erzielten Ergebnisse aufgabenweise darstellen und bewerten. Sie können sich an folgenden Fragen orientieren:

- Welche Lösungs- und Umsetzungsstrategie wurde gewählt?
- Wie schätzen Sie die Leistungsfähigkeit der Lösung ein? Wurden neben der Basisfunktionalität zusätzliche Funktionen realisiert?
- Welche nicht-funktionalen Anforderungen (z.B. Skalierbarkeit und Durchsatz) wurden untersucht? Mit welchem Ergebnis?
- Wie haben Sie sich von der Korrektheit und Vollständigkeit der Lösung überzeugt? Gibt es eine systematische Teststrategie?
- Wie lassen sich die berechneten Ergebnisse auf geeignete Weise darstellen? Gibt es naheliegende Visualisierungsmöglichkeiten?
- Welche zusätzlichen Frameworks und Bibliotheken wurden eingesetzt? Warum?

Aus diesen Fragen leiten sich die Bewertungskriterien ab. Beachten Sie, dass einige Aufgaben bewußt nicht vollständig spezifiziert wurden. Seien Sie kreativ und schließen Sie mögliche Lücken auf sinnvolle Weise.

Die Abschlusspräsentationen aller Lösungen finden gruppenweise nach individueller Absprache ab dem 24. Januar statt. Präsentationen von Zwischenergebnissen erfolgen in den Praktikumsstunden.

Die Aufgaben sollten in Zweierteams bearbeitet werden.

Die Gesamtnote für das Praktikum setzt sich folgendermaßen zusammen: 2/3 lauffähige Lösungen der Aufgaben und 1/3 Dokumentation. Achten Sie auf "Lesbarkeit" der Dokumentation.

Aufgabenübersicht

Die angegebenen Bearbeitungstermine dienen zur zeitlichen Orientierung.

1. Event Sourcing, nativ, 18. Oktober
2. Event Sourcing, nativ, erweiterte Funktionalität, 25. Oktober

3. Event Sourcing, Event Store mit JMS, 8. November
4. Event Sourcing, Event Store mit Apache Kafka, 15. November
5. Datenanalyse "Verkehrsüberwachung" mit Apache Kafka, 22. und 29. November
6. Apache Kafka: Skalierung, 6. Dezember
7. Datenanalyse "Verkehrsüberwachung" mit Apache Beam, 13. Dezember
8. Complex Event Processing "Verkehrsüberwachung" mit EPL und Esper, 20. Dezember und 10. Januar
9. Read-Process-Write-Pattern (optional)
10. Vergleichende Analyse - Lessons Learned, 17. und 24. Januar

Aufgabe 1 [18. Oktober]

Erstellen Sie eine *Event Sourcing* Anwendung, mit der Positions- und Zustandsänderungen von Objekten verwaltet werden können, die das Interface `MovingItem` implementieren:

```
public interface MovingItem {
   String getName();
   int[] getLocation();
   int getNumberOfMoves();
   int getValue();
}
```

Jedes `MovingItem` Objekt hat einen eindeutigen Namen, eine Position in einem n-dimensionalen Koordinatensystem sowie einen Wert vom Typ `int`. Für Objekte in einem 3D-Raum werden die 3 Werte an den ersten 3 Positionen in dem `int`-Array abgelegt. Die Anzahl der bei einem Objekt durchgeführten Positionsänderungen kann über `getNumberOfMoves()` abgefragt werden.

Über die folgenden Befehle können Objekte erzeugt, vernichtet und deren Zustände verändert werden:

```
public interface Commands {
   void createItem(String id);
   void createItem(String id, int[] position, int value);
   void deleteItem(String id);
   void moveItem(String id, int[] vector);
   void changeValue(String id, int newValue);
}
```

Ein neues Objekt wird über `createItem(...)` erzeugt, wobei die initiale Position der Nullpunkt ist und `value` den Wert 0 hat. Alternativ kann die initiale Position und Wert des Objektes angegeben werden. In beiden Fällen muss eine Id vergeben werden. Mit `moveItem(...)` wird der Bewegungsvektor angegeben. Das Argument [2, -4, 7] beispielsweise bedeutet, dass die aktuelle x-Koordinate um 2 Einheiten erhöht wird, die y-Koordinate um den Wert 4 verringert wird und die z-Koordinate sich um 7 Einheiten erhöht. `changeValue(...)` ändert den Wert eines Objektes.

Über eine Query-Schnittstelle können u.a. die aktuellen Positionen der Objekte abgefragt werden:

```
public interface Query {
   public MovingItemDTO getMovingItemByName(String name);
   public Enumeration<MovingItemDTO> getMovingItems();
   public Enumeration<MovingItemDTO> getMovingItemsAtPosition(int[] position);
}
```

Beachten Sie, dass für die Rückgabe von Daten auf der Query-Seite der eigenständige Typ `MovingItemDTO` (anstatt `MovingItem`) verwendet wird. Dies scheint unnötig zu sein, erzeugt jedoch eine starke Entkopplung der *Write Side* von der *Read Side*. Definieren Sie einen geeigneten Typ `MovingItemDTO`.

Erstellen Sie – basierend auf diesen Schnittstellen-Vorgaben – eine *Event Sourcing* Anwendung gemäß der Komponenten der Grobarchitektur von Folie "CQRS with Event Sourcing". Hierzu können Sie wie folgt vorgehen:

- Erstellen Sie eine Implementierungsklasse für `MovingItem`.
- Erstellen Sie eine Implementierungsklasse für `Commands` bzw. einen *Command Handler*. Sie benötigen ein Domänenmodell, um Validierungen durchführen zu können. Soll beispielsweise ein weiteres Objekt mit einem bereits vergebenen Namen angelegt werden, so wird dieses Kommando nicht ausgeführt.
- Einfachheitshalber soll in dieser Aufgabe das Domänenmodell nicht auf Basis der im *Event Store* abgelegten Ereignisse aufgebaut werden. Stattdessen kann eine Map zur Verwaltung der aktuell vergebenen Objektnamen verwendet werden, weitere Informationen werden im Domänenmodell (zunächst) nicht benötigt. Wird ein Kommando akzeptiert, werden die erforderlichen Ereignisse erzeugt. Ein Beispiel für ein Ereignistyp ist etwa `MovingItemCreatedEvent`. Im einfachsten Fall wird ein Kommando auf ein entsprechendes Ereignis abgebildet. Je nach Kommando muss auch das Domänenmodell aktualisiert werden.
- Die erzeugten Ereignisse werden im *Event Store* abgelegt. Verwenden Sie hierzu eine Datenstruktur wie etwa eine `BlockingQueue`. Hinweis: In späteren Aufgaben kommen an dieser Stelle verteilte Messaging Systeme zum Einsatz. Sie sollten den Zugriff auf den *Event Store* kapseln.
- Implementieren Sie die `Query` Schnittstelle. Hierzu benötigen Sie einen *Query Handler* als eigenständige Komponente. Überlegen Sie sich eine geeignete Datenstruktur für das zugrundeliegende *Query Model* (z.B. eine Map), um Anfragen auf einfache Weise beantworten zu können.
- Nun benötigen Sie noch eine Projektion, die die im *Event Store* eingegangenen Ereignisse auswertet und das *Query Model* aktualisiert.

Erstellen Sie eine Client-Anwendung, die `MovingItem` Objekte erzeugt und deren Positionen und Werte ändert. Hierzu bietet es sich an, u.a. mit einem Zufallszahlengenerator zu arbeiten, der für die erstellten Objekte Bewegungsvektoren erzeugt und diese über den `moveItem(...)` Befehl anwendet.

Achten Sie auf eine strikte Trennung zwischen dem Domänenmodell und dem *Query Model*, sodass beide Komponenten unabhängig voneinander weiterentwickelt werden können. Überlegen Sie sich, wie Sie auf sinnvolle Weise die Korrektheit und Vollständigkeit Ihrer Lösung prüfen können. Setzen Sie nachfolgend das Testkonzept um.

Prof. Dr. Bernhard Hollunder
Fakultät Informatik
bernhard.hollunder@hs-furtwangen.de

Hochschule Furtwangen
Robert-Gerwig-Platz 1
D-78120 Furtwangen

Aufgabe 2 [25. Oktober]

Erweitern Sie Ihre Lösung von Aufgabe 1 dahingehend, dass folgendes Verhalten umgesetzt wird. Nach einer bestimmten Anzahl von Bewegungen (z.B. 20) eines Objektes soll dieses entfernt werden. Wird also das zwanzigste Mal der `moveItem(...)` Befehl für ein Objekt angewendet, wird diese Bewegung nicht ausgeführt. Stattdessen wird das Objekt gelöscht, der Name des Objektes wird freigegeben und kann nachfolgend wieder vergeben werden. Überlegen Sie sich, wie das Domänenmodell auf einfache Weise erweitert werden kann, um diese Funktionalität umzusetzen.

Betrachten Sie nun folgende Erweiterung. Bei der Durchführung von `moveItem(...)` mit einem Argument ungleich dem Nullvektor bei einem Objekt soll überprüft werden, ob sich auf der neuen Position bereits ein Objekt befindet. In diesem Fall soll letzteres entfernt werden. Wie kann diese Funktionalität umgesetzt werden? Ist es sinnvoller das Domänenmodell geeignet anzupassen oder sollte eher eine erweiterte Query-Schnittstelle genutzt werden? Prüfen Sie die Möglichkeiten und setzen Sie eine Variante um.

Aufgabe 3 [8. November]

Modifizieren Sie die Lösung von Aufgabe 2 dahingehend, sodass ein JMS Messaging System wie etwa Apache ActiveMQ zur Verwaltung der Ereignisse genutzt wird. Der *Command Handler* nimmt hierbei die Rolle des Nachrichtenproduzenten ein; die Projektion konsumiert die erzeugten JMS-Ereignisse und aktualisiert das *Query Model*.

Ermitteln Sie auch die durchschnittliche Zeit für den Transport eines Ereignisses vom Produzenten zum Konsumenten.

Aufgabe 4 [15. November]

Setzen Sie nun Apache Kafka zur (persistenten) Speicherung der Ereignisse im *Event Store* ein.

Des Weiteren soll nun das Domänenmodell auf Basis der im *Event Store* gespeicherten Ereignisse dynamisch bei der Abarbeitung eines Befehls aufgebaut werden. Realisieren Sie entsprechende *Event Store*-Abfragen wie etwa `loadNamesOfMovingItems()` als Apache Kafka Konsumenten. Dies bedeutet auch, dass die in Aufgabe 1 - 3 verwendete Map zur Speicherung der Objektnamen nicht mehr benötigt wird.

Aufgabe 5 [22. und 29. November]

In den folgenden Aufgaben soll mit Hilfe unterschiedlicher Programmiermodelle die Verarbeitung kontinuierlicher Datenströme untersucht werden. Hierzu wird ein Szenario aus der Domäne "Verkehrsüberwachung" betrachtet. Zum einen sind die zu lösenden Aufgaben in diesem Kontext schnell zu verstehen, ohne dass tief gehendes Domänenwissen aufgebaut werden muss. Zum anderen sind die durchzuführenden Datenanalysen und -aufbereitungen repräsentativ für eine Vielzahl weiterer Fachdomänen.

Entwickeln Sie zunächst einen Testdatengenerator, der Datensätze mit folgender Struktur erzeugt: `timestamp id val-0, val-1, val-2, ..., val-n`.

Beispiel:

```
2023-09-23T10:01:29.551Z 2 20.4,33.5,40.0
2023-09-23T10:01:40.895Z 1 35.6,-17.4,28.6
2023-09-23T10:01:54.759Z 2 42.1,34.7
2023-09-23T10:02:10.452Z 1
2023-09-23T10:02:20.596Z 3 28.4,21.1
...
```

Eine Zeile soll folgendermaßen interpretiert werden: Der führende Zeitstempel gibt den Zeitpunkt der Datenerzeugung an. Die folgende Zahl legt den eindeutigen Namen eines Sensors fest. Die nachfolgenden Zahlen sind die zu einem Zeitpunkt gemessenen Geschwindigkeitswerte des Sensors in der Einheit Meter pro Sekunde (m/s). Die Anzahl der Sensoren auf einem Streckenabschnitt sowie die Anzahl der zu einem Zeitpunkt zu erzeugenden Geschwindigkeitswerte sowie die minimale und maximale Geschwindigkeit soll konfigurierbar sein. Ebenso kann die Taktung, also der zeitliche Abstand zwischen zwei erzeugten Datensätzen, durch zwei `int` Zahlen `m1` und `m2` eingestellt werden. Dies bedeutet, dass der nächste Datensatz mindestens `m1` und höchstens `m2` Millisekunden nach dem vorhergehenden erzeugt wird. Es sollen gelegentlich auch negative Geschwindigkeitswerte produziert werden. Die Datensätze sollen in einen Apache Kafka Topic mit genau einer Partition eingestellt werden.

Die Aufgabe besteht darin, die Durchschnittsgeschwindigkeit in der Einheit km/h für jeden Sensor in Zeitfenstern vorgegebener Länge (z.B. im 30 Sekundentakt) zu ermitteln. Implementieren Sie hierzu eine Anwendung, die die Datensätze von dem Topic konsumiert und eine entsprechende Datenaufbereitung durchführt. Die berechneten Werte können nun genutzt werden, um

1. den zeitlichen Verlauf der Durchschnittsgeschwindigkeit an einer Messstelle (d.h. eines Sensors) sowie
2. die Durchschnittsgeschwindigkeiten auf einem Streckenabschnitt (definiert durch eine Folge von Sensoren) zu einem bestimmten Zeitpunkt

zu ermitteln.

Ein Datensatz ohne Geschwindigkeitswert (vgl. Zeile 4 im obigen Beispiel) soll nicht berücksichtigt werden. Eine negative Geschwindigkeit (Messfehler) soll ebenso von der weiteren Verarbeitung ausgeschlossen werden.

Erstellen Sie eine Anwendung, die die Datensätze aus dem Kafka Topic konsumiert und die geforderten Berechnungen durchführt. Konzipieren Sie eine geeignete Programmstruktur und realisieren Sie diese (vorzugsweise in Java).

Überlegen Sie sich eine geeignete intuitive textbasierte Ausgabe der berechneten Ergebnisse und implementieren Sie diese. Ergänzend können Sie eine graphische Darstellung (beispielsweise mit Grafana) realisieren. Überlegen Sie sich, wie Sie die Korrektheit Ihrer Lösung prüfen können.

Aufgabe 6 [6. Dezember]

Der Testdatengenerator von Aufgabe 5 soll die erzeugten Datensätze in mehrere Partitionen des Topics einstellen. Arbeitet Ihr Auswertungsprogramm immer noch korrekt? Falls nicht, modifizieren Sie Ihre Lösung entsprechend und prüfen Sie die Korrektheit. Gibt es signifikante Performanzunterschiede zwischen den Lösungen von Aufgabe 5 und 6?

Aufgabe 7 [13. Dezember]

Die Aufgabenstellung von Aufgabe 5 soll nun mit dem Pipeline-/Transforms-Programmiermodell umgesetzt werden. Nutzen Sie hierzu Apache Beam. Eine Apache Beam Pipeline soll die Datensätze über den KafkaIO-Adapter einlesen und nachfolgend aufbereiten, bereinigen sowie aggregieren.

Hinweise zum Vorgehen:

- Definieren Sie geeignete `PCollection`s und `Transform`s zur Ermittlung der geforderten Ausgabe.
- Es bietet sich an, mit `Transform`s wie etwa `GroupByKey`, `Combine` und `Window` zu arbeiten.

- Der Typ `IntervalWindow` hat die Methoden `start()` und `end()` zur Abfrage des Start- und Endzeitpunkts.

Vergleichen Sie die berechneten Werte mit den Ergebnissen von den vorhergehenden Aufgaben.

Aufgabe 8 [20. Dezember und 10. Januar]

Die Kernfunktionalität von Aufgabe 5 soll nun mittels CEP umgesetzt werden. Ergänzend sollen Sie "komplexe" Ereignisse identifizieren, definieren und umsetzen. Verwenden Sie hierzu das System Esper und EPL.

Erstellen Sie entsprechende EPL-Anfragen zur Datenbereinigung und Berechnung der Durchschnittsgeschwindigkeit.

Nachdem die Durchschnittsgeschwindigkeit für die Sensoren pro Zeitfenster berechnet wurden, sollen diese jeweils als (aggregiertes) Ereignis ist das System eingestellt werden. Definieren Sie hierzu einen entsprechenden Ereignistyp und verwenden Sie die `insert into` Klausel, um Instanzen von diesem Typ zu erzeugen.

Diese erzeugten Ereignisse sollen nun wie folgt weiter verarbeitet werden: Kommt es innerhalb einer bestimmten Zeit (z.B. 30 Sekunden) zu einem starken Abfall der Durchschnittsgeschwindigkeit, soll ein weiteres Ereignis erzeugt werden, welches auf eine mögliche Stauentwicklung hinweist. Definieren Sie hierzu einen weiteren Ereignistyp.

Überlegen Sie sich geeignete Testfälle, wie Sie die Korrektheit Ihrer EPL-Anfragen testen können.

Aufgabe 9 (optional)

Implementieren Sie das Read-Process-Write-Pattern mit Apache Kafka. Betrachten Sie insbesondere Fehlersituationen, die zeigen, dass alle Datensätze tatsächlich genau einmal verarbeitet werden.

Aufgabe 10 [17. und 24. Januar]

In den bisherigen Praktikumsaufgaben wurden vorgegebene Aufgabenstellungen mit unterschiedlichen Technologien und Frameworks bearbeitet. Das Ergebnis war jeweils eine lauffähige Implementierung. Hierbei haben Sie das jeweilige Programmiermodell angewendet und die Arbeitsweise bzw. Fähigkeiten der Technologien praktisch erproben können.

In dieser abschließenden Aufgabe sollen Sie ein noch tieferes Verständnis bezüglich der in der Vorlesung eingeführten Konzepte, Technologien, Frameworks und Programmiermodelle erwerben.

Konkret sollen Sie folgende Systeme

- Java Messaging System (JMS)
- Apache Kafka
- Apache Beam
- Complex Event Processing (CEP) mit EPL / Esper

miteinander vergleichen. Damit verbunden sollen Sie eine Bewertung der aus Ihrer Sicht jeweiligen Stärken und Schwächen vornehmen bzw. den jeweiligen Erfüllungsgrad von Kriterien einschätzen.

Bei der Bewertung können Sie sich anhand der folgenden Kriterien orientieren:

1. Repräsentation von Ereignissen: Wie und mit welchen Datenstrukturen werden Ereignisse repräsentiert? Wie flexibel schätzen Sie die Datenstrukturen ein? Werden Vererbungsstrukturen unterstützt? Werden Metainformationen mitgeführt?

2. Wie werden Ereignisse / Nachrichten erzeugt und wie können diese dem Broker übermittelt werden? Welche Schritte sind hierzu erforderlich? Können Ereignisse gebündelt übertragen werden?

3. Wie kann eine Konsumentenanwendung Ereignisse / Nachrichten von dem Broker entgegennehmen? Welche Stategien werden hierzu angeboten?

4. Werden Ereignisse / Nachrichten dauerhaft gespeichert?

5. Welche Auslieferungs- und Verarbeitungsgarantien werden unterstützt?

6. Welche Maßnahmen müssen ergriffen werden, damit Ereignisse / Nachrichten gegen einen unberechtigten Zugriff gesichert werden können?

7. Welche Programmiermodelle werden angeboten, um eintreffende Nachrichten / Ereignisse zu verarbeiten, beispielsweise Aggregationsfunktionen anzuwenden oder Nachrichten / Ereignisse zeitlich zu gruppieren? Wird eine Auswertung nach Ereigniszeit unterstützt?

8. Welche Skalierungsmöglichkeiten werden vom System unterstützt, um mitwachsen zu können hinsichtlich Datenvolumen, Datenfrequenz sowie Anzahl von Produzenten und Konsumenten?

9. Gibt es Konzepte hinsichtlich einer Ausfallsicherheit des Systems? Wenn ja, wie ist die grundsätzliche Arbeitsweise.

10. Typsicherheit: Wie typsicher ist das Programmiermodell? Wie können Fehler bei der Interpretation einer Nachricht erkannt werden? Z.B. erwartet ein Konsument einen Ereignistyp A, erhält jedoch eine Instanz vom Typ B.

11. In welchen Programmiersprachen können die Produzenten- und Konsumentenanwendungen erstellt werden?

Geben Sie auch eine persönliche Einschätzung zu den Stärken, Schwächen und dem zukünftigen Potenzial der genannten Technologien.

Ihre Darstellungen zu dieser Aufgabe 10 können Sie in einem separaten Pdf-Dokument hinterlegen oder in das Lösungsdokument der Aufgaben 1 - 9 aufnehmen.