Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik



Masterarbeit

# Polyglotte Persistenz zur Prozessierung und Analyse von Metaproteomdaten

Autor:

Sören Falkenberg

05. September 2019

Betreuer:

Prof. Gunter Saake

Arbeitsgruppe Datenbanken und Software Engineering

Dr.-Ing. Sascha Bosse

Magdeburg Research and Competence Cluster

**Falkenberg, Sören:**
*Polyglotte Persistenz zur Prozessierung und Analyse von Metaproteomdaten*
Masterarbeit, Otto-von-Guericke-Universität Magdeburg, 2019.

# Inhaltsangabe

Ein Einblick in die Phänotypen von Mikroorganismen wird in der Metaproteomik durch Identifizierung und Quantifizierung von Proteinen aus mikrobiellen Gemeinschaften ermöglicht. Der Arbeitsablauf unterteilt sich dabei in die auf Massenspektrometerdaten basierende Proteindatenbanksuche und anschließende Analysen in Form von beispielsweise Quantifizierungen und Taxonomie-Funktion-Relationen. Aufgrund der Diversität dieser Prozesse und der Menge der zu prozessierenden Daten lassen sich unterschiedliche Anforderungen an die Datenhaltung ableiten.

Für ein auf Big-Data-Technologien basierendes Cloud-System soll untersucht werden, ob die Verwendung von polyglotter Persistenz zu einer höheren Effizienz bezüglich dieser Prozessierung und Analyse von Metaproteomdaten führt. Dafür wird ein Konzept entwickelt, welches beschreibt, wie die Daten aus einer spaltenorientierten Cassandra-Datenbank in eine Graphdatenbank, welche der Abbildung komplexer Beziehungen zwischen den Daten gerecht wird, überführt werden können. Darüber hinaus wird eine Evaluierung durchgeführt, welche aufzeigt, dass die Kombination zweier Datenbanken in der Lage ist, jeweilige Vor- und Nachteile zu kompensieren.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Quelltextverzeichnis

# 1. Einführung

Für die Analyse von mikrobiellen Gemeinschaften hat sich im Bereich der Metaproteomik [HPCG13] ein Arbeitsablauf etabliert, der auf der Untersuchung von Proteinen in biologischen Proben unter dem Einsatz eines Massenspektrometers basiert. Die durch das Massenspektrometer gemessenen Spektren einer Probe werden digitalisiert und im Anschluss mittels Ähnlichkeitsberechnungen mit bereits identifizierten Proteinen verglichen. Da in der Metaproteomik nicht nur die Proteine einzelner Spezies, sondern die von Gemeinschaften untersucht werden, müssen für die Proteindatenbanksuche die Proteine aller relevanten Spezies verglichen werden. Auf Grund der stetigen Verbesserungen der Massenspektrometer belaufen sich die Mengen der gemessenen Daten von einem einzigen Experiment mittlerweile im Bereich mehrerer Gigabytes. Dies führt zu einer enormen Anzahl an Vergleichen und stellt die erste wichtige Anforderung an das eingesetzte Datenbankensystem dar [HSZ+17]. Vor allem CREATE- und READ-Operationen spielen hierbei eine übergeordnete Rolle.

Die anschließende Ergebnisauswertung stellt sich als die zweite Komponente im System dar. Neben der Identifikation ist es für die Forschung relevant, die Beziehung zwischen den erhobenen Daten zu erkennen und die erkannten Proteine nach Funktion und Spezies zu klassifizieren. Dafür ist es für die performante Verwendung des Systems erforderlich, dass die Daten schnell gelesen werden können. So soll es möglich sein, die Daten in einer geeigneten Form zu abstrahieren und visualisieren, um ein effizientes Arbeiten zu ermöglichen [ZSB+17]. Zum anderen muss es auch möglich sein, die vorliegenden Datensätze zu bearbeiten und gegebenenfalls zu löschen.

Dadurch ergeben sich zwei verschiedene Anforderungen an die Datenstruktur und das eingesetzte Datenbanksystem. Derzeitige Anwendungen - wie X!Tandem [CB04] und MaxQuant [TTC16] - verfolgen einen dateibasierten Ansatz. Bei der steigenden

Größe und Komplexität der Daten wird dieser den Anforderungen jedoch nicht mehr gerecht und kann als veraltet angesehen werden.

### Motivation

MStream [RKD+19], ein System das sich derzeit von Zoun et al. in der Entwicklung befindet, soll die genannten Anforderungen nun erfüllen und Proteindatenbanksuche und Ergebnisanalyse ermöglichen. Bei diesem System handelt es sich um eine Kombination von cloud-basierten Big-Data Technologien. Diese überzeugen durch ihre horizontale Skalierbarkeit und sind für große Datenmengen bestens geeignet. Zusätzlich ist es auch möglich "Live-Messungen" durchzuführen. Dabei werden die Daten, die das Massenspektrometer produziert, bereits während der Messung vom System konsumiert und verarbeitet. Dadurch muss nicht auf die Beendigung einer Messung gewartet werden und die gemessenen Spektren können parallel vom System verarbeitet werden. Während das eingesetzte Datenbanksystem für Hochgeschwindigkeitsdatenströme, die in kleinen Zeitintervallen erzeugt werden, optimiert ist, ist es noch nicht möglich, die Daten effizient zu analysieren. Dies liegt vor allem an den erforderlichen komplexen analytischen Anfragen, die auf den hoch-relationalen Ergebnissen zu vielen Join-Operationen führen würden.

### Ziele

Wie bereits erläutert, soll MStream die bestehenden Anwendungen ablösen und deren Funktionen optimal kombinieren. Dafür muss das System MStream so weiterentwickelt werden, dass es neben der Proteindatenbanksuche auch die anschließende Analyse der Daten ermöglicht.

Ziel dieser Arbeit ist es zu prüfen, ob zur Erfüllung der unterschiedlichen Anforderungen die Verwendung von zwei verschiedenen Datenbanksystemen besser geeignet ist als ein einziges. Während dafür ein Konzept entwickelt wird, sollen vor allem zwei wichtige Fragen beantwortet werden.

Als erstes stellt sich die Frage, welche Anforderungen die Analyse der Ergebnisse erfüllen muss. Dafür muss die Datenstruktur der vorhandenen Daten betrachtet und eine geeignete Form der Datenhaltung gefunden werden.

Die zweite Frage, die es zu beantworten gilt, betrifft eine Aussage darüber, für wie sinnvoll die Kombination zweier verschiedener Datenbanksysteme anzusehen ist. Diese Kombination, die sogenannte polyglotte Persistenz, soll bestenfalls die Stärken beider Systeme vereinen und Schwächen kompensieren. Zur Beantwortung dieser Frage, soll eine Evaluierung dienen.

Zusammengefasst soll diese Arbeit die folgenden wissenschaftliche Fragen beantworten:

**WF 1:** Welches Datenbankmodell erfüllt die Strukturanforderungen zur Analyse von Metaproteom-Daten am ehesten?

**WF 2:** Ist die polyglotte Persistenz performanter als ein einzelnes Datenbanksystem?

### Aufgaben

Um die genannten Ziele zu erreichen und die wissenschaftlichen Fragen zu beantworten, ergeben sich folgende Aufgaben:

1. Konzept für die Kombination zweier Datenbanken entwickeln
2. Datenbanksystem finden, welches im Hinblick auf die Datenstruktur geeignet ist
3. Transformator entwickeln, der die Daten vom einen ins andere System überführt
4. Prototypische Implementierung evaluieren

Zunächst soll ein Konzept entwickelt werden, wie das bestehende System MStream erweitert werden muss, um die Anforderungen zu erfüllen. Da diese Arbeit die Eignung der Verwendung polyglotter Persistenz betrachten soll, muss die Architektur des Systems dementsprechend erweitert werden. Diesbezüglich muss, um die erste wissenschaftliche Frage beantworten zu können, die Datenstruktur betrachtet und ein geeignetes Datenbanksystem für die Analyse gefunden werden. Ein prototypischer Transformator soll es dann ermöglichen, die Ergebnisdaten vom einen in das andere Datenbanksystem zu überführen. Zusätzlich soll dieser es ermöglichen, Analyse-Anfragen auszuführen. Um auch die zweite wissenschaftliche Frage beantworten zu können, soll eine Evaluierung durchgeführt werden, mit Hilfe derer die Eignung polyglotter Persistenz für diesen Anwendungsfall diskutiert wird.

### Aufbau

In den folgenden Kapiteln sollen zunächst in Kapitel 2 die Grundlagen, die für die Untersuchung und das Verständnis dieses Themas relevant sind, betrachtet werden. Im Anschluss wird in Kapitel 3 ein Konzept beschrieben, durch welches die eingangs erwähnten Ziele erreicht werden können. Darauf aufbauend folgt eine prototypische Implementierung, die in Kapitel 4 beschrieben wird. Darin werden

die verwendeten Technologien genannt und erläutert, wie bei der Umsetzung des Konzepts vorgegangen wurde. Der Prototyp soll dann im darauf folgenden Kapitel 5 evaluiert und auf seine Einsatzfähigkeit getestet werden. Nachdem in Kapitel 6 verwandte Arbeiten beschrieben werden, folgt zum Abschluss mit Kapitel 7 eine Zusammenfassung, in der ebenso ein Ausblick auf mögliche Verbesserungen gezeigt und diskutiert wird, wie in Zukunft auf diese Lösung aufgebaut werden kann.

# 2. Grundlagen

Bevor in den folgenden Kapiteln die Thematik als solches ausgearbeitet wird, ist es nötig, diverse Grundlagen zu klären. Das Forschungsfeld "Metaproteomik" und die Datenverwaltungs-Strategie "Polyglotte Persistenz" stellen einen wichtigen Parameter für das Verständnis der Arbeit dar.

## 2.1 Metaproteomik

Die Metaproteomik ist ein Forschungsfeld der Biologie, in welchem durch Identifizierung und Quantifizierung von Proteinen aus mikrobiellen Gemeinschaften ein direkter Einblick in die Phänotypen von Mikroorganismen ermöglicht wird. Das Vorhandensein und die Anzahl bestimmter Proteine einer zu untersuchenden Probe erlaubt es, funktionelle Rollen und Interaktionen einzelner Spezies in einer mikrobiellen Gemeinschaft zu verstehen [Kle19]. Solche Gemeinschaften können beispielsweise von Umweltproben aus Abwasser oder dem Boden entstammen, oder aus Organismen wie dem menschlichen Körper entnommen werden. Dadurch können Indikatoren gewonnen werden, um z.B. frühzeitig Anzeichen von Krankheiten zu erkennen, oder die Effizienz von biotechnologischen Prozessen zu optimieren [MRML07].

Einen wichtigen Teil des Arbeitsprozesses stellt dabei das Massenspektrometer dar, welches die Häufigkeiten von Massen der Moleküle misst, die sich in den biologischen Proben befinden. Durch Proteindatenbanksuchen, bei denen die digitalisierten Messergebnisse mit Proteindaten aus öffentlichen Datenbanken verglichen werden, können Rückschlüsse auf die in den Proben befindlichen Proteine gewonnen werden. Ziel ist es, sämtliche Proteine der untersuchten Organismen zu identifizieren, einer entsprechenden Spezies zuzuordnen und sie nach verschiedenen Kriterien zu klassifizieren [Rom18].

### 2.1.1 Proteindatenbanksuche

Bevor die Proteine einer mikrobiellen Probe mit bereits identifizierten Proteinen verglichen werden können, sind verschiedene Vorverarbeitungen nötig, um die zu untersuchenden Proteine in ein vom Computer lesbares Format zu überführen.

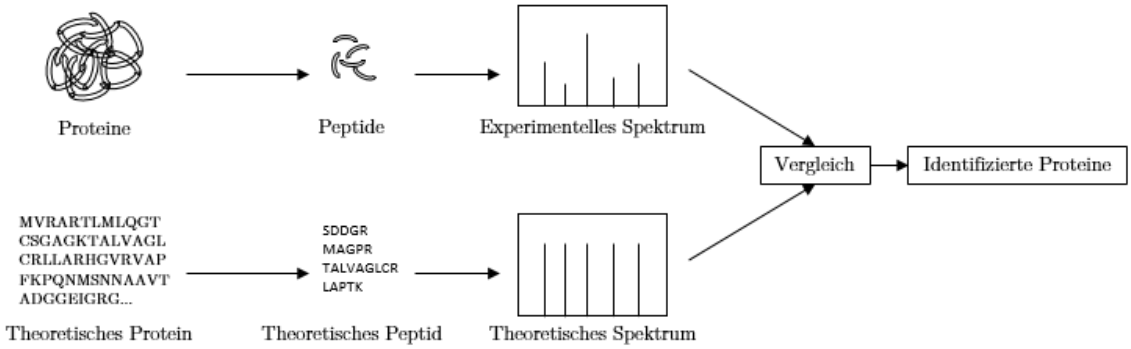


Abbildung 2.1: Ablauf der Proteindatenbanksuche nach [DAC10]

Der dafür nötige Arbeitsablauf (dargestellt in Abbildung 2.1) kann direkt vom traditionellen Ansatz der Proteomik übernommen werden und basiert auf der Benutzung eines Massenspektrometers, mittels dessen die Massen kleinster Teilchen gemessen werden. Der Unterschied zur Metaproteomik ist, dass bei der Proteomik ausschließlich Proteine eines einzigen Organismus untersucht werden [Dar13].

Ein Massenspektrometer besteht aus drei Grundkomponenten. Der Ionenquelle, die sich in der Ionisationskammer befindet, dem Massenanalysator, der das Masse-Lade-Verhältnis ($m/z$) der ionisierten Stoffe misst und dem Detektor, der die Anzahl der Ionen bei jedem $m/z$-Wert registriert. [AM03, MC15]

Vor der Messung im Massenspektrometer werden die gesammelten Proben zunächst durch biologische Prozesse bereinigt und die darin enthaltenen Proteine isoliert. Die Proteinfülle einer biologischen Probe kann dabei bis zu $10^6$ betragen [PM00]. Während die Massenspektrometrie von ganzen Proteinen weniger empfindlich ist als die Peptid-Massenspektrometrie, ist die Untersuchung der Masse des intakten Proteins alleine allerdings unzureichend für die Identifizierung. Deshalb werden die Proteine im Vorfeld enzymatisch zu Peptiden abgebaut. Anschließend wird ein Massenspektrum pro Peptid der derzeit untersuchten Probe aufgenommen und digitalisiert [AM03]. Das am häufigsten verwendete Textformat, um die Massenspektren darzustellen, ist das Mascot Generic Format (MGF). Dieses vereint mehrere Massenspektren in einer einzigen Datei und enthält für jedes die Informationen über die gemessenen Masse-Lade-Verhältnisse beziehungsweise Intensitäten [Deu12].

Bereits identifizierte Proteine werden für einen Vergleich aus Proteindatenbanken bezogen und überlicherweise im FASTA-Format textuell repräsentiert [SCY04, ZDS+18].

UniProt, eine öffentliche Wissenssammlung von Proteininformationen, enthält über 100 Millionen Proteinsequenzen aus allen Lebensbereichen [Con18]. Eine FASTA-Datei enthält einen aus einer einzigen Zeile bestehenden Header, der sich aus einer Reihe von unformatierten und sich zwischen Datenquellen unterscheidenden Metainformationen zusammensetzt. Darauf folgen eine oder mehrere Zeilen der Proteinsequenz, bei der jede Aminosäure nur durch einen Buchstaben repräsentiert wird [Deu12].

Für die anschließende Proteindatenbanksuche werden die bereits identifizierten Proteine ebenfalls in Peptide fragmentiert und nicht-redundant gesammelt. Da die Aminosäurenkette der Proteine und die Masse der zusammengesetzten Aminosäuren bekannt ist, können für jedes Peptid theoretische Spektren errechnet werden [ZDS+18]. Die gemessenen experimentellen Spektren können nun mit den theoretischen Spektren verglichen werden. Implementiert sind unterschiedliche Vergleichsalgorithmen in verschiedenen kommerziellen und nicht-kommerziellen Anwendungen, wie MASCOT [PPCC99], X!Tandem [CB04] oder OMSSA [GMK+04]. Diese errechnen für jeden Vergleich zwischen theoretischem und experimentellem Spektrum einen Grad der Übereinstimmung und bestimmen anhand der Ähnlichkeit die am ehesten zutreffende Peptidsequenz. Abschließend wird die identifizierte Peptidsequenz den Proteinen, die sie beinhalten, zugeordnet und zusammen mit den ermittelten Informationen in der Proteindatenbank hinterlegt [DAC10].

### 2.1.2 Analyse

Für die Forschung ist es im Anschluss von Relevanz, die gesammelten Informationen der untersuchten Probe auszuwerten, um Rückschlüsse auf die Qualität des Experiments und der verwendeten Probe treffen zu können [Kle19]. Dabei sollen unter anderem folgende Fragen beantwortet werden:

- Welche Organismen sind in der Gemeinschaft enthalten?
- Wie hoch ist die Wachstumsrate individueller Organismen?
- Welche Rolle spielt das Protein in der untersuchten Umgebung?

Sogenannte ”markierungsfreie” Techniken bieten dafür einen vielversprechenden Ansatz. Die markierungsfreie Quantifizierung kann direkt auf die gesammelten Daten angewendet werden und stellt einen unkomplizierten Ansatz zur Schätzung der Proteinhäufigkeit dar. Dieser umfasst das Zählen der Spektren für ein bestimmtes Protein und nimmt an, dass mit einer höheren Anzahl von Spektren ein erhöhtes

Proteinreichtum abgebildet werden kann [DAC10, MBR+13]. Durch wiederholte Untersuchungen einer Umgebung können somit Variationen und mögliche Fluktuationen der Organismen in einer begutachteten Umgebung untersucht werden.

Außerdem können die Proteindatenbanken verschiedene Metadaten zu den identifizierten Proteinen enthalten. Neben der Schlussfolgerung über die Existenz eines Proteins, stellt die Zuordnung zum taxonomischen Ursprung und die Funktion dessen eine weitere wichtige Erkenntnis dar [MVdJW+17]. Die Taxonomie, die Theorie und Praxis der biologischen Klassifikation, beschäftigt sich mit der Einteilung von Lebewesen in ein hierarchisches System. Die einzelnen Gruppen bilden sich anhand von Merkmalen, die sie von anderen Lebewesen unterscheiden. So entsteht eine Baumstruktur, die sich über 8 Ränge erstreckt. Diese reicht von der Wurzel, an der sich die Domänen Eukaryoten und Prokaryoten (Bakterien und Archaeen) trennen, bis hin zu der Spezies, die in Zoologie und Bakteriologie den niedrigsten Rang darstellt. Namen und Abstammungen von Organismen können beispielsweise der NCBI-Datenbank (National Center for Biotechnology Information) entnommen werden [Fed11, Cai18]. Weiterhin können den Proteinen auch biologische Prozesse und molekulare Funktionen zugeordnet werden. Diese besitzen ebenfalls eine hierarchische Struktur und sind zum Beispiel in der Datenbank UniProtKB [1] hinterlegt.

Um mit den in der Metaproteomik erwarteten Mehrartensystemen besser arbeiten zu können, wird in [MBH+15] eine Technik beschrieben, wie Gruppen von verwandten Proteinen zu sogenannten Metaproteinen, einer weiteren Abstraktionsebene, zusammengefasst werden. Diese Gruppierung erfolgt dabei nach einem bestimmten Regelwerk, das abhängig von Analysezweck und Spezifikation gewählt werden könnte. Das Metaprotein kombiniert somit die Metadaten der Proteine in einem einzigen Eintrag. Für die Taxonomie wird zum Beispiel der niedrigste gemeinsame Ahne gewählt. Ein Metaprotein wird also nicht als einzelnes Protein mit mehrdeutiger Identifizierung betrachtet, sondern stellt eine Gruppe aus verwandten Proteinen dar, die potenziell in der Probe enthalten sind und bietet somit zusätzliche Analysemöglichkeiten.

## 2.2 Datenbanksysteme

Unter einer Datenbank wird eine Sammlung strukturierter Daten verstanden, welche über spezielle Anwendungen die Fakten der modellierten Welt repräsentiert und sie dauerhaft (persistent) speichert. Das Datenbank-Management-System (DBMS) bezeichnet die Software, die das anwendungsunabhängige Erzeugen, Löschen und Ändern einer Datenbank ermöglicht. Die Kombination eines DBMS und einer oder

[1] https://www.uniprot.org/docs/keywlist

mehrerer, unterscheidbarer Datenbanken wird folglich als Datenbanksystem verstanden. In der Abbildung 2.2 ist eine Grobarchitektur eines Datenbanksystems dargestellt, bei welchem verschiedene Anwendungen auf die im System verwalteten Datenbanken zugreifen [SST97].

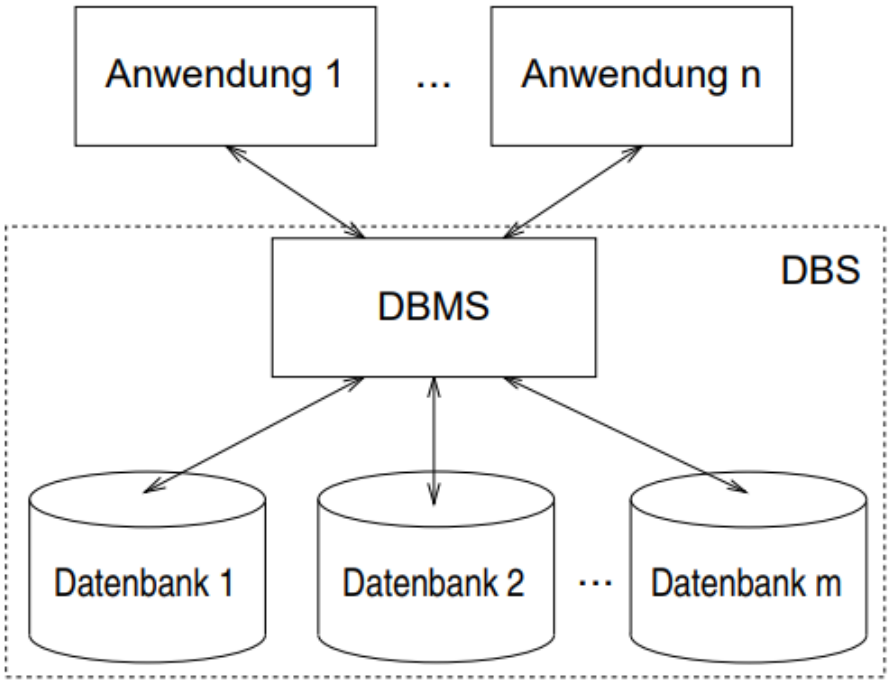


Abbildung 2.2: Grobarchitektur eines Datenbanksystems [SST97]

Seit Edgar Codd 1970 das relationale Datenbankmodell einführte, sind relationale Datenbanken das meistgenutzte Modell für die Datenhaltung und werden auch heute noch für Finanzunterlagen, Personendaten und vieles mehr verwendet [NLIH13, MGAOI14]. Seit der Entstehung des Web 2.0 und Innovationen wie Cloud Computing wurden Anwendungen jedoch immer komplexer, wodurch auch die Anforderungen an die verwendeten Datenbanksysteme stiegen [JHGJ11, HSS11]. Zu diesen gehören die folgenden:

- Hohe Schreib- und Leselast bei geringer Latenz
- Effiziente Speicherung großer Datenmengen
- Hohe Verfügbarkeit und Skalierbarkeit
- Geringe Betriebs- und Managementkosten

Da relationalen Datenbanken im Bezug auf die genannten Anforderungen einige Einschränkungen aufweisen, entstanden eine Vielzahl neuer Arten von Datenbanken.

Diese unterscheiden sich im Allgemeinen sehr von den relationalen Datenbanken und werden als NoSQL-Datenbanken ("Not only SQL") bezeichnet [JHGJ11].

In den nachfolgenden Abschnitten sollen dafür zunächst die relationalen und NoSQL-Datenbanken näher beschrieben werden.

### 2.2.1 Relationale Datenbanksysteme

Relationale Datenbanken erschienen bereits in den frühen 1970er Jahren und basieren auf dem gleichnamigen Datenmodell von Edgar Codd. Grundlegend besteht eine relationale Datenbank aus einer Ansammlung von Tabellen, hinter denen mathematisch die Idee einer Relation steht [SSH11]. Die Daten werden also strukturiert als Tupel dargestellt. Durch die Entkopplung von logischer und physikalischer Darstellung bieten sie ein hohes Maß an Datenunabhängigkeit. Darauf basierend wurden deklarative Abfragesprachen definiert, durch die Anwendungsentwickler Zugriff zu den Daten erhielten. Als Standardsprache für Definition, Manipulation und Abfrage der Daten hat sich seither die Structured Query Language (SQL) durchgesetzt. Aufgrund der einfachen Handhabung konnten diese Systeme bisherige Systeme ablösen und werden noch heute weit verbreitet für verschiedene Anwendungen in der Geschäftsdatenverarbeitung eingesetzt.

In relationalen Datenbanken wird die Struktur der Daten durch die Gestaltung der Tabellen definiert. Dafür werden im Vorfeld die Namen und Datentypen der Spalten bestimmt. Außerdem garantieren sie eine hohe Transaktionszuverlässigkeit, da die Transaktionen die Regeln der vier ACID-Grundprinzipien einhalten. Diese Grundprinzipien lauten Atomarität (**A**tomicity), Konsistenz (**C**onsistency), Abgrenzung (**I**solation) und Dauerhaftigkeit (**D**urability) [Cat11].

- Atomarität: Wenn ein Teil der Transaktion unvollständig bleibt, wird die gesamte Transaktion abgebrochen.
- Konsistenz: Versichert, dass die Datenbank vor und nach jeder Transaktion stabil ist und sich in einem validen Zustand befindet.
- Abgrenzung: Unterschiedliche Transaktionen bleiben voneinander isoliert und wirken sich nicht auf die jeweils andere Ausführung aus.
- Dauerhaftigkeit: Ein Datensatz bleibt nach erfolgreicher Transaktion dauerhaft gespeichert.

Als größte Herausforderung ist die vertikale Skalierbarkeit von relationalen Datenbanken zu nennen. Da sie vom Hinzufügen weiterer Hardware-Ressourcen abhängig ist,

ist die vertikale Skalierung aus Gründen der Hardwarebeschränkung eher unpraktisch. Infolgedessen sind relationale Datenbanken für Webanwendungen und andere Dienste, die Millionen gleichzeitige Nutzer unterstützen, weniger geeignet [PRS+11].

### 2.2.2 NoSQL-Datenbanksysteme

Die Entwicklung von Technologien zur Verwaltung großer Mengen heterogener Daten, welche mit hoher Geschwindigkeit von Mensch und Maschine erzeugt und verwaltet werden müssen, gehörte zu den größten Herausforderungen in der Datenbankforschung der letzten Jahre. Aktuelle relationale Datenbankentechnologien verfügen gerade im Bereich von Big-Data-Anwendungen über unzureichende Ressourcen, um deren Daten zu verarbeiten. So stieg die Nachfrage nach einer neuen Klasse von nicht-relationalen Datenspeichersystemen, die unter dem Oberbegriff NoSQL zusammengefasst werden. NoSQL-Datenbanksysteme stellen aufgrund ihrer Charakteristika - wie Flexibilität des Datenschemas und horizontale Skalierbarkeit - eine effektive Lösung zur Verwaltung großer heterogener Datensätze, welche auf viele Server verteilt sind, dar [ROdVC16]. Dadurch wird eine sehr hohe Erreichbarkeit geboten.

Die horizontale Skalierbarkeit, der die Replikation der Daten auf mehrere Knoten innerhalb eines Clusters einhergeht, kann jedoch nur auf Kosten temporärer Inkonsistenzen innerhalb des Systems erreicht werden [Bre12]. So basieren viele NoSQL-Systeme - im Gegensatz zu den strengen ACID-Prinzipien der relationalen Datenbanksysteme - nur auf den gelockerten BASE-Eigenschaften. Diese stehen für **B**asically **A**vailable, **S**oft **S**tate und **E**ventual consistency. Damit wird ausgedrückt, dass das System immer erreichbar ist, aber nicht immer das aktuellste Ergebnis liefert. "Irgendwann" erreicht das System einen konsistenten Zustand [MGAOI14].

Während sich relationale Datenbanksysteme also auf die Konsistenz der Datensätze fokussieren, steht für BASE-Systeme die Verfügbarkeit im Vordergrund. Einhergehend mit einer Lockerung der ACID-Eigenschaften ermöglichen die Transaktionsmodelle der NoSQL-Systeme geringere Antwortzeiten, da nicht auf einen konsistenten Zustand gewartet werden muss, bis eine weitere Abfrage bearbeitet werden kann.

Das CAP-Theorem [Bre00], welches bereits im Jahr 2000 aufgestellt wurde, besagt, dass es in verteilten Systemen nicht möglich ist, die Eigenschaften Konsistenz (**C**onsistency), Verfügbarkeit (**A**vailability) und Partitionstoleranz (**P**artition Tolerance) gleichermaßen zu erfüllen.
Die Eigenschaften besagen, dass die Daten

- auf allen Knoten des Systems identisch sind (Konsistenz)

- jederzeit erreichbar und verfügbar sind (Verfügbarkeit)
- das System auch dann korrekt weiterarbeitet, wenn es partitioniert wird oder einzelne Knoten ausfallen (Partitionstoleranz)

Entscheidend für ein NoSQL-System sind daher die Anforderungen der Anwendung. So soll es SQL nicht verwerfen, sondern ein alternatives System für Anwendungen darstellen, für die ein relationales Modell nicht geeignet erscheint. Kann eine Anwendung auf hohe Konsistenz verzichten, benötigt jedoch ein hochverfügbares und partitionstolerantes Modell, ist NoSQL möglicherweise die bessere Wahl. Für Anwendungen im Finanzbereich hingegen ist die optimistische Art der BASE-Eigenschaften beispielsweise ungeeignet [HJ11].

Nachfolgend sollen die verschiedenen NoSQL-Modelle erläutert werden. Jedes der vorgestellten Modelle ist auf Grund seiner Eigenschaften für unterschiedliche Verwendungszwecke geeignet. Die Entscheidung für den Einsatz eines jeden Systems ist deshalb abhängig vom jeweiligen Anwendungsszenario [DCL18].

#### Key-Value-Datenbankmodelle

Diese sind die populärsten und simpelsten Systeme, in denen die Daten als Schlüssel-Wert-Paare in effizienten, hochskalierbaren Strukturen gehalten werden. Typ und Struktur der Daten können dabei willkürlich gewählt werden. Jeder Schlüssel ist einzigartig und besteht aus einem Hash, einer URI oder einem Dateinamen. Da der Zugriff immer über den Primärschlüssel läuft, sind diese Datenbanken sehr performant. Allerdings können die Anfragen nicht gezielt nach den Werten fragen, die sich hinter den Schlüsseln verbergen [AXF+12].

#### Dokumentenorientierte Datenbankmodelle

Dokumentenorientierte Datenbanken ähneln den Key-Value Datenbanken - die Daten werden ebenfalls als Schlüssel-Wert-Paar gespeichert. Unterschiedlich ist jedoch, dass es sich bei den Werten um halbstrukturierte oder strukturierte Daten handelt. Diese Daten werden als Dokument bezeichnet und können im XML- oder JSON-Format vorliegen. Im Gegensatz zu den Key-Value Datenbanken kennen die dokumentenbasierten Systeme das Format der Dokumente und ermöglichen daher Indizierung und Suchfunktionalitäten basierend auf Attributen und Werten [V14].

#### Spaltenorientierte Datenbankmodelle

Spaltenorientierte Datenbanken stellen eine Erweiterung des Key-Value-Modells dar. Jede Zeile eines Key-Value-Paars besitzt weitere, verschachtelte Key-Value-Paare, die

als Spalten bezeichnet werden. Die Anzahl der Spalten ist dabei jedoch flexibel und kann während der Laufzeit variieren. Weiterhin repräsentiert jede Zeile ein hochstrukturiertes Datenelement, das ebenfalls durch einen Schlüssel eindeutig identifiziert werden kann. Die Möglichkeit der Adressierung einzelner Spalten ermöglicht beschleunigte Leseprozesse, da keine unnötigen Daten gelesen werden müssen. Demzufolge profitieren vor allem Anwendungen mit vielen Lese- und Schreibprozessen von diesem Modell [JABH09].

#### Graphendatenbankmodelle

Während sich die bereits angeführten Systeme darauf fokussieren, Entitäten zu speichern, basieren Graphdatenbanken auf der Graphentheorie und sind diesbezüglich besser für graphenähnliche Datenmengen geeignet. Entitäten werden als Knoten und die Beziehungen zwischen ihnen als Kanten dargestellt. Dies ermöglicht eine Traversierung zwischen den einzelnen Entitäten und erlaubt eine effizientere Analyse komplexer Datensätze, bei der die Antwortzeit nicht abhängig von der Datenmenge, sondern der Komplexität und zu traversierenden Knoten abhängig ist [VMZ+10].

Abhängig davon, welches Problem eine Anwendung lösen soll, wird bestimmt, ob und welches NoSQL-Modell verwendet wird. Weiterhin ist es auch möglich, dass man sich für eine hybride Mischung verschiedener NoSQL- und SQL-Datenbanken entscheidet [NLIH13].

### 2.2.3 Polyglotte Persistenz

Da sich moderne Anwendungen zunehmend mit der Aufgabe konfrontiert sehen, steigende Nutzerzahlen und wachsende Datenmengen zu bedienen, nehmen gleichzeitig auch die Anforderungen an diese zu. Eine hohe Schemaflexibilität, Ausfallsicherheit und horizontale Skalierbarkeit sind in Zeiten datengetriebener Anwendungen unabdingbar geworden. Mit der ansteigenden Systemvielfalt von NoSQL-Datenbankmodellen ist es jedoch möglich, für bestimmte Probleme das jeweils am besten geeignete Datenbanksystem zu wählen. So ist es auch möglich, innerhalb einer Anwendung mehrere Datenbankmodelle zu kombinieren [GR15].
Bereits 2006 führte Neal Ford den Begriff "Polyglot Programming" (Polyglotte Programmierung) ein, um auszudrücken, dass Anwendungen in einer Mischung aus Sprachen geschrieben werden sollten. Dadurch soll die Tatsache genutzt werden, dass bestimmte Sprachen für bestimmte Probleme besser geeignet sind als andere. Da komplexe Anwendungen unterschiedliche Arten von Problemen kombinieren, sei es produktiver, nicht die gesamte Anwendung in einer einzigen Sprache zu verfassen [SF12]. Basierend darauf prägte Martin Fowler 2011 den Begriff "Polyglot Persistence" (Polyglotte Persistenz), um die Idee einer hybriden Datenbankarchitektur zu

betiteln. Für verschiedene Anforderungen innerhalb einer Anwendung bezeichnet diese Architektur den Einsatz verschiedener Datenbanken und stellt sich somit gegen die Theorie der ”one-size-fits-all”-Datenbank [GR15, SC05].

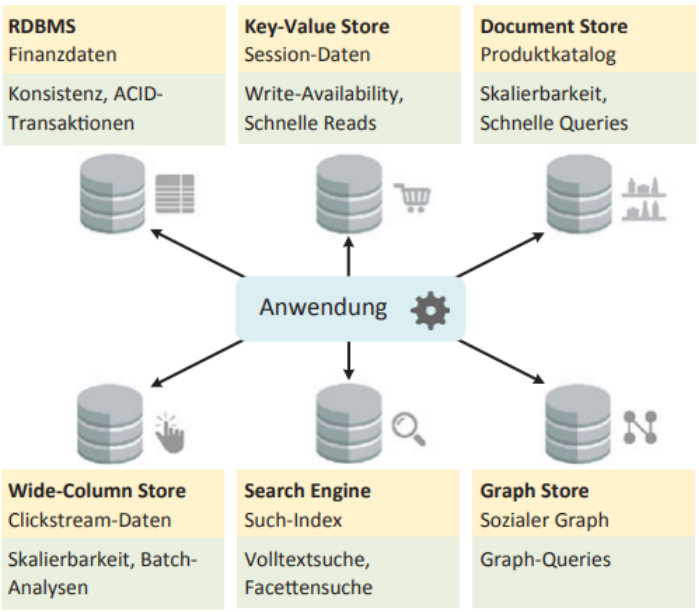


Abbildung 2.3: Beispiel einer Polyglotte-Persistenz-Architektur [GR15]

Abbildung 2.3 zeigt eine Polyglotte-Persistenz-Architektur am Beispiel einer E-Commerce-Applikation. Für die einzelnen Funktionen spricht die Anwendung dabei verschiedene Datenbanken an. So kann aufgrund der Einhaltung der ACID-Grundprinzipien eine relationale Datenbank beispielsweise für die Finanztransaktionen verwendet werden. Eine dokumentenorientierte Datenbank eignet sich dank der Skalierbarkeit des Datenvolumens und der Leselast für die Produktbeschreibungen, deren Struktur sich ideal als Dokument abspeichern lässt. Anwendungsereignisse können in einer spaltenorientierten Datenbank gespeichert werden, die sich durch ihre hohen Schreibraten auszeichnet und zusätzlich Anbindungen an analytische Plattformen bietet [GR15]. Sollen außerdem noch Produktvorschläge getätigt werden, die von anderen Nutzern getätigt wurden, bietet sich eine Graphdatenbank an [SF12].

Allerdings muss es nicht immer die Kombination von unterschiedlichen Datenbankmodellen sein. Allein schon die Kombination mehrerer für bestimmte Zwecke spezialisierte relationale Datenbanken - wie im Falle des Data-Warehouse - kann als polyglotte Persistenz angesehen werden [HLLP15].

## 2.3 Zusammenfassung

In diesem Kapitel wurde zunächst das Forschungsfeld der Metaproteomik vorgestellt und ein Einstieg in die Prozessierung und Analyse von Metaproteomdaten gegeben.

Dabei wurde der auf einem Massenspektrometer basierende Arbeitsablauf der Proteindatenbanksuche und typische Analysemethoden der erhobenen Daten beschrieben. Anschließend folgte eine grundlegende Umschreibung verschiedener Datenbanksysteme inklusive der Unterscheidung zwischen relationalen und NoSQL-Systemen, sowie eine generelle Erklärung des Begriffs ”Polyglotte Persistenz”.

# 3. Konzept

Nachdem die Grundlagen erklärt wurden, soll nun ein Konzept für den Einsatz polyglotter Persistenz zur Prozessierung und Analyse von Metaproteindaten entwickelt werden. Dies wird anhand einer Erweiterung des bereits bestehenden Systems "MStream" demonstriert. Dafür wird zunächst das System und die Anforderungen beschrieben. Infolgedessen wird die Struktur der Daten analysiert und gezeigt, wie sich eine zusätzliche Datenbank in MStream integrieren lässt.

## 3.1 MStream

MStream ist ein System, welches von Zoun et al. entwickelt wurde und eine cloudbasierte Lösung für die Verarbeitung von Metaproteomdaten bietet. Die Grobarchitektur des Cloud-Systems ist in Abbildung 3.1 dargestellt und zeigt den Zusammenschluss der eingesetzten Big-Data-Technologien. Die Technologien zeichnen sich durch ihre horizontale Skalierbarkeit aus und ermöglichen eine effiziente Verarbeitung der Metaproteomdaten in naher Echtzeit [RKD+19, ZSB+19].

### 3.1.1 Architektur und Komponenten

Das Massenspektrometer, das die Experimentmessungen durchführt, ist über einen Digitalisierer direkt mit dem System verbunden und ermöglicht es dadurch, sogenannte "Live-Messungen" durchzuführen. Dabei werden die gemessenen Spektren bereits während der Messung zur Verarbeitung in die Cloud gesendet. Die Datenverarbeitung und damit einhergehende Proteindatenbanksuche kann somit parallel zur unabdingbaren Mess- und Digitalisierungsdauer des Massenspektrometers durchgeführt werden, wodurch zusätzlich auch die komplizierte und manuelle Handhabung der Rohdaten entfällt [ZSB+19].

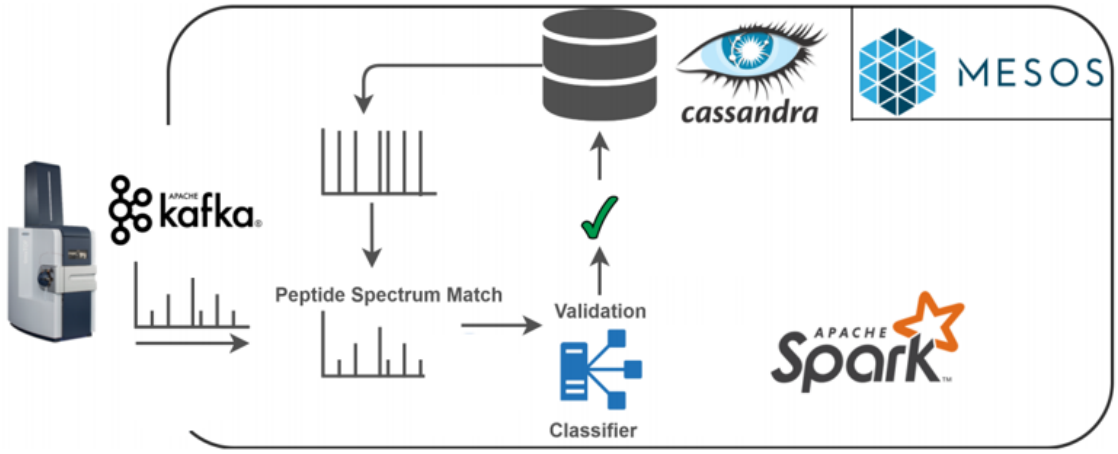


Abbildung 3.1: Grobarchitektur MStream nach [ZSB+19]

Die vom Massenspektrometer produzierten experimentellen Spektren werden noch während der Messung in ein Apache-Kafka-Cluster [Gar13] geschrieben und können nach Bedarf und Verfügbarkeit vom weiteren System konsumiert werden. Apache-Kafka stellt eine Nachrichten-Queue zur Verfügung, die Speicherung und Verarbeitung von Datenströmen ermöglicht und stellt die Schnittstelle zwischen dem Massenspektrometer und der Cloud dar [ZSB+19].

Für jedes eintreffende Spektrum wird ein Vergleich mit bereits in der Datenbank befindlichen Proteindaten durchgeführt. Durchgeführt werden die Vergleiche in einem Apache-Spark-Cluster, welches auf die Verarbeitung hoher Datenmengen optimiert ist und eine Parallelisierung der Arbeitsschritte ermöglicht [ZXW+16]. Übereinstimmungen zwischen den experimentellen und theoretischen Spektren, welche sich aus den Proteindaten ergeben, erzeugen ”Peptid-Spektrum-Matches” [ZDS+18]. Diese Übereinstimmungen werden, basierend auf Machine Learning Klassifikatoren, auf Fehlerkennungen untersucht und zusammen mit den experimentellen Daten in die Datenbank geschrieben [ZSJ+18, ZSB+19]. Verwendung findet in MStream Cassandra, eine spaltenorientierte NoSQL Datenbank, welche sich durch hohe Skalierbarkeit auszeichnet und für den Umgang mit großen Datenmengen geeignet ist [LM10].

Der derzeitige Entwicklungsstand bietet eine vielversprechende Lösung einer performanten Proteindatenbanksuche und beschleunigt den Arbeitsablauf durch die Idee der ”Live-Messungen” enorm. Die Verbesserung der Analyse bleibt dabei jedoch aus, was der komplexen Datenstruktur und ausbleibenden Visualisierungsmöglichkeiten geschuldet ist.

### 3.1.2 Datenstruktur der Cassandra-Datenbank

Um das folgende Konzept einer polyglotten Datenverwaltung rechtfertigen zu können, wird zunächst die derzeitige Datenstruktur der spaltenorientierten Datenbank Cassandra betrachtet, die speziell für die Proteindatenbanksuche angelegt wurde. In Abbildung 3.2 ist der Aufbau der Tabellen innerhalb MStreams veranschaulicht. An dieser Stelle wird, um die Verständlichkeit nicht zu beeinträchtigen, ausschließlich auf die Tabellen eingegangen, die für den Kontext der Arbeit relevant sind. Die Struktur der verwendeten Tabellen ist so gewählt, dass die Proteindatenbanksuche möglichst effizient durchgeführt werden kann [RKD+19].

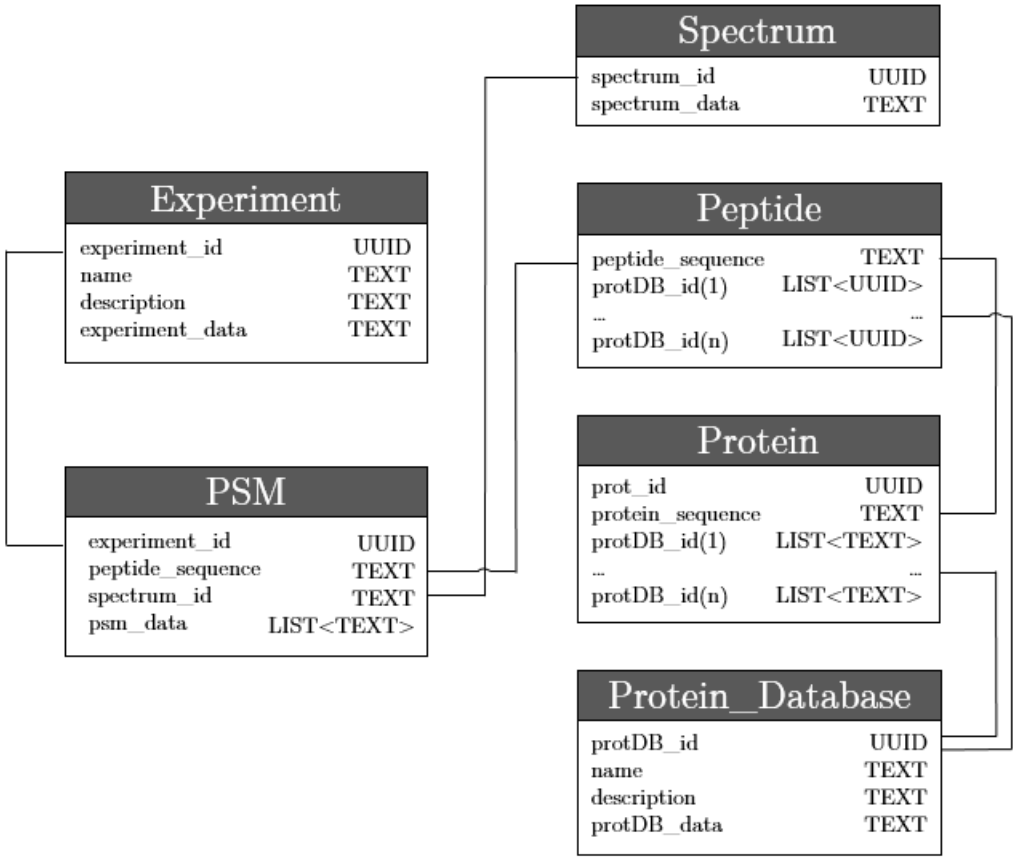


Abbildung 3.2: Datenbankstruktur in Cassandra

#### Experiment-Tabelle

In dieser Tabelle sind Informationen über die durchgeführten Experimente hinterlegt. Jedem Experiment ist eine eindeutige UUID und beschreibende Informationen wie Name und Beschreibung zugeordnet.

#### PSM-Tabelle

Die bei der Proteindatenbanksuche entstandenen Paarungen der aus den Peptiden gebildeten theoretischen und aus dem durchgeführten Experiment stammenden ex-

perimentellen Spektren werden in der Peptid-Spectrum-Match (kurz PSM) Tabelle gespeichert. Jede Übereinstimmung besteht zum einen aus der ID des jeweiligen Spektrums, der Sequenz des zugeordneten Peptids und der ID des entsprechenden Experiments. Zum anderen werden zusätzliche Informationen über die Übereinstimmung und für die Ähnlichkeitsanalyse verwendete Funktion gespeichert. Diese hat besonderen Einfluss auf die spätere Analyse, da unterschiedliche Vergleichsfunktionen zu unterschiedlichen Resultaten führen können.

#### Spectrum-Tabelle

Diese Tabelle beinhaltet die Informationen aller bisher in das System eingebrachten experimentellen Spektren. Diese können über eine eindeutige UUID referenziert werden und enthalten Metadaten, die während der Messung aufgenommen wurden.

#### Peptide-Tabelle

In dieser Tabelle sind die gesammelten Informationen aller Peptide gespeichert. Peptide stellen Proteinfragmente dar und werden aus den Informationen der Protein-Tabelle erzeugt. Bei der Proteindatenbanksuche werden aus ihnen die theoretischen Spektren gebildet, die für die Ähnlichkeitsanalyse verwendet werden. Jeder Eintrag ist eindeutig über die Sequenz der Aminosäuren referenzierbar und enthält neben Metadaten auch eine Liste von IDs der Proteine, denen die Peptide zugeordnet werden können.

#### Protein-Tabelle

Die Protein-Tabelle enthält alle Informationen über Proteine, die aus verschiedenen Proteindatenbanken zusammengetragen wurden. Im Gegensatz zu den anderen Tabellen sind die Spalten der Protein-Tabelle dynamisch und abhängig von den Einträgen in der Protein_Database-Tabelle. Dies ist besonders deshalb unabdingbar, da differierende Quellen unter Umständen zu voneinander abweichenden Informationen führen können.
Außerdem beinhaltet jede dieser Spalten eine Liste der Proteine, die der Proteinsequenz zugeordnet werden konnten. Dies ergibt sich daraus, dass Proteine bei gleichbleibender Sequenz in unterschiedlichen Organismen oder abweichenden Voraussetzungen vorkommen können.

#### Protein_Database-Tabelle

Die Protein_Database-Tabelle beschreibt die Quelle, aus der die Protein-Informationen entnommen wurden. Dazu gehören Informationen, wie beispielsweise Taxonomien

und Funktionen, die den Proteinen zugeordnet wurden. Die Spalten der Tabelle setzen sich aus einer ID, einem Namen, einer Beschreibung und möglicher - die Datenquelle definierender - Metadaten zusammen. Referenziert wird diese Tabelle direkt über die entsprechenden Spalten-Bezeichnungen in der Peptide- sowie Protein-Tabelle.

## 3.2 Anforderungsanalyse

Um die erforderlichen Eigenschaften für das aufzustellende Konzept zu definieren, wird es nötig, zuerst die Anforderungen an dasselbe genauer zu beleuchten, was hier in Form einer Anforderungsanalyse erfolgen soll. Die zu benennenden Anforderungen sind in zwei Bereiche - die allgemeinen Anforderungen und die Anforderungen an die Datenstruktur - zu untergliedern.

### 3.2.1 Allgemeine Anforderungen

Eine wichtige Rolle kommt dem Erhalt der Datenstruktur und den damit verbundenen komplexen Detailinformationen in Cassandra zu. Da die Fähigkeiten im Erlangen wissenschaftlicher Erkenntnisse immer potenter und damit das Datenvolumen immer größer wird, braucht es eine Datenbank, die für hohe Lese- und Schreibraten geeignet ist. Die Bedeutsamkeit Cassandras für eine effektive Proteindatenbanksuche wurde bereits in [RKD+19] erklärt und soll demnach erhalten bleiben.

Des weiteren soll die Visualisierung der Daten und eine nutzbringende Verarbeitung selbiger ermöglicht werden. Diese Fähigkeit zur Analyse der Daten ist ein wesentlicher Faktor. Das zu entwickelnde Konzept soll allerdings nicht nur die Analyse der gesammelten Daten, sondern auch weiterhin die Proteindatenbanksuche ermöglichen. Dazu zählt auch, dass das derzeitige Datenmodell und die verwendeten Technologien unverändert bleiben, eine Skalierbarkeit durch hinzugefügte Technologien nicht beeinträchtigt wird und die Häufung zeitgleicher Analysen die Proteindatenbanksuche weder verhindert noch verzögert. Eine Analyse muss jederzeit und beliebig oft wiederholbar sein. Das Durchführen von Analysen in Form von Protein-Quantifizierungen, Taxonomie-Funktion-Relationen, und Filterfunktionen, um ausschließlich exemplarisch einige wenige zu benennen, soll schlussendlich möglich werden. Einhergeht, dass bei der Analyse zur Visualisierung förderliche Änderungen vorgenommen und persistiert werden können. Dadurch soll ein dynamischer und kollaborativer Arbeitsablauf gefördert werden.

Ebenso kann es vorkommen, dass bei der Analyse der Datensätze im Vorfeld nicht bekannt ist, wonach gesucht wird. Verschiedene Parameter müssen während der Analyse dementsprechend ignorierbar sein oder hinzugefügt werden können. Daraus

lässt sich eine weitere Anforderung schlussfolgern, nämlich, dass das Datenschema eine explorative Arbeit unterstützt und daher flexibel aufgebaut sein muss.

Zusammengefasst können die Anforderungen wie folgt in funktionale und nicht-funktionale unterteilt werden:

**Funktionale Anforderungen**

- Taxonomien und Proteinfunktionen können abgebildet werden
- Proteindatenbanksuche ist weiterhin möglich
- Analyse der Daten in Form von verschiedenen Filtern und Aggregierungen wird ermöglicht
- Änderungen am Datensatz können vorgenommen und persistiert werden

**Nicht-funktionale Anforderungen**

- Datenstruktur und komplexe Einzelinformationen der Daten in Cassandra bleiben erhalten
- Datenstruktur ist schemafrei und flexibel
- Analyse kann beliebig oft wiederholt werden
- Analyse behindert dabei nicht die Proteindatenbanksuche
- Skalierbarkeit des Systems wird nicht beeinträchtigt
- Kollaborativer und explorativer Arbeitsablauf wird gefördert

### 3.2.2 Anforderungen an die Datenstruktur

An dieser Stelle sollen verschiedene Analysemöglichkeiten, die typisch für den Umgang mit Metaproteomdaten sind und somit eine dringende Anforderung an die zu verwendende Datenstruktur darstellen, beleuchtet werden. Dabei ist Darstellung der Daten in den kommenden Abbildungen ist den in Abschnitt 3.1.2 beschriebenen Tabellenbeziehungen nachempfunden. Während Proteine aus mehreren Peptiden bestehen, können jedem Peptid mehrere Spektren zugeordnet sein. Weiterhin sollen die ausgewählten Methoden deutlich machen, welche Bedeutung die Auswertung der Beziehungen zwischen den Daten hat.

### Proteinquantifizierung

Die Quantifizierung - also das Zählen der einem Protein zugeordneten experimentellen Spektren - spielt eine zentrale Rolle, um Schätzungen über die Proteinfülle der untersuchten Probe durchführen zu können. Die Spektren werden dabei - ausgehend vom Protein - nicht-redundant gezählt. Um diese Zählung durchführen zu können, soll es möglich sein, Beziehungen zwischen Proteinen und Spektren abzubilden.

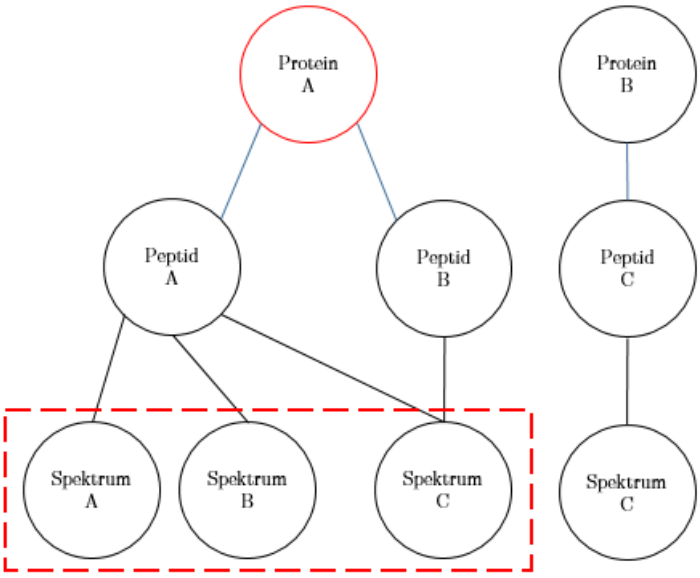


Abbildung 3.3: Darstellung einer Proteinquantifizierung

Mit Abbildung 3.3 soll eine Proteinquantifizierung dargestellt werden. Protein A besteht aus zwei Peptiden. Da die Spektren nicht-redundant gezählt werden, sind es zwar vier PSM-Beziehungen, aber nur drei Spektren sind dem Protein zugeordnet. Das Protein A ist demzufolge mit größerer Fülle und Wahrscheinlichkeit enthalten.

### Filtern nach PSM

Die in Abschnitt 3.1.2 beschriebene PSM-Tabelle enthält die in der Proteindatenbanksuche gefundenen Übereinstimmungen zwischen Spektren und Peptiden. Da diese Übereinstimmung die Ähnlichkeit zwischen den experimentellen Spektren und theoretischen Spektren darstellt, die aus den vorliegenden Pepdiddaten erzeugt wurden, ist diese Verbindung von Experiment und Proteindatensatz von großer Bedeutung. Daher ist auch die Wahl der Vergleichsfunktion entscheidend für die Analyse. Diese bestimmt bei der Ähnlichkeitsanalyse der Proteindatenbanksuche, welche Spektren welchen Peptiden zugeteilt werden. Aufgrund dessen ist in der PSM-Tabelle für jede Übereinstimmung auch der Name der verwendeten Vergleichsfunktion und der ermittelte Ähnlichkeitswert hinterlegt.

Für die Analyse der Daten soll es nun - die Visualisierung unterstützend - möglich sein, Spektren außerhalb eines bestimmten Schwellwerts liegend, ein- beziehungsweise auszublenden. Abbildung 3.4 soll dafür eine beispielhafte PSM-Filterung darstellen, bei der nur die Spektren, deren Ähnlichkeitswert größer gleich 0.5 ist, angezeigt werden.

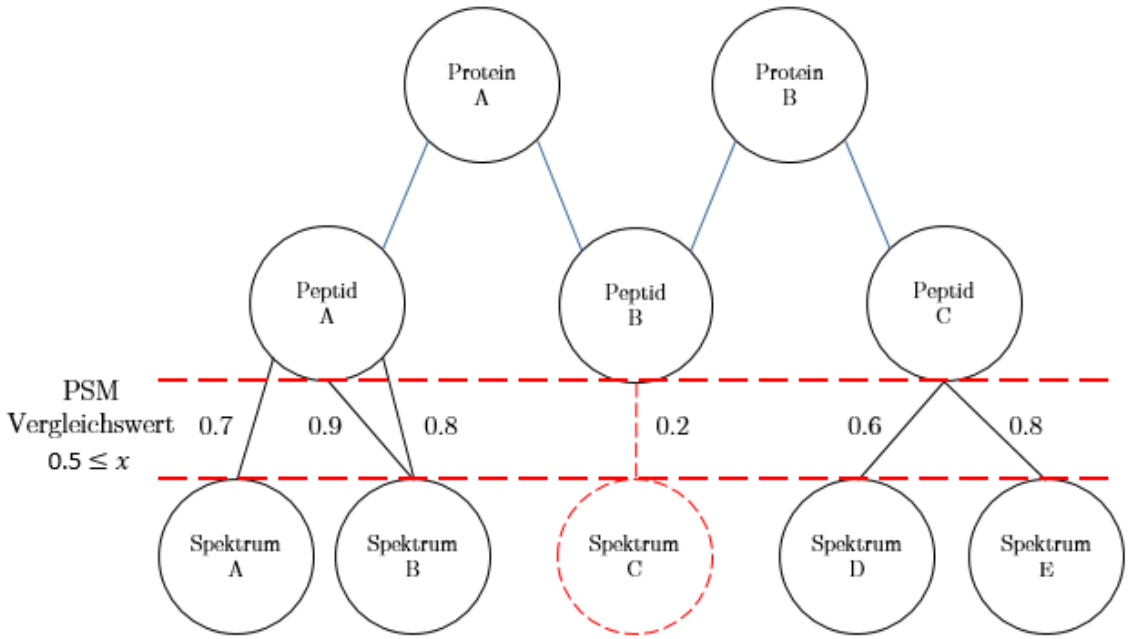


Abbildung 3.4: Darstellung einer Filterung nach PSM

Dadurch wird zum einen eine den Fokus auf das Wesentliche verlagernde, explorative Datenanalyse ermöglicht. Zum anderen ist es möglich, Übereinstimmungen mit geringem Ähnlichkeitsgrad bei der Quantifizierung zu ignorieren.

### Funktion-Taxonomie-Relation

Die Datenexploration unterstützend ist ein häufiger Schritt der Analyse, Proteine zu suchen, die einer bestimmten Taxonomie angehören oder gewisse molekulare Funktionen ausüben.

Abbildung 3.5 soll veranschaulichen, dass jedem Protein jeweils eine Funktion und Taxonomie zugeordnet werden kann. Somit kann z.B. der prozentuale Anteil einer Spezies der in den untersuchten Proben befindlichen mikrobiellen Gemeinschaften untersucht werden. Umgekehrt können auffällige Mengen von Proteinen schnell einer Spezies oder Funktion zugeordnet werden. Dadurch können beispielsweise im Feld der Diagnostik Rückschlüsse getroffen werden, in welchem Zustand sich die entnommene Probe befindet.

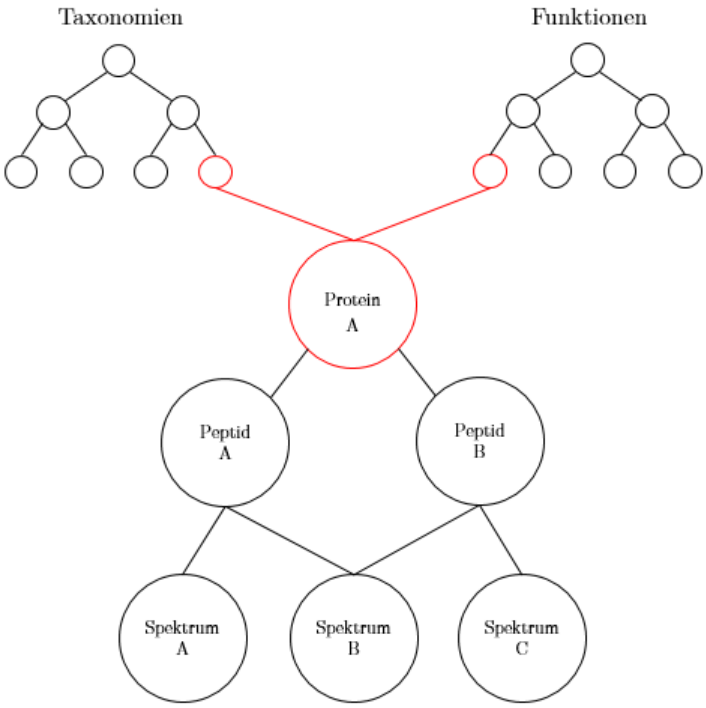


Abbildung 3.5: Darstellung einer Funktion-Taxonomie-Relation

## 3.3 Architekturentscheidung

Dieser Abschnitt soll die Wahl der Datenstruktur begründen und zugleich die erste der eingangs aufgestellten wissenschaftlichen Frage beantworten. Basierend auf den bereits genannten Anforderungen wird zunächst die Überführung der Daten in eine zweite Datenstruktur begründet. Infolgedessen wird anhand selbiger Anforderungen eine Entscheidung getroffen, welche Datenbank sich für die Abbildung und Analyse von Metaproteomdaten eignet.

Da die Anforderung aufgestellt wurde, dass die Datenstruktur sowie Informationen der Daten in Cassandra unverändert erhalten bleiben sollen, muss zunächst die Eignung Cassandras zur alleinigen Nutzung für Proteindatenbanksuche und Analyse betrachtet werden.

Neben dem Aufbau sind in Abbildung 3.2 auch die Zusammenhänge zwischen den Tabellen dargestellt. Die Beziehungen unter den Objekten der hoch-relationalen Metaproteomdaten stellen für die Analyse unerlässliche Informationen dar. Allerdings ist es nicht möglich, aus SQL bekannte Join-Operationen durchzuführen, da Cassandra das Konzept der Fremdschlüssel-Regeln nicht unterstützt.

Um die Relationen dennoch effizient abbilden zu können, müssen eigens zur Darstellung der Beziehungen zuständige Tabellen generiert werden. Aufgrund der Viele-zu-Viele-Beziehungen zwischen den Daten und mangelnder Indizierungsmöglichkeiten

Cassandras würden für jedes Beziehungspaar zwei Tabellen benötigt. Dies kann wie folgt anhand der Peptide-Protein-Beziehung verdeutlicht werden.

Ein Peptid kann ein Fragment verschiedener Proteine sein und ein Protein kann aus mehreren Peptiden bestehen. Es bedarf also jeweils einer Tabelle, um die zugehörigen Objekte auszulesen.

Dafür wird zunächst eine erste Relationstabelle angelegt, über die zu einer Peptid-Sequenz zugehörige Proteine abgefragt werden können.

Nun können über eine Abfrage alle Proteine einer bestimmten Peptid-Sequenz ausgelesen werden, da es sich bei der Peptid-Sequenz um den Primärschlüssel handelt. Gleichermaßen ist eine zweite Relations-Tabelle erforderlich, über die alle einem Protein entstammenden Peptide abgefragt werden können. Auf Kosten erhöhter redundanter Speicherung wäre es so möglich, die Beziehungen zwischen den Daten abzubilden und für Anfragen bereitzustellen.

Als Anforderung wurde definiert, dass Änderungen an den Analysedaten auch persistent gespeichert werden sollen. Modifikationen am Datensatz innerhalb Cassandras könnten die Folge haben, dass sämtliche Relationstabellen erneut erstellt werden müssten.

Eine weitere Möglichkeit wäre, geeignete Logik in die aufrufende Applikation auszulagern. Folglich würden nach jeder Anfrage alle Einträge der Tabellen durchsucht und verglichen werden, um mögliche Verbindungen aufzufinden. Aufgrund der Menge der Daten und den dafür erforderlichen Mengen an Arbeitsspeicher ist dieser Ansatz jedoch sehr ineffizient. Selbiger Grund spricht daher auch gegen In-Memory-Datenbanken, die hohe Geschwindigkeiten nur auf Kosten des Arbeitsspeichers ermöglichen.

Folglich ist die alleinige Verwendung von Cassandra für Proteindatenbanksuche und Analyse auszuschließen. Neben Cassandra wird deshalb ein zweites Datenbanksystem gesucht, welches aufgrund der Eigenschaften für die Analyse und persistente Speicherung der Datensätze geeignet ist. Daraus lässt sich die erste wissenschaftliche Frage ableiten:

**WF 1:** Welches Datenbankmodell eignet sich am besten für die Analyse von Metaproteomdaten?

Wie bereits geschildert, ist ein effizienter Umgang mit hoch-relationalen Daten und die Möglichkeit zur Visualisierung der Beziehungen zwischen den Objekten unumgänglich. Während Key-Value- sowie dokumentorientierte Datenbankmodelle auf Grund der

mangelnden Eigenschaft zur Darstellung von Beziehungen hinfällig sind, scheitern relationale Systeme an deren Schema-Restriktionen.

Graphdatenbanken zeichnen sich neben der variablen und schemafreien Struktur vor allem durch ihre Eigenschaft aus, Beziehungen effizient darstellen zu können. Da die Informationen der Objekte weiterhin in Cassandra zur Verfügung stehen, können die Graphen möglichst leichtgewichtig aufgebaut werden. Sollten nähere Informationen zu den Objekten vonnöten sein, können diese über eine Referenz aus Cassandra entnommen werden.

Dank der auf der Graphentheorie basierenden Struktur des Systems ist es nicht notwendig, starre Tabellenstrukturen mit aufwendigen Joins über Relationstabellen zu verbinden. Die Knoten des Graphen können vom Ausgangsknoten direkt über die Kanten traversiert werden. So ermöglicht es eine Graphdatenbank, die Vielzahl an Daten der Tabellen von Cassandra - gefüllt mit Metainformation - sinnvoll und effektiv zu nutzen.

Erfüllt wird auch die in Abschnitt 3.2 aufgestellte Anforderung, ein schemafreies Modell zu verwenden, welches das dynamische Hinzufügen und Entfernen von Eigenschaften und Beziehungen unterstützt. Änderungen an Knoten und Kanten können persistiert werden, wodurch ein kollaborativer Arbeitsablauf ermöglicht wird. Somit kann davon ausgegangen werden, dass die Analyse der Metaproteomdaten von der hohen Flexibilität der Graphtechnologie profitiert.

## 3.4 Allgemeines Konzept

Basierend auf der Anforderung, dass die Proteindatenbanksuche unverändert bleibt und weiterhin von MStream durchgeführt werden soll, folgt nun ein allgemeines Konzept der Erweiterung MStreams, wie die Realisierung der geforderten Analysemethoden erfolgen kann.

Diesbezüglich werden zunächst die dafür erforderlichen architekturellen Änderungen MStreams beleuchtet. Anschließend wird eine Graphstruktur entworfen, welche die Metaproteomdaten, wie sie bereits in Cassandra vorliegend sind, abbilden kann. Zuletzt soll gezeigt werden, wie eine mögliche Überführung und Transformation der Daten in die konzipierte Graphstruktur vollzogen werden kann.

### 3.4.1 Integration in MStream

In den vorherigen Abschnitten wurde gezeigt, dass eine Graphdatenbank für die Analyse von Massenspektrometern prädestiniert ist. Aufgrund der Anforderung,

MStream weiterhin für die Proteindatenbanksuche verwenden zu können und die verwendete Datenbank Cassandra beizubehalten, sieht das Konzept nun vor, das bestehende Cloud-System zu erweitern.

In Abbildung 3.6 ist die angestrebte Architektur MStreams dargestellt, in der die letztendlichen Funktionalitäten in drei voneinander getrennte Module aufgeteilt sind. Dies soll die geforderte unabhängige Funktionsweise und nötige Datenbankzugriffe der einzelnen Arbeitsschritte demonstrieren.

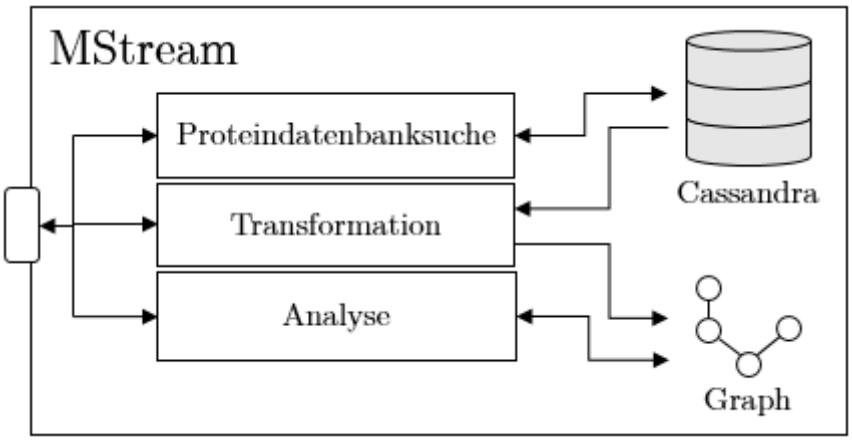


Abbildung 3.6: Konzeptionelle Erweiterung MStreams

Das Modul der Proteindatenbanksuche umfasst sämtliche Funktionen und Technologien, die im derzeitigen System vorhanden sind und greift ausschließlich auf die Daten innerhalb Cassandras zu. Das zweite Modul, welches in der Darstellung als "Transformation" bezeichnet wird, benötigt Zugriff auf beide Datenbanken und dient der Umwandlung der bei der Proteindatenbanksuche gesammelten Daten in die Graphstruktur. Die Pfeile sollen visualisieren, dass dieses Modul nur Lese-Operationen auf Cassandra beziehungsweise Schreib-Operationen auf dem Graphen ausführen wird. Als drittes Modul ist die Analyse der Daten zu nennen, welche lediglich die Graphdatenbank verwendet und dort Lese- sowie Schreib-Operationen durchführen soll.

Anwender, die über eine Schnittstelle mit dem System interagieren, kommunizieren dabei niemals direkt mit dem entsprechenden Modul oder der Datenbank. Koordiniert wird die Datenbankzuordnung programmatisch von MStream und folgt dabei stets der Modularisierung. Dadurch soll neben der Entwicklung auch die Wartung des Systems erleichtert werden. Eine solche Architektur wird als "Applikationskoordinierte Polyglotte Persistenz" bezeichnet [GR15].

### 3.4.2 Graphstruktur

Die Struktur des Graphen soll die nach der Proteindatenbanksuche erfassten Metaproteomdaten widerspiegeln und dazu dienen, bestmöglich die Beziehungen zwischen den Daten darzustellen. Um die Knoten voneinander unterscheiden zu können, wird ein Property-Graph benötigt. In einem Property-Graphen können Knoten und Kanten entsprechend betitelt und mit zusätzlichen Eigenschaften versehen werden. Die Bezeichnungen und Beziehungen der Knoten können dafür weitestgehend von denen der in Cassandra existierenden und in Abschnitt 3.1.2 beschriebenen Tabellen übernommen werden. So sind in Abbildung 3.7 die Knoten und Kanten für das Konzept des Graphen dargestellt.

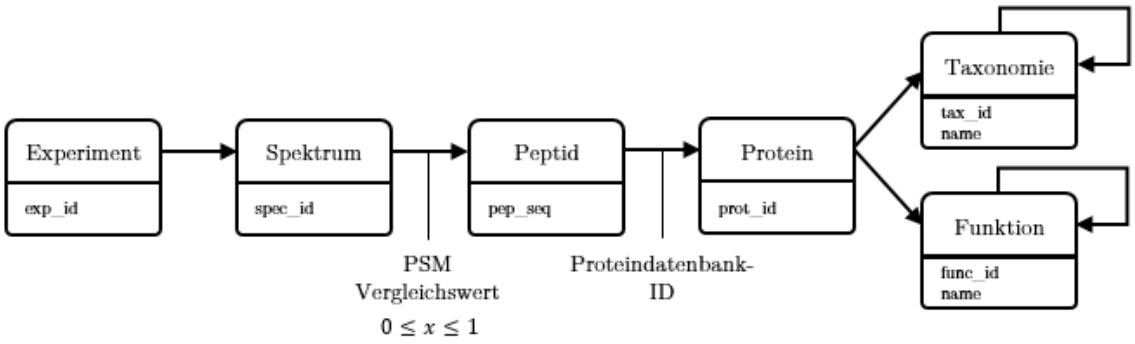


Abbildung 3.7: Konzeptionelle Datenstruktur der Graphdatenbank

Ein Experiment, in welchem einzig und allein die Referenz für den Eintrag der Experiment-Tabelle in Cassandra enthalten ist, stellt dabei den Wurzelknoten des Graphen dar. Mit diesem sind die Spektren, welche im entsprechenden Experiment gemessen wurden, über Kanten erreichbar. Ein Spektrum-Knoten enthält ebenfalls eine Referenz aus der betreffenden Tabelle sowie eine Anzahl an Kanten - die sich nach der Anzahl der Einträge in der PSM-Tabelle richtet - zu den passenden Peptid-Knoten. Für die bereits erwähnte PSM-Filterung wird in den Kanten zwischen Spektrum und Peptid der während der Proteindatenbanksuche bestimmte Vergleichswert festgehalten. Peptide - ebenfalls nur die entsprechende Referenz enthaltend - sind wiederum mit den Proteinen verbunden, zu denen sie sich zusammensetzen. Eine Referenz auf die Proteindatenbank, aus der das identifizierte Protein stammt, wird dabei in der entsprechenden Kante hinterlegt. Da auch für Proteine eine Tabelle in Cassandra existiert, sind diese ebenfalls diesbezüglich mit einer Referenz versehen. Zusätzlich dazu sind in der Protein-Tabelle auch die jeweiligen Verweise enthalten, welcher Taxonomie und Funktion ein Protein zugeordnet werden konnte. Diese Information kann je nach Datenquelle variieren und über eine Kante zum jeweiligen Knoten abgebildet werden. Taxonomie- und Funktions-Daten werden aus einer externen Quelle - wie NCBI und UniProt - bezogen. Beide Knotentypen besitzen jeweils eine

ID und den Namen der Taxonomie beziehungsweise Funktion, die sie darstellen. Da es jeweils einen Taxonomie- sowie Funktionsgraphen gibt, sind sie jeweils über eine Kante mit ihren Elternknoten verbunden.

### 3.4.3 Transformation der Daten

Um die Daten in die neue Datenstruktur zu überführen, wird ein Transformator benötigt, den es eigens zu konzipieren gilt. Dieser soll in der Lage sein, die experimentellen Ergebnisse aus Cassandra zu lesen und nicht-redundant in einen Graphen, in welchem sie durch Knoten und Kanten repräsentiert werden, zu überführen. Da es primär gilt, die Beziehungen der einzelnen Objekte abzubilden, sollen die gesammelten Informationen auf das wesentliche reduziert werden. So soll sichergestellt werden, dass einzelne Knoten möglichst wenige Metadaten enthalten, jedoch weiterhin identifizierbar bleiben und eine Referenz auf den zugehörigen Eintrag in Cassandra besitzen.

Zur Übergabe von Parametern und Informationen, welche Daten für den Aufbau des Graphen verwendet werden sollen, benötigt der Transformator eine Schnittstelle zur Kommunikation mit dem Benutzer. Dies ermöglicht eine gezielte Analyse der relevanten Daten, die beispielsweise in einem konkreten Experiment angefallen sind. Somit kann der Benutzer mehrere Graphen erstellen und anschließend die Daten verschiedener Experimente vergleichen.

Die Vorgehensweise des Transformators soll im Nachfolgenden mit Hilfe von Pseudocode-Beispielen schrittweise beschrieben werden. Dafür wird zunächst ausschließlich auf das Lesen und Schreiben der Daten aus Cassandra in den Graphen eingegangen. Vorausgesetzt wird, dass im Vorfeld eine Proteindatenbanksuche stattgefunden hat und die benötigten Daten zum Zeitpunkt der Transformation in Cassandra enthalten sind. Außerdem wird davon ausgegangen, dass bereits ein Taxonomie- sowie Funktions-Graph existieren, dessen Knoten später über einen Identifikator mit denen der Proteine verbunden werden können.

Quelltext 3.1 beschreibt den Ablaufplan zur Überführung der Daten. Als Übergabeparameter werden von der Funktion $transformCassandraToGraph$ zwei nichtredundante Objektmengen erwartet, die im Vorfeld vom Benutzer geliefert werden müssen. Dies umfasst zum einen die Identifikatoren der zu betrachtenden Experimente, zum anderen die IDs der Proteindatenbanken, deren Informationen für die Erzeugung der Protein-Daten verwendet werden sollen. Die Übergabe der Proteindatenbank-Identifikatoren ermöglicht es, gezielt die Protein-Daten einer gewünschten Datenquelle zu analysieren und nur Knoten für die darin enthaltenen Proteine zu erzeugen.

Zu Beginn (Zeilen 2 bis 4) werden die einzelnen Einträge aus den entsprechenden Tabellen gelesen und in nicht-redundanten Mengen gehalten, um anschließend (Zeilen 5 bis 7) schrittweise in Form von Knoten und Kanten in die neue Datenstruktur umgewandelt zu werden.

```
1 transformCassandraToGraph (expIds, proteinDbIds)
    inputs : Set of Experiment-IDs expIds, Set of Protein-DB-IDs
             proteinDbIds
2   psmSet ← readPSMs(expIds);
3   pepSet ← readPeptides(proteinDbIds, psmSet);
4   protSet ← readProteins(proteinDbIds, pepSet);
5   writePsmData(psmSet);
6   writePeptideData(pepSet);
7   writeProteinData(protSet);
8   return;
```

Quelltext 3.1: Ablauf des Transformators

### Lesen der PSM-Daten aus Cassandra (Quelltext 3.1 Zeile 2)

Quelltext 3.2 zeigt die Funktion, mit der der Transformator die PSM-Tabelle aus Cassandra ausliest. Darin enthalten sind die IDs der Experimente, die der dabei gemessenen Spektren und die beim Vergleich ermittelten Peptidsequenzen.

Übergeben wird dafür nur eine Liste von Strings, welche die IDs der gewünschten Experimente enthält, die für die Erstellung des Graphen in Betracht gezogen werden sollen. Für jeden gelesenen Eintrag wird also in Zeile 5 geprüft, ob es sich um eine der geforderten Experimente handelt. Wenn dem so ist, wird der Eintrag in ein entsprechendes Objekt umgewandelt und in einer nicht-redundanten Menge im Speicher gehalten, die am Ende von der Funktion als Rückgabewert zurückgegeben wird (Zeile 7).

### Lesen der Peptid-Daten aus Cassandra (Quelltext 3.1 Zeile 3)

Im nächsten Schritt - zu sehen in Quelltext 3.3 - wird die Peptide-Tabelle ausgelesen. Von Interesse sind hierbei die Proteine, zu denen die einzelnen Peptide zugeordnet sind. Informationen für die Peptid-Knoten wurden im vorherigen Schritt bereits in der PSM-Menge gesammelt. Das zuvor gesammelte Set der PSM-Objekte und eine Liste von IDs der Proteindatenbanken, deren Proteindaten zur Analyse dienen sollen, bilden dabei die Übergabeparameter.

Bevor die Peptide-Tabelle ausgelesen wird, werden zur Effizienzsteigerung zunächst die Peptid-Sequenzen aus der PSM-Menge in einer weiteren Menge separiert (Zeilen 4 - 6). Da mehreren Spektren dasselbe Peptid zugeordnet werden kann, werden durch

```
1  readPSMs (expIds)
       inputs  : Set of Experiment-IDs expIds
       output  : Set of PSM psmSet
2      psmSet ← empty Set of PSM;
3      resultSet ← READ psm from Cassandra;
4      foreach Row r in resultSet do
5          if r.exp_id ∈ expIds then
6              insert new PsmObject(r.exp_id, r.spec_id, r.pep_seq, r.psm_data)
                 into psmSet;
7      return psmSet;
```

Quelltext 3.2: Auslesen der PSM-Tabelle

```
1  readPeptides (psmSet, proteinDbIds)
       inputs  : Set of PSM psmSet, Set of Protein-DB-IDs proteinDbIds
       output  : Set of Peptides peptideSet
2      resultSet ← READ peptides from Cassandra;
3      peptideSet ← empty Set of peptides;
4      pepSeqSet ← empty Set of peptide sequences;
5      foreach PSM psm in psmSet do
6          insert psm.pep_seq into pepSeqSet;
7      foreach Row r in resultSet do
8          if r.pep_seq ∈ pepSeqSet then
9              protIds ← empty List of Strings;
10             foreach Protein-DB-ID dbId in proteinDbIds do
11                 protIds ← content of column named like dbId;
12             insert new PeptideObject(pep_seq, protIds) into peptideSet;
13     return peptideSet;
```

Quelltext 3.3: Auslesen der Peptide-Tabelle

diesen Schritt vermeintliche Redundanzen vermieden. Die neu erzeugte Menge dient im Anschluss der Vergleichsoperation mit den aus der Peptide-Tabelle gelesenen Sequenzen (Zeile 8).

Ist die gelesene Peptid-Sequenz in der verglichenen Menge enthalten, können die entsprechenden Spalten, die die Protein-IDs beinhalten, mit der übergebenen Liste an Proteindatenbank-Identifikatoren verglichen, ausgelesen und zusammen mit der Peptid-Sequenz als Objekt zusammengefasst werden (Zeilen 10 - 12). Die Protein-IDs enthaltenden Spalten sind nach den Protein-Datenbank-IDs benannt, aus denen die Informationen der Proteine entnommen wurden. Anschließend wird die Menge der gesammelten Peptid-Objekte von der Funktion zurückgegeben.

#### Lesen der Protein-Daten aus Cassandra (Quelltext 3.1 Zeile 4)

Die letzte der aus Cassandra auszulesenden Tabellen ist nun die der Proteine. Notwendig ist dieser Schritt, um weitere Informationen, wie zugeordnete Taxonomien und Funktionen, zu erhalten.

Neben der Menge der Peptide, die bereits gesammelt wurden, erhält die in Quelltext 3.4 dargestellte Funktion - wie auch beim Lesen der Peptid-Daten - eine Liste von IDs der Proteindatenbanken, deren Informationen für die Analyse verwendet werden sollen.

Auch hier wird wieder im ersten Schritt - um redundante Vergleiche zu vermeiden - vorerst eine neue Menge angelegt, die alle Protein-IDs enthalten soll (Zeilen 4 - 6).

Ist die Bedingung in Zeile 8 erfüllt und die ID des gelesenen Proteins in der Menge der Protein-IDs enthalten, beginnt eine weitere Schleife (Zeilen 10 - 12). Diese durchläuft die übergebene Liste der Protein-Datenbanken und liest die entsprechende Spalte aus, deren Name der der ID der Protein-Datenbank gleicht. In Zeile 13 werden die gesammelten Protein-Daten wiederum in einer weiteren Menge gesammelt um der Funktion am Ende als Rückgabewert zu dienen.

#### Schreiben der PSM-Daten in den Graphen (Quelltext 3.1 Zeile 5)

Nachdem die zu transformierenden Daten erfolgreich gelesen und in separaten Mengen gesammelt wurden, kann jetzt mit dem Erstellen des Graphen begonnen werden. Nötige Informationen für Experiment-, Spektrum- und Peptid-Knoten können dafür direkt aus der PSM-Menge bezogen werden, Protein-Knoten werden aus den Daten der Protein-Menge erstellt. Dieser Vorgang wird in Quelltext 3.5 veranschaulicht.

Zu Beginn wird die Menge der gesammelten PSM-Daten, die dem zu analysierenden Experiment entstammen, durchlaufen und in den Graphen überführt. Da jedes

```
1  readProteins (proteinDbIds, pepSet)
       inputs  : Set of Protein-DB-IDs proteinDbIds, Set of Peptides pepSet
       output  : Set of Proteins proteinSet
2      resultSet ← READ proteins from Cassandra;
3      proteinSet ← empty Set of proteins;
4      protIdSet ← empty Set of protein ids;
5      foreach Peptide pep in pepSet do
6          insert content ofpep.protIds into protIdSet;
7      foreach Row r in resultSet do
8          if r.protId ∈ protIdSet then
9              protData ← empty List of Strings;
10             foreach Protein-DB-ID dbId in proteinDbIds do
11                 rowProtData ← content of column named like dbId;
12                 insert rowProtData into protData;
13             insert new ProteinObject(r.prot_id, protData) into proteinSet;
14     return proteinet;
```

Quelltext 3.4: Auslesen der Protein-Tabelle

```
1  writePSMData (psmSet)
       inputs  : Set of PSM psmSet
2      foreach PSM psm in psmSet do
3          if psm.exp_id ∉ graph then
4              WRITE vertex experiment(psm.exp_id) into graph;
5          if psm.spec_id ∉ graph then
6              WRITE vertex spectrum(psm.spec_id) into graph;
7          if psm.pep_seq ∉ graph then
8              WRITE vertex peptide(psm.pep_seq) into graph;
9          WRITE edge experiment → spectrum;
10         WRITE edge spectrum − [psm.data] → peptide;
11     return;
```

Quelltext 3.5: Schreiben der PSM-Daten in den Graphen

Objekt der Menge die Information über ein Experiment, ein Spektrum sowie eine Peptidsequenz enthält, muss beim Iterieren geprüft werden, ob die Knoten bereits im Graphen vorhanden sind. Ist dies nicht der Fall, wird ein Knoten, der einzig und allein einen entsprechenden Identifikator enthält, erstellt (Zeilen 3 - 7).

Im Anschluss kann die eben beschriebene Redundanz genutzt werden, um die Knoten noch innerhalb der Schleife mit Kanten zu verbinden. Während eine Kante zwischen Experiment und Spektrum keine Eigenschaften besitzen soll (Zeile 9), wird in die Kante zwischen Spektrum und Peptid der ermittelte Ähnlichkeitswert, den die Vergleichsfunktion bei der Proteindatenbanksuche ermittelt hat, geschrieben (Zeile 10).

### Schreiben der Peptid-Daten in den Graphen (Quelltext 3.1 Zeile 6)

Nun gilt es, die Peptid-Knoten, die bereits mithilfe der PSM-Daten erstellt wurden, mit den noch zu erstellenden Protein-Knoten zu verbinden.

Die in der Peptid-Menge gesammelten Objekte bestehen nur aus der jeweiligen Peptidsequenz und einer Liste sich aus dieser Sequenz zusammensetzender Proteine. Wie in Quelltext 3.6 dargestellt, wird über die Menge der Peptide iteriert (Zeilen 2 - 7) und pro Objekt eine weitere Schleife durchlaufen, um an die zutreffenden Proteine zu gelangen. Existiert noch kein entsprechender Knoten, wird ein solcher erzeugt und mit der ID des Proteins versehen (Zeile 5). Unabhängig von der vorherigen Existenz wird daraufhin eine Kante zwischen Peptid und Protein erzeugt (Zeile 7). Der Kante wird dabei noch die ID der Proteindatenbank, welche sich in den Metadaten des Proteins befindet und bei der Analyse die Unterscheidung der Beziehungen erleichtern soll, hinzugefügt (Zeile 6).

```
1 writePeptideData (pepSet)
    inputs : Set of Peptides pepSet
2   foreach Peptide peptide in pepSet do
3     foreach Protein protein in peptide.prot_list do
4       if protein ∉ graph then
5         WRITE vertex protein(protein.id) into graph;
6       protDbId ← Protein-DB-ID from protein.protData;
7       WRITE edge peptide − [protDbId] → protein;
8   return;
```

Quelltext 3.6: Schreiben der Peptid-Daten in den Graphen

#### Schreiben der Protein-Daten in den Graphen (Quelltext 3.1 Zeile 7)

Da nun bereits alle Knoten erstellt wurden, besteht der letzte Schritt darin, die vorhandenen Proteine mit den im Vorfeld erstellten Taxonomie- und Funktions-Knoten mittels Kanten zu verbinden. Dargestellt ist dieser Schritt in Quelltext 3.7.

```
1 writeProteinData (protSet)
    inputs : Set of Proteins protSet
2   foreach Protein protein in protSet do
3     taxonomyId ← taxonomy id from protein.protData;
4     functionId ← function id from protein.protData;
5     WRITE edge protein → protein.taxonomy;
6     WRITE edge protein → protein.function;
7   return;
```

Quelltext 3.7: Schreiben der Protein-Daten in den Graphen

Dafür wird die Menge der übergebenen Proteine in einer Schleife durchlaufen und jedem Objekt die ID der entsprechenden Taxonomie sowie Proteinfunktion entnommen (Zeilen 3 - 4). Diese werden dann verwendet, um den Protein-Knoten mit den dadurch referenzierten Taxonomie- beziehungsweise Funktions-Knoten zu verbinden (Zeilen 5 - 6).

## 3.5 Zusammenfassung

In diesem Kapitel wurde das Konzept der Arbeit verdeutlicht. Dafür wurde zunächst das System MStream vorgestellt, auf dessen Grundlage aufgebaut und eine Applikationskoordinierte polyglotte Persistenz konzipiert wurde. Nachdem die Anforderungen an das System zusammengestellt und die Architekturentscheidung begründet wurde, konnte das allgemeine Konzept aufgestellt werden. Es wurde herausgearbeitet, welche Gründe dafür sprechen, eine Graphdatenbank in das System zu integrieren. Im Anschluss wurde beschrieben, wie die Struktur des Graphen aussehen und die Transformation der Daten durchgeführt werden soll.

# 4. Implementierung

Für die später folgende Evaluierung galt es nun, einen dem zuvor in Kapitel 3 beschriebenen Konzept zugrundeliegenden Transformator, der die Daten in eine Graphstruktur umwandeln soll, zu entwickeln. Dieses Kapitel beschreibt die Implementierung des Prototyps und geht dabei im Rahmen der Softwarearchitektur auf die umgesetzten Komponenten und deren Funktionen ein. Bevor die Architektur des Prototyps erläutert wird, soll zunächst auf die Wahl der verwendeten Technologien und Frameworks eingegangen werden.

## 4.1 Verwendete Technologien

Zu Beginn der Entwicklung stellte sich die Frage, welche Technologien und Frameworks für die Umsetzung eines entsprechenden Transformators gewählt werden sollen. Da der Großteil der Anwendungen, die bereits zum System MStream gehören, in Java entwickelt wurden, wurde diese Programmiersprache (Java 8) auch für den Transformator gewählt. Als Build-Management-Tool wurde, wie auch in den anderen Projekten MStreams, Apache Maven [MVM10] gewählt. Maven ermöglicht eine einfache Gruppierung zusammengehöriger Projekte und erleichtert das Verwalten von Build-Abhängigkeiten aus fremden Quellverzeichnissen.

Als Graphdatenbank fiel die Wahl auf Neo4j [VWA+14]. Neo4j ist eine Open Source Graphdatenbank und unterscheidet zwischen der frei verfügbaren Community-Edition und der Enterprise-Edition, welche kostenpflichtig ist und erhöhte Sicherheits- und Verfügbarkeitseigenschaften bietet. Für die Realisierung des Prototyps wurde die Enterprise-Edition 3.5.6 verwendet, welche in der ”Neo4j-Desktop”-Variante eine freie Entwicklungslizenz beinhaltet [Neo19].

Zu den zusätzlichen Komponenten der Enterprise-Edition gehört beispielsweise das sogenannte ”High-Availability-Clustering”, wodurch das Betreiben einer verteilten Graphdatenbank ermöglicht wird. Der Cluster stellt ein Master-Slave-Replication Set-up dar, in dem jede Instanz den vollständigen Datensatz beinhaltet. Erfolgreiche Write-Queries, welche auf dem Master ausgeführt werden, werden anschließend auf allen Slave-Instanzen repliziert. Zum einen ermöglicht dies eine horizontale Skalierung der Lese-Geschwindigkeit der Datenbank, zum anderen kann bei Ausfällen des Masters sofort ein Slave als neuer Master definiert werden, wodurch eine hohe Verfügbarkeit gewährleistet wird [Hun14].

Für die ACID-konformen Transaktionen stellt Neo4j eine eigene Anfragesprache namens ”Cypher” [FGG+18] zur Verfügung. Mit dieser SQL-ähnlichen Sprache können verschiedene Graph-Operationen ausgedrückt werden. Der Prototyp verwendet dafür den offiziellen Neo4j-Java-Treiber[1] in der Version 1.7.5, der das Ausführen von Cypher-Queries gegen einzelne und geclusterte Datenbanken ermöglicht.

Darüber hinaus wurde das Spark-Framework[2] 2.8.0 verwendet, wodurch die Anwendung auf einem eingebetteten Jetty-Server[3] bereitgestellt und über eine HTTP-Schnittstelle erreicht werden kann. Für die Verbindung zu Cassandra wird Datastax-Java-Driver-for-Apache-Cassandra[4] in der Version 4.0.0 verwendet und erlaubt es, Operationen über die Cassandra Query Language (CQL) auszuführen [LM10].

## 4.2 Softwarearchitektur

In diesem Abschnitt soll die Architektur des entwickelten Prototyps unter Verwendung der im vorherigen Abschnitt aufgezeigten Technologien beschrieben werden, der die Transformation der Daten aus Cassandra in eine Neo4j-Graphdatenbank erlaubt. Die grobe Softwarearchitektur des Transformators ist in Abbildung 4.1 dargestellt und unterteilt sich in die im Folgenden erklärten Schichten und Komponenten.

**Kommunikationsschicht**

Die Kommunikationsschicht beschreibt die REST-API, die dem Benutzer eine Schnittstelle zur Kommunikation über HTTP-Requests ermöglicht. So wurde beispielsweise ein POST-Endpunkt implementiert, der die Experiment-ID des zu analysierenden Experiments und eine Liste von Protein-Datenbank-IDs, um die entsprechenden Protein-Daten aus Cassandra auszulesen, erwartet. Dadurch wird eine gezielte Analyse eines Experiments ermöglicht. Wird der besagte Endpunkt angesprochen und die nötigen Informationen übergeben, beginnt die Transformation der Daten.

[1] https://github.com/neo4j/neo4j-java-driver
[2] http://sparkjava.com
[3] https://github.com/eclipse/jetty.project
[4] https://github.com/datastax/java-driver

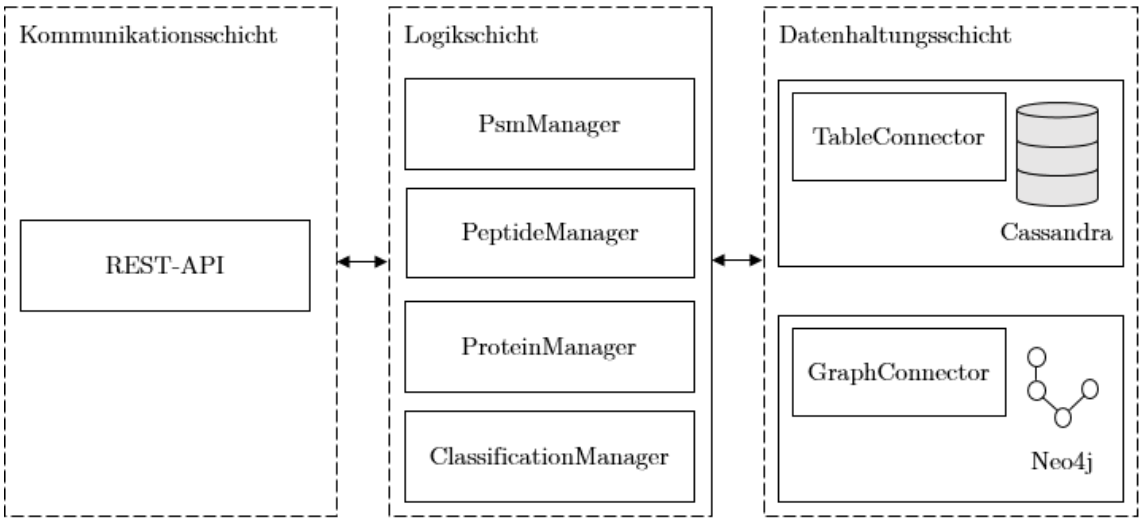


Abbildung 4.1: Architektur des entwickelten Prototyps

**Logikschicht**

In der Logikschicht sind die Komponenten vereint, welche die eigentliche Umwandlung der Daten vornehmen, die Befehle der Kommunikationsschicht entgegennehmen und entsprechende Datenbank-Anfragen für die Datenhaltungsschicht vorbereiten. Neben zahlreichen Helferklassen enthält die Schicht folgende wichtige Klassen:

- PsmManager: Diese Klasse ist dafür zuständig, die Daten, welche in der PSM-Tabelle in Cassandra enthalten sind, in nicht-redundanten Mengen zu sammeln. Zu diesen gehören die Informationen über die Experimente und die experimentellen Spektren mit den übereinstimmenden theoretischen Peptiden. Zusätzlich koordiniert sie die Erstellung der Kanten zwischen Experiment-, Spektrum- und Peptid-Knoten. Die Kante zwischen Spektrum- und Peptid-Knoten erhält zusätzlich noch den ermittelten Ähnlichkeitswert, der ebenfalls in der PSM-Tabelle hinterlegt ist.

- PeptideManager: In dieser Klasse werden die aus der PSM-Menge entnommenen Peptid-Sequenzen dafür verwendet, um für jede Sequenz ein Peptid-Objekt anzulegen, in welchem die Sequenz mit den dazugehörigen Protein-IDs gesammelt wird. Außerdem koordiniert sie die Erstellung der Kanten von Peptid- zu Protein-Knoten, nachdem letztere erstellt wurden. In diesen wird die Protein-Datenbank-ID hinterlegt, aus der die Protein-Informationen entstammen.

- ProteinManager: Diese Klasse organisiert nach Einholen der nötigen Protein-Informationen die Erstellung der Protein-Knoten aus den Informationen der Peptid-Objekt-Menge sowie die Verbindung dieser zu den im Vorfeld erstellten Taxonomie- und Funktion-Knoten.

- ClassificationManager: Die aus NCBI beziehungsweise UniProt entnommenen Informationen über Taxonomien und Protein-Funktionen, werden in dieser Klasse zu CSV-Dateien umgewandelt, um die Erstellung der entsprechenden Knoten und Kanten zu beschleunigen. Neo4j stellt eine Import-Funktionalität bereit, mittels derer CSV-Dateien batchweise ausgelesen und als Knoten beziehungsweise Kanten importiert werden können.

**Datenhaltungsschicht**

Die Komponenten der Datenhaltungsschicht koordinieren die Verbindung zur jeweiligen Datenbank und übersetzen die vorbereiteten Datenbank-Anfragen in die jeweilige Datenbanksprache, sodass die benötigten Daten aus der Cassandra-Datenbank gelesen beziehungsweise Knoten und Kanten in die Neo4j-Datenbank geschrieben werden. Zu dieser Schicht gehören die folgenden Klassen:

- TableConnector: Diese Klasse ist für den Aufbau einer Verbindung und der Kommunikation zu Cassandra zuständig. Dafür nimmt sie Informationen der Komponenten der Logikschicht entgegen und formuliert die nötigen CQL-Anfragen, um Tabellen aus Cassandra auszulesen.
- GraphConnector: Analog zum ”TableConnector” ist der ”GraphConnector” dafür zuständig, eine Verbindung zu Neo4j aufzubauen und die Cypher-Queries zu formulieren, um Knoten und Kanten im Graphen zu erstellen. Dazu stellt die Klasse verschiedene Funktionen bereit, die es ermöglichen, einzelne sowie beliebige Mengen von Knoten- und Kanten-Kombinationen anzulegen.

Die eingesetzte Schichtenarchitektur erlaubt eine logische Trennung der modularen Komponenten nach ihren Aufgabenfeldern und verringert die Komplexität innerhalb der Anwendung. Eine Schicht interagiert stets nur mit ihren direkten Nachbarn, wodurch eine hohe Flexibilität gewährleistet wird, da bei möglichen Änderungen nur angrenzende Schichten angepasst werden müssen. Neben einer erleichterten Entwicklung und Wartung der Anwendung wird auch das Hinzufügen neuer Funktionen oder Austauschen einzelner Komponenten vereinfacht [JPM02]. Wird beispielsweise vorgesehen, eine andere Graphdatenbank zu verwenden, sind nur Änderungen in der Datenhaltungsschicht nötig.

## 4.3 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie eine prototypische Umsetzung des im Vorfeld aufgestellten Konzepts eines Transformators, der Daten aus Cassandra ausliest und

in eine Graphdatenbank umwandelt, umgesetzt werden kann. Dafür wurden zunächst die verwendeten Technologien sowie Frameworks genannt und im Anschluss die Architektur inklusive der wichtigsten Komponenten und deren Funktionen beschrieben. Der entwickelte Prototyp ist in der Lage, die in Abschnitt 3.4.3 konzipierte Überführung der Daten auszuführen und somit einen Graphen zu erzeugen, auf dem die in Abschnitt 3.2.2 beschriebenen Analysemethoden durchgeführt werden können. Der im Rahmen der Arbeit entwickelte Prototyp erfüllt alle aufgestellten funktionalen Anforderungen.

# 5. Evaluierung

Um die Effizienz des entwickelten Prototyps zu untersuchen, soll nun eine Evaluierung folgen. Diese dient der Untersuchung der zweiten wissenschaftlichen Frage dieser Arbeit, ob die polyglotte Persistenz performanter als ein einzelnes Datenbanksystem ist. In den nun folgenden Kapiteln, angefangen mit der Motivation und den Zielen der Evaluierung, wird die dafür nötige Vorbereitung beschrieben, gefolgt von der Durchführung sowie einer abschließenden Diskussion der aufgekommenen Ergebnisse.

## 5.1 Motivation

Ob ein Produkt - in diesem Fall MStream unter der Verwendung zweier Datenbanksysteme zur Prozessierung und Analyse von Metaproteomdaten - seinen Anforderungen gerecht wird und in der Praxis verwendet werden kann, sollte im Vorfeld eine praxisnahe Evaluierung durchgeführt werden. Diese dient der Generierung von Erfahrungen und Informationen, welche gesammelt und im Nachhinein bewertet werden können. Daraus können dann wiederum Entscheidungen getroffen werden, ob das besagte Produkt verwendet und wie es verbessert werden kann [Sto04].

Die Evaluierung soll nun mithilfe von verschiedenen Datenbankabfragen aufzeigen, ob die verwendeten Datenbanken für ihr jeweiliges Anwendungsgebiet geeignet sind.

Für ein aussagekräftiges Ergebnis werden die Abfragen außerdem noch gegen eine relationale Datenbank ausgeführt. Die Anwendung "MetaProteomeAnalyzer" (MPA) [MBH+15], ein auf MySQL basierendes System, welches Proteindatenbanksuche und Metaproteomanalyse vereint, soll die dafür nötige Datenbank bereitstellen. Gewählt wurde dieses System, da es bereits für die Verwendung von Metaproteomdaten optimiert ist und die Analyse mithilfe von im Vorfeld angelegten Verbundansichten (Views) beschleunigt.

Ob und wie sich die Kombination von Cassandra und Neo4j gegen das auf MySQL basierende Produktivsystem MPA behaupten kann, soll in den folgenden Abschnitten untersucht werden.

## 5.2 Vorbereitung

Bevor die der Evaluierung dienenden Methoden durchgeführt werden können, müssen die Datenbanken zunächst mit repräsentativen Daten gefüllt werden. Da es sich um die Analyse von Metaproteomdaten handelt, wird eine im Vorfeld absolvierte Proteindatenbanksuche in MStream sowie MPA mit vorausgesetzt. Für eine faire Ausgangssituation wird in beiden Systemen der gleiche Experimentdatensatz, welcher aus einer Probe einer Biogasanlage gewonnen wurde, verwendet. Der Datensatz liegt in Form einer MGF-Datei vor und enthält 27.542 Spektren. Für die Proteindatenbanksuche benötigte Proteine werden einer FASTA-Datei aus UniProt entnommen, die Informationen über insgesamt 554.982 Proteine enthält.

Da sich die Implementierung der Vergleichsfunktion in MStream von der in MPA unterscheidet, können die Ergebnisse und somit die Einträge in den PSM-Tabellen voneinander abweichen. Dies ist an dieser Stelle jedoch nicht zu vermeiden und muss bei der Auswertung der Evaluierungsergebnisse berücksichtigt werden. Außerdem ist bei den Ergebnissen zu berücksichtigen, dass möglicherweise eine weitaus höhere Lese- sowie Schreib-Performanz von Cassandra erreicht werden könnte, wenn man ein Cluster aus mehreren Datenbank-Knoten verwenden würde. Für die Lese-Geschwindigkeit von Neo4j gilt das gleiche. Ein Clusterzugriff ist aufgrund von Hardware-Limitierungen im Rahmen dieser Arbeit nicht möglich.

Nach der Proteindatenbanksuche befinden sich in den Tabellen von MStream und MPA alle Informationen, die für die Evaluierung benötigt werden. Im Anschluss wird der prototypisch entwickelte Transformator verwendet, um die Daten innerhalb MStreams von der Cassandra-Datenbank in die Neo4j-Graphdatenbank zu überführen. Die zusätzlich dafür nötigen Informationen über Funktionen und Taxonomien der Proteine werden aus NCBI beziehungsweise UniProt entnommen.

Zur Kommunikation mit den Datenbanken wurden separate Java-Anwendungen entwickelt, welche die Abfragen an die entsprechende Datenbank ausführen.

## 5.3 Durchführung

Die Anforderungen an die verwendete Datenbank variieren zwischen der Proteindatenbanksuche und der anschließenden Analyse. Während bei erstgenanntem

ausschließlich READ- und INSERT-Operationen durchgeführt werden, sind es bei der Analyse vorwiegend READ- und UPDATE-Operationen. Um praxisnahe Szenarien abzubilden und eine aussagekräftige Evaluierung zu ermöglichen, wurden Datenbankabfragen formuliert, welche die genannten Operationen einschließen.

Dazu wurden folgende Abfragen formuliert:

**A 1:** Anzahl Spektren pro Experiment (READ)

**A 2:** Anzahl Spektren pro Protein (READ)

**A 3:** Anzahl Spektren pro Protein mit PSM-Filter (READ)

**A 4:** Anzahl Spektren pro Protein mit Taxonomie-Funktion-Kombination (READ)

**A 5:** Ausblenden aller Spektren (UPDATE)

**A 6:** Einfügen eines Proteins (INSERT)

Die oben genannten Abfragen werden für jede der drei Datenbanken durchgeführt. Die für die Kommunikation mit den Datenbanken entwickelten Java-Anwendungen sind ausschließlich dafür vorgesehen, die Verbindung zur Datenbank aufzubauen und eine Abfrage auszuführen. Um ein substanzielles Ergebnis zu erhalten, wird je Datenbank (Neo4j, Cassandra und MySQL) für jede Abfrage ein Zeit-Durchschnittswert ermittelt, der im Rahmen einer definierten Anzahl von Durchläufen gebildet wird.

Um potenzielle Fehlerquellen, die aus einer möglichen Konkurrenz um Hardwareressourcen resultieren können, auszuschließen, werden die Anwendungen nacheinander ausgeführt.

Um eine hohe Performanz zu erreichen, wurden die Abfragen so konzipiert, dass die Ergebnisse auf einem möglichst effizienten Weg erzielt werden konnten. Exemplarisch wird dies unter anderem durch die für bestimmte Abfragen optimierten Views der MySQL-Datenbank erreicht. Außerdem werden stets ausschließlich die für die Abfrage relevanten Daten selektiert.

Alle Messungen wurden auf folgender Hardwarekomposition durchgeführt:

- CPU: Intel® Core™i5-4670K @3,4 GHz
- RAM: 16GB
- HDD: 1TB @7200 UpM

## 5.4 Erwartungen und Hypothesen

Bei den zu evaluierenden Datenbanken handelt es sich um drei stark differente Systeme, welche jeweils ihre eigenen Vor- und Nachteile aufweisen. Erwartet wird, dass sich bezüglich der formulierten INSERT-Operation die Vorteile von Cassandra bei der Proteindatenbanksuche zeigen und bei den komplexen Beziehungsabfragen sich die Überlegenheit von Neo4j aufgrund der Graphdatenbank-Eigenschaften herausstellt. Demgegenüber wird erwartet, dass die relationale Datenbank den Anforderungen - dem Prozessieren und Analysieren von Metaproteomdaten - genauso gerecht werden kann, allerdings bei geringerer Effizienz. Ausgehend von den genannten Erwartungen lassen sich folgende Hypothesen ableiten:

**Hypothese 1:** Keines der untersuchten Systeme ist bei der Ausführung aller Operationen am schnellsten

**Hypothese 2:** Neo4j ist bei allen untersuchten Messungen schneller als MySQL

Mithilfe der gesammelten, im nächsten Abschnitt beschriebenen Ergebnisse, sollen die aufgestellten Hypothesen in der anschließenden Diskussion ausgewertet werden.

## 5.5 Ergebnisse

Die nun folgende Auswertung der Evaluierung ist aus Gründen der Übersicht in mehrere Abschnitte aufgeteilt. Zunächst werden die Messergebnisse - entsprechend der in Abschnitt 5.3 genannten Operationen - nacheinander präsentiert und diskutiert. Eine sinnvolle Darstellung macht es in einigen Abbildungen erforderlich, die Werte der y-Achse zur Basis 10 zu logarithmieren.

### 5.5.1 A1: Anzahl Spektren pro Experiment

Die Ergebnisse der ersten Messung (Abbildung 5.1) veranschaulichen die durchschnittliche Antwortzeit der einzelnen Datenbanken für eine Zählung aller in einem Experiment gemessenen Massenspektren. Es wird gezeigt, dass beide Vertreter MStreams deutlich schneller als die MySQL-Datenbank die Abfrage bearbeiten. Neo4j benötigt hierbei sogar durchschnittlich nur 3,9 Millisekunden, was 0,03% der Abfragezeit von MySQL entspricht (13.240,4 Millisekunden im Durchschnitt). Mit durchschnittlich 106,2 Millisekunden erzielte Cassandra Ergebnisse in im Gegensatz zu MySQL weitaus kürzerer Zeit.

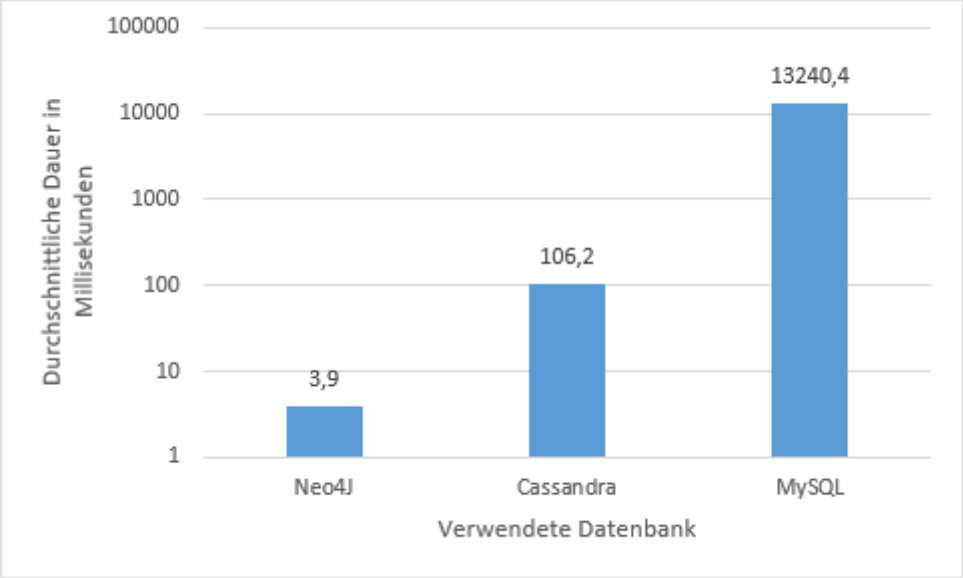


Abbildung 5.1: Messergebnis Anzahl Spektren pro Experiment

### 5.5.2 A2: Anzahl Spektren pro Protein

In Abbildung 5.2 werden die Ergebnisse der Messung aller Spektren, die einem gegebenen Proteins zugeordnet werden können, veranschaulicht. MySQL liegt dabei mit durchschnittlich 0,5 Millisekunden zwar knapp vor Neo4j mit durchschnittlich 1,2 Millisekunden, ist jedoch mehr als doppelt so schnell. Cassandra hingegen benötigte für diese Abfrage durchschnittlich 95.503,5 Millisekunden. Aufgrund dieses enorm hohen Zeitaufwandes wurde bei nachfolgenden Abfragen mit erhöhter Komplexität von Messungen auf Cassandra abgesehen, da das Ergebnis in keinem vernünftigen Verhältnis zu den beiden anderen Datenbanken steht.

Begründet ist dieser enorme Zeitaufwand von Cassandra in der Art und Weise, wie die Spektren eines Proteins ermittelt werden. Da keine Tabelle existiert, in der Spektren und Proteine beieinander liegen, muss zunächst die gesamte Peptid-Tabelle ausgelesen werden, um die Peptid-Sequenzen zu erhalten, die dem gegebenen Protein entstammen. Diese Peptid-Sequenzen werden dann gegen alle in der PSM-Tabelle befindlichen Peptid-Sequenzen verglichen, um die geforderten Spektren zählen zu können.

Im MySQL liegen die Referenzen für Protein- und Spektrum-Ids in MySQL in der angefragten View direkt in einer Reihe und können Dank der Indizes effizient gelesen werden. Neo4j hingegen traversiert beginnend vom Protein-Knoten mit der gegebenen Protein-ID die mittels Kanten verbundenen Knoten und gelangt über die anliegenden Peptide an die entsprechenden Spektren.

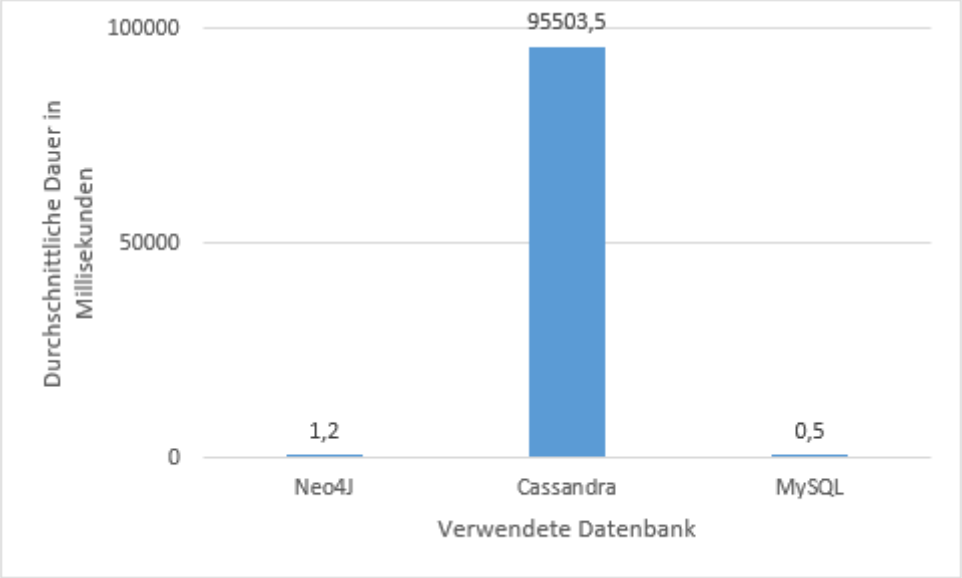


Abbildung 5.2: Messergebnis Anzahl Spektren pro Protein

Die durchschnittlichen Messungen bei Neo4j und MySQL sind in diesem Fall geringer als bei der vorherigen Operation aus Abschnitt 5.5.1, da die Ergebnismenge - bestehend aus durchschnittlich weniger als 5 Spektren pro Protein - weitaus geringer.

### 5.5.3 A3: Anzahl Spektren pro Protein mit PSM-Filter

Auch in Abbildung 5.3, in der das Ergebnis einer um einen PSM-Filter erweiterten Spektrum-Zählung dargestellt wird, ist MySQL mit durchschnittlich 0,52 Millisekunden knapp drei mal schneller als Neo4j mit durchschnittlich 1,4 Millisekunden. Wie bereits in Abschnitt 5.5.2 erwähnt, entfällt die Messung der Abfragen der Cassandra-Datenbank komplett.

Die Messergebnisse zeigen, dass der angewendete PSM-Filter, welcher die vorherige in Abschnitt 5.5.2 durchgeführte Anfrage um eine weitere WHERE-Klausel erweitert, nur geringe Auswirkungen auf die durchschnittliche Dauer der Anfragen hat. Allerdings ist der prozentuale Anstieg bei Neo4j mit durchschnittlich 16% zur in Abschnitt 5.5.2 durchgeführten Operation höher, als bei MySQL (Anstieg von 4%).

### 5.5.4 A4: Anzahl Spektren pro Protein mit Taxonomie-Funktion-Kombination

Mit den Messergebnissen, welche in Abbildung 5.4 gezeigt werden, in denen erneut Cassandra ausgenommen ist, wird ein erheblich größerer Zeitaufwand bei der MySQL-Abfrage verdeutlicht. Hierbei ging es um eine komplexe Abfrage einer Anzahl von Spektren, die einem gegebenen Protein zugeordnet werden können, welches wiederum

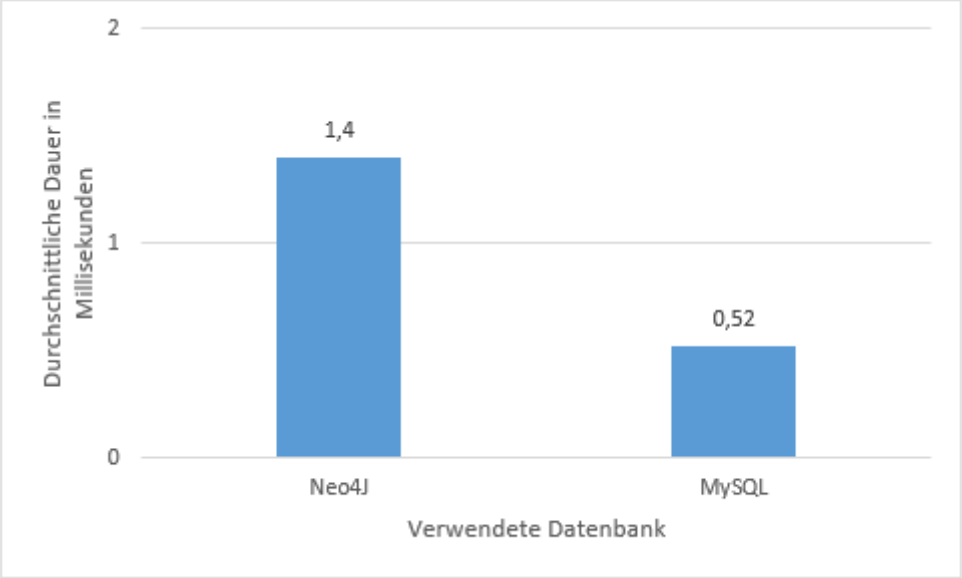


Abbildung 5.3: Messergebnis Anzahl Spektren pro Protein mit PSM-Filter

einer im Vorfeld bestimmten Taxonomie sowie Proteinfunktion unterliegt. Während Neo4j nach bereits durchschnittlich 1,8 Millisekunden eine Antwort liefert, benötigt MySQL dafür im Durchschnitt mit 16.897,6 Millisekunden unverhältnismäßig länger.

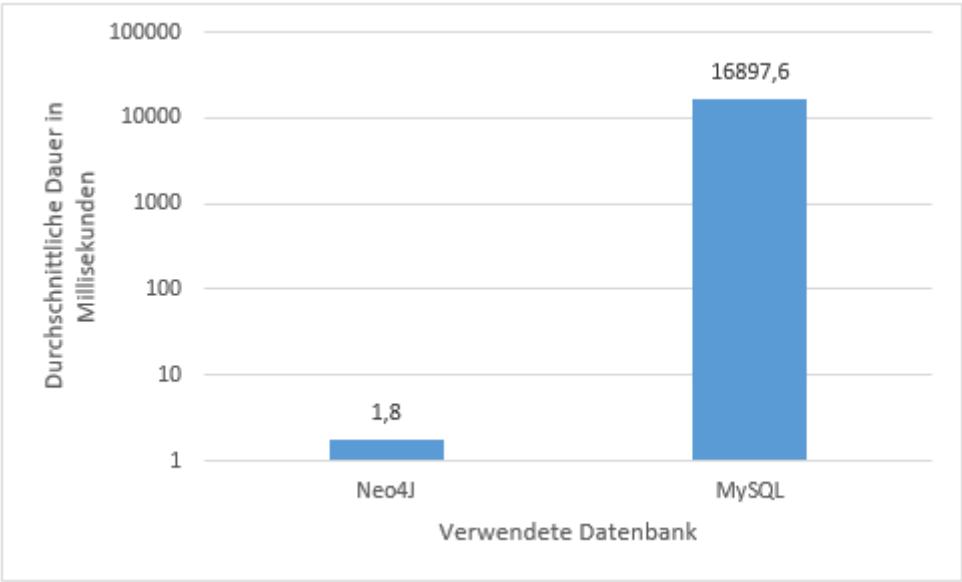


Abbildung 5.4: Messergebnis Anzahl Spektren pro Protein mit Taxonomie-Funktion-Kombination

Der enorme Unterschied zu den vorherigen Operationen ist dadurch zu erklären, dass die für diese Zählung nötige Anfrage eine String-Vergleichs-Operation für jeden Eintrag durchführen muss. In Neo4j scheint dieser Mehraufwand deutlich weniger Auswirkungen auf die Performanz zu haben als bei MySQL.

### 5.5.5 A5: Ausblenden aller Spektren

Um die Darstellung von Messerergebnissen einer UPDATE-Operation auf den Datensatz geht es in Abbildung 5.5. Deutlich wird auch hier der hohe Zeitaufwand von durchschnittlich 16.843,3 Millisekunden bei MySQL. Mit Neo4j ist die Operation nach bereits durchschnittlich 125,3 Millisekunden prozessiert.

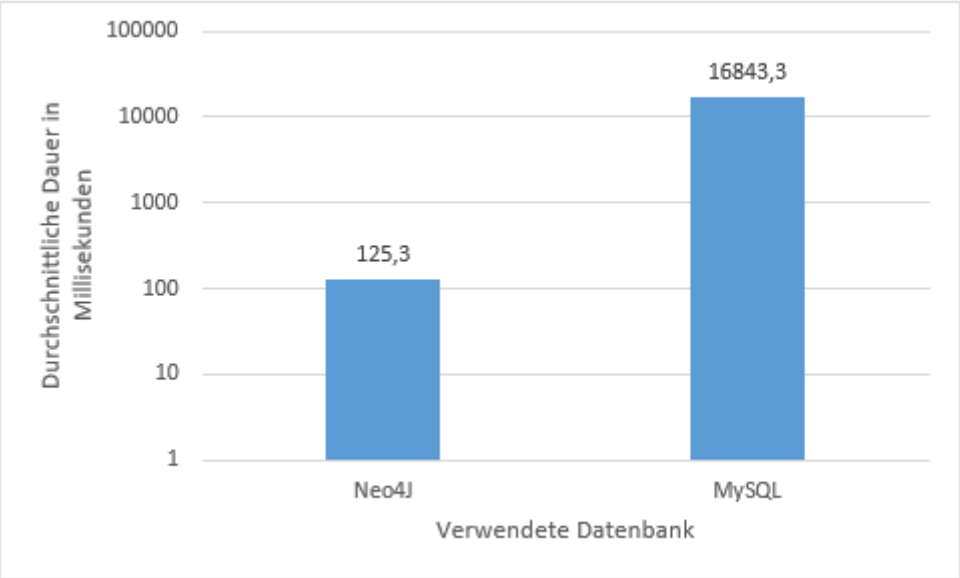


Abbildung 5.5: Messergebnis Ausblenden aller Spektren

Der große durchschnittliche Zeitaufwand MySQLs liegt vermutlich darin begründet, dass aufgrund fehlender Indizies die Daten nicht sortiert werden können und demzufolge alle vorliegenden Reihen verglichen werden müssen.

### 5.5.6 A6: Einfügen eines Proteins

Geht es darum, große Datenmengen zu schreiben, wird mit Abbildung 5.6 deutlich, wie effektiv der Einsatz einer Cassandra-Datenbank sein kann. In durchschnittlich 3,4 Millisekunden wird ein Protein einschließlich Proteinsequenz und Metadaten in die Datenbank geschrieben. Für die gleiche Informationsmenge benötigt Neo4j mit durchschnittlich 45,17 Millisekunden bereits mehr als das zehnfache der Zeit. MySQL beendet den Schreibvorgang sogar erst nach durchschnittlich 123,7 Millisekunden und ist damit um ein Vielfaches langsamer.

Eine Vermutung, wieso MySQL auch hier langsamer als Neo4J ist, ist, dass das Einfügen einer Reihe dazu führt, dass sämtliche zu der Tabelle gehörenden Views upgedatet werden müssen. Cassandra kann hier vermutlich aufgrund der BASE-Eigenschaften sehr gute Ergebnisse erzielen.

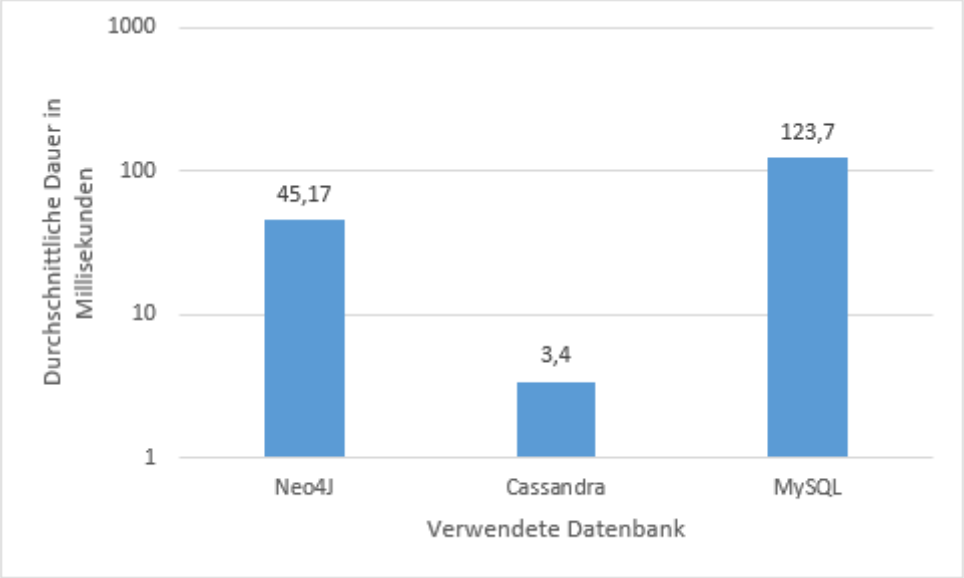


Abbildung 5.6: Messergebnis Einfügen eines Proteins

## 5.6 Diskussion der Hypothesen (WF2)

Nach dem Veranschaulichen der verschiedenen Ergebnisse sollen die aufgestellten Hypothesen im Rahmen einer Diskussion untersucht werden. Darüber hinaus gilt es, die zweite wissenschaftliche Frage, ob die polyglotte Persistenz effizienter als ein einzelnes System ist, zu beantworten. Effizienz ist eine vielschichtige Kategorie hinsichtlich beispielsweise der Ressourcen CPU-Auslastung, Speicherintensität oder beanspruchte Zeit. Im Rahmen der durchgeführten Untersuchungen wurde ausschließlich die Performanz in Bezug auf die benötigte Ausführungszeit untersucht.

**Hypothese 1: Keines der untersuchten Systeme ist bei der Ausführung aller Operationen am schnellsten**

Die Ergebnisse zeigen, dass die untersuchten Systeme unterschiedliche Stärken und Schwächen aufweisen. Zu einem großen Teil resultiert dies aus der Architektur und Struktur der jeweiligen Datenbank. Zusammengefasst sind die Ergebnisse in Tabelle 5.1, wobei sehr gute Ergebnisse mit einem doppelten Pluszeichen beschrieben werden, akzeptable Ergebnisse mit nur einem Pluszeichen gewertet werden. Während Neo4j bei allen READ-Operationen sehr gute Ergebnisse erzielt, ist MySQL dafür nur bei den einfacheren Operationen in der Lage. Cassandra wiederum kann als einzige Datenbank beim Schreiben sehr gute Ergebnisse erzielen.

Aus den Ergebnissen lässt sich ableiten, dass keines der untersuchten Systeme bei der Ausführung aller Operationen am schnellsten ist. Die Annahme der Hypothese 1 gilt somit als bestätigt.

| | Neo4j | Cassandra | MySQL |
|---|---|---|---|
| A1: Anzahl Spektren pro Experiment | ++ | + | - |
| A2: Anzahl Spektren pro Protein | ++ | - | ++ |
| A3: Anzahl Spektren pro Protein mit PSM-Filter | ++ | - | ++ |
| A4: Anzahl Spektren pro Pr. mit Tax.-Fkt.-Komb. | ++ | - | - |
| A5: Ausblenden aller Spektren | + | - | - |
| A6: Einfügen eines Proteins | + | ++ | - |

Tabelle 5.1: Zusammenfassung und Vergleich der durchschnittlichen Messergebnisse

**Hypothese 2: Neo4j ist bei allen untersuchten Messungen schneller als MySQL**

Es muss festgestellt werden, dass diese Hypothese mit den durchgeführten Messungen nicht bewiesen werden konnte. Um die durchschnittlichen Messergebnisse von Neo4j und MySQL zusammenzufassen, soll Tabelle 5.2 dienen.

| | Neo4j | MySQL |
|---|---|---|
| A1: Anzahl Spektren pro Experiment | 3,9 | 13240,4 |
| A2: Anzahl Spektren pro Protein | 1,2 | 0,5 |
| A3: Anzahl Spektren pro Protein mit PSM-Filter | 1,4 | 0,52 |
| A4: Anzahl Spektren pro Pr. mit Tax.-Fkt.-Komb. | 1,8 | 16897,6 |
| A5: Ausblenden aller Spektren | 125,3 | 16843,3 |
| A6: Einfügen eines Proteins | 45,17 | 123,7 |

Tabelle 5.2: Durchschnittliche Messdauer in Millisekunden von Neo4j und MySQL

Es gibt durchaus Operationen, in denen MySQL ein schnelleres Ergebnis liefert als Neo4j, beispielsweise bei der Zählung von Spektren eines gegebenen Proteins, wie in Abschnitt 5.5.2 und Abschnitt 5.5.3 erläutert. Auch wenn es sich bei diesen Messungen um eine nahezu dreifache Zeitersparnis durch die Verwendung von MySQL handelt, ist diese Zeitersparnis als marginal anzusehen. Dies würde eine Entscheidung für MySQL und gegen Neo4j in keinster Weise rechtfertigen, denn andere Messungen wiederum belegen, dass Neo4J bei komplexeren Operationen - wie das Zählen von Spektren eines Proteins mit bestimmter Taxonomie-Funktion-Kombination - weitaus schneller Ergebnisse erzielt. Hierbei stehen sich Werte von durchschnittlich 1,8 Millisekunden, die von Neo4j erzielt wurden und durchschnittlich 16.897,6 Millisekunden bei MySQL gegenüber.

Die Hypothese 2 gilt somit als widerlegt.

**Fazit**

Die vorangegangene Evaluierung inklusive der diskutierten Hypothesen lässt an dieser Stelle die Beantwortung der eingangs aufgestellten zweiten wissenschaftlichen Frage zu. Diese lautet wie folgt:

**WF 2:** Ist die polyglotte Persistenz performanter als ein einzelnes Datenbanksystem?

Wie in der Diskussion der ersten Hypothese dargestellt wurde, war keines der Systeme in allen durchgeführten Messungen durchweg herausragend. Daraus lässt sich ableiten, dass eine Kombination mehrerer Systeme zielführender ist, wenn es darum geht, eine stets performante Ausführung der Operationen zu gewährleisten. Damit werden die jeweiligen Schwächen der einen Datenbank durch die Stärken einer anderen kompensiert. Diese Kompensation schafft die Voraussetzung für eine erhöhte Performanz.

Aufgrund der Ergebnisse der durchgeführten Messungen, stellt sich nun die Frage, welche Datenbankkombination den gewünschten Anforderungen am ehesten gerecht wird. Da jedoch - wie bereits in Abschnitt 3.2 beschrieben wurde - Cassandra als feststehende Größe anzusehen ist, gilt es, herauszufinden, welches Datenbanksystem die Schwächen von Cassandra am effektivsten kompensiert. Die Messergebnisse untermauern, dass eine Graphdatenbank eine effektive Kombination zu Cassandra darstellt.

## 5.7 Zusammenfassung

In diesem Kapitel wurde eine Evaluierung, welche die Eignung dreier untersuchter Datenbanken hinsichtlich verschiedener für die Metaproteomprozessierung typischer Operationen, aufzeigen sollte. Dafür wurden zunächst die dafür nötigen Vorbereitungsschritte erklärt, die Durchführung beschrieben und anhand von Erwartungen Hypothesen aufgestellt, die es in der Auswertung zu diskutieren galt. Die Evaluierung zeigte, dass die unterschiedlichen Anforderungen, welche durch Proteindatenbanksuche und Analyse der Daten an die jeweils verwendete Datenbank gestellt werden, sehr unterschiedlich bedient wurde. So zeigt sich, dass durch den gemeinsamen Einsatz von Cassandra und Neo4j - entsprechend der jeweiligen Aufgabenstellung - die Vorteile der Datenbanken optimal kombiniert werden können, um stets ein Ergebnis in kürzest möglicher Zeit zu erzielen.

# 6. Verwandte Arbeiten

An dieser Stelle sollen verwandte Arbeiten vorgestellt werden, die sich ebenfalls mit dem Thema ”Polyglotte Persistenz” auseinandersetzen.

**NextGen data persistence pattern in healthcare: Polyglot persistence [PS13]**

Der Text zeigt auf, wie vielseitig und komplex die Anforderungen sind, denen Datenbanken in Gesundheitsinformationssystemen (KIS) gerecht werden müssen. Beispielsweise mache es der Wert, den die Daten an sich für ein Unternehmen haben, notwendig, dass es sich um einen stabilen Datenspeicher handelt, der gleichermaßen strukturierte Daten wie unstrukturierte unterstützt, da die Daten aus einer Vielzahl von Quellen stammen und in unterschiedlichen Formaten vorliegen. Kriterien wie schnelles Lesen und oder Schreiben, Verfügbarkeit, Konsistenz, Sicherheitsanforderungen und viele mehr können, dem Autor folgend, nicht von einer Datenbank alleine erfüllt werden, da jede einzelne so konzipiert ist, dass sie zwar einen spezifischen Bereich optimal abdeckt, bei Verwendung für alle anfallenden Belange jedoch ungenügende Lösungen hervorbringen würde. Diese Ausführung zeigt, dass die Sachlage eine polyglotte Persistenz notwendig macht. In der vorliegenden Arbeit wird eine Kombination aus SQL- und NoSQL-Speichern vorgeschlagen, wobei der Verfasser bei seiner prototypischen Implementierung nicht ausschließlich zwei, sondern mehrere Datenbanken verknüpft. Angesichts der KIS-Module, die im Verlauf der Arbeit beschrieben werden, können die wichtigsten Gesundheitsinformationen in mehrere Datenspeicher getrennt werden, beispielsweise Sessions-Daten in einer Redis wie NoSQL-Datenbank, Transaktionsdaten wie Krankenhausabrechnungsinformationen in einem traditionellen RDBMS-Speicher, Daten für Arztempfehlungen, verwandte Krankenhäuser und ähnliche Beziehungen in einer Grafikdatenbank wie Neo4j usw.

Auf diese Weise wird sichergestellt, dass jeweils die Datenbank verwendet wird, die optimal auf die gerade zu bedienende Anforderung zugeschnitten ist.

**A Smart Polyglot Solution for Big Data in Healthcare [KR15]**

Auch diese Abhandlung zeigt auf, wie breit die Anforderungen an Datenspeicher in Gesundheitsinformationssystemen sind und dass dies polyglotte Persistenz notwendig macht. Das Konzept, das die Autoren vorlegen, heißt PolyglotHIS und sieht innerhalb der Persistenz sogenannte Agenten, also Techniken künstlicher Intelligenz vor, die eine Zusammenarbeit zwischen den beteiligten Datenspeichern mit unterschiedlichen Semantik-, Darstellungs- und Abfragemechanismen erreichen. Die Arbeit des PolyglotHIS ist in mehrere Schritte zu untergliedern. Es gibt einen Nutzer, hier Mitglied des medizinischen Fachpersonals, der eine Anfrage an das integrierte System stellt, die unter Umständen Daten von mehr als einem Datenspeicher braucht. Des Weiteren die Vermittlungsschicht bestehend aus den kooperierenden Agenten, die die Unterabfragen an die unterschiedlichen Datenspeicher formuliert, die Verarbeitung und Rückgabe der Ergebnisse an das Reduktionsmittel und schlussendlich die Zusammenführung der Teilergebnisse. Das Resultat wird dem Benutzer in dem von ihm gewünschten Format angezeigt. In der entwickelten Konstellation erfüllt jeder Agent eine ihm zugewiesene Aufgabe. Es erscheint zu umfassend, hier alle Agenten nebst ihrer Bestimmung aufzuführen, daher sollen nur einige wenige bespielhaft Erwähnung finden. So gibt es einen grafischen Interaktionsagenten, der es dem Benutzer ermöglicht, Anfragen über die Representation State Transfer-API durchzuführen und die Ergebnisse einzusehen, einen Query-Mapping-Agent der Subabfragen an die Zieldatenspeicher sendet und die Zwischenergebnisse in einem Teilergebnisspeicher festhält uva. Eine Schlüsselrolle kommt dem Data Store Apropos Agent zu, der, laut Verfasser, einen primären Einstiegspunkt in die Mediationsschicht des Systems darstellt und spontan bestimmt, welcher Datenspeicher für die Beantwortung der gegebenen Anfragen genutzt werde muss. Anhand des Datalog-Agenten, der mit dem Data Store Apropos Agent zusammenarbeitet, werden die dafür notwendigen Informationen aus der Knowledge Base of Data (KBoD) in Form von Datalog-Abfragen entnommen. KBoD sammelt sämtliche semantischen Beschreibungen der Inhalte aller einzelnen Mitgliedsdatenspeicher. Abschließend soll der Lernagent erwähnt werden, der es möglich macht, nach der kostengünstigsten Abfrage zu suchen.

**Application of polyglot persistence to enhance performance of the energy data management systems [PA14]**

Diese Arbeit, die sich ebenfalls mit den Vorteilen polyglotter Persistenz befasst, weicht grundsätzlich durch das Wirkungsfeld Energiemanagement von den oben angeführten ab. Energy Data Management (EDM) Utilities zielt darauf, die Bedürfnisse

von Energiedatenmanagementverbreitung und –analyse abzudecken. Es geht, wie auch in den anderen Fachtexten, um Daten, die unter anderem aufgrund ihrer unterschiedlichen Beschaffenheit aus mehreren Quellen stammen. Das Energiemanagement muss vielerlei Ansprüchen gerecht werden - beispielsweise Skalierbarkeit, schnelles Lesen und Schreiben, Angaben zum Stromverbrauch an Kunden und die Pflege und Verwaltung von Kundenprofilen - um nur einige wenige zu nennen. Die technischen Vorteile, die eine Nutzung von polyglotter Persistenz auf dem Gebiet des Energiedatenmanagements mit sich bringt, werden ausführlich aufgeführt. Exemplarisch sei hier die Flexibilität bei der Speicherung und Nutzung unstrukturierter Daten und die Unterstützung von Echtzeitanwendungen für schnellere Analysen genannt. Die Verwendung von traditionellen RDBMS- ebenso wie von NoSQL-Datenbanken ermöglicht es, so der Autor, sich die Vorteile der jeweiligen Systeme zunutze zu machen. Während zum Beispiel für Abrechnungs- und Tarifdetails, gelieferte elektrische Energie, Umsatz etc. in der Regel weiterhin herkömmliche RDBMS zur Speicherung verwendet werden, werden Zeitreihendaten, also Daten, die entweder turnusmäßig oder aufgrund diverser Auslöser erfasst werden, von Smart-Metern und anderen Komponenten des Smart-Grid ausgegeben werden. Diese Daten können analysiert werden um Ausfälle zu erkennen, Rechnungen zu prognostizieren und vieles mehr. Im weiteren Verlauf widmet sich der Autor unter anderem dem Business Wert, der dadurch, dass das System Analysen, Prognosen etc. ermöglicht, in seinen Augen erheblich ist.

# 7. Fazit

Durch Identifizierung und Quantifizierung von Proteinen ist es möglich, einen direkten Einblick in die Phänotypen von Mikroorganismen zu gewinnen, sowie funktionelle Rollen und Interaktionen einzelner Spezies zu verstehen. Während die Proteomik - ein Forschungsfeld der Biologie - sich ausschließlich mit der Untersuchung von Proteinen eines einzelnen Organismus beschäftigt, geht die Metaproteomik einen Schritt weiter und untersucht die Proteine mikrobieller Gemeinschaften.

Mit dem Einsatz eines Massenspektrometers - einem heute etablierten, wichtigen Teil des Arbeitsprozesses - werden Häufigkeiten der Molekülmassen von Proteinfragmenten gemessen, die sich nach entsprechender Aufbereitung in den zu untersuchenden biologischen Proben befinden. Bereits bekannte Proteine stehen in Proteindatenbanken zur Verfügung und werden mit neuen, sich aus den Experimenten ergebenden Massenspektren, verglichen. Um dies zu ermöglichen, werden die Sequenzen der theoretischen Proteine ebenfalls fragmentiert und anhand bekannter molekularer Massen in theoretische Spektren umgewandelt. Aufgrund der möglichen enormen Proteinfülle einer biologischen Probe und der sich aus der technischen Entwicklung der Massenspektrometer ergebenden, stets steigenden Datenmenge, macht sich eine effiziente Datenhaltung zwingend erforderlich.

MStream - ein in Abschnitt 3.1 vorgestelltes Cloud-System - wurde entwickelt um die beschriebene Proteindatenbanksuche auf Grundlage von Big-Data-Technologien zu effektivieren. Die eingesetzte spaltenorientierte Cassandra-Datenbank bietet aufgrund ihrer Struktur hervorragende Schreib- und Leseperformanz für große Datenmengen. Das Fehlen spezieller Relationstabellen sowie die nicht-vorhandene Möglichkeit der Join-Operationen, wirken sich negativ aus im Hinblick auf die Analyseverfahren der hoch-relationalen Metaproteomdaten.

Beispielhaft für diese Analyseverfahren wurden sowohl die markierungsfreie Quantifizierung - ein unkomplizierter Ansatz zur Schätzung der Proteinhäufigkeit - als auch die Klassifizierung der Proteine nach Taxonomien und molekularen Funktionen, vorgestellt. Für solche Analysen macht es sich hauptsächlich erforderlich, die Beziehungen zwischen den Daten zu betrachten.

Da Cassandra im Rahmen dieser Forschung als gegebenes System anzusehen ist, aber eine geeignete Abbildung der Beziehung zwischen den Daten sowie komplexe Betrachtungen sehr ineffizient sind, ließen sich zwei wissenschaftliche Fragen ableiten.

**WF 1:** Welches Datenbankmodell erfüllt die Strukturanforderungen zur Analyse von Metaproteom-Daten am ehesten?

**WF 2:** Ist die polyglotte Persistenz performanter als ein einzelnes Datenbanksystem?

Im Zuge des in Kapitel 3 entwickelten Konzepts, welches das Ziel hatte, eine Lösung für die Erweiterung von MStream um die Analyse von Metaproteomdaten zu finden, wurde gezeigt, dass eine Graphdatenbank alle in Abschnitt 3.2 genannten Anforderungen erfüllt. Damit wurde gleichzeitig auch die **WF1** beantwortet.

Basierend auf diesen Anforderungen wurde in Abschnitt 3.4.3 das Konzept eines Transformators erstellt, der die von MStream im Zuge der Proteindatenbanksuche ermittelten Daten in eine dafür konzipierte Graphstruktur überführt. Auf dieser Grundlage wurde in Kapitel 4 der Prototyp eines Transformators entwickelt, der die Daten in eine Neo4j-Graphdatenbank umwandelt.

Zur Beantwortung der **WF2** wurde in Kapitel 5 anschließend eine Evaluierung durchgeführt. Um zu zeigen, ob eine Datenbankkombination im Kontext von Metaproteomdaten sinnvoll ist, wurden die durchschnittlichen Antwortzeiten typischer Analyseabfragen von drei verschiedenen Datenbanken - Cassandra, Neo4j und MySQL - miteinander verglichen. Die daraus resultierten Ergebnisse aus Abschnitt 5.5 sollen hier noch einmal zusammengefasst werden.

Tabelle 7.1 zeigt die Stärken und Schwächen der drei untersuchten Datenbanken. Die Ergebnisse zeigen, dass MySQL zwar in A2 und A3 durchschnittlich die besten Messzeiten aufweist, allerdings bei den komplexeren Anfragen deutlich schlechtere Werte aufweist, als Neo4j. Neo4j erzielte mit im Durchschnitt stets kurzen Antwortzeiten für alle READ-Operationen (A1 - A4) durchweg akzeptable Ergebnisse.

Es wird auch deutlich, dass Cassandra für die durchgeführten READ-Operationen (A1 - A4) nicht geeignet ist. Die durchschnittliche Antwortzeit bei A2 ist bereits nicht vertretbar hoch, weswegen auf die Messung komplexerer Anfragen verzichtet

| | Cassandra | Neo4j | MySQL |
|---|---|---|---|
| A1: Anzahl Spektren pro Experiment | 106,2 | **3,9** | 13240,4 |
| A2: Anzahl Spektren pro Protein | 95503,5 | **1,2** | 0,5 |
| A3: Anzahl Spektren pro Protein mit PSM-Filter | - | **1,4** | 0,52 |
| A4: Anzahl Spektren pro Pr. mit Tax.-Fkt.-Komb. | - | **1,8** | 16897,6 |
| A5: Ausblenden aller Spektren | - | **125,3** | 16843,3 |
| A6: Einfügen eines Proteins | **3,4** | 45,17 | 123,7 |

Tabelle 7.1: Durchschnittliche Messdauer in Millisekunden von Cassandra, Neo4j und MySQL

wurde. Im Gegensatz dazu erfolgt das Schreiben von Daten mit höchster Performanz (A6), wodurch Cassandra als gegebenes System durchaus seine Berechtigung hat. Dies lässt die Schlussfolgerung zu, dass die **WF2** positiv beantwortet werden kann.

Betrachtet man die gezeigten Ergebnisse und die in Abschnitt 3.2 genannten Anforderungen, zeigt sich, dass die Kombination aus Cassandra und Neo4j sehr effektiv ist und somit diese polyglotte Persistenz einen vielversprechenden Ansatz für die Prozessierung und Analyse von Metaproteomdaten darstellt.

### Zukünftige Arbeiten

Eine logische Fortführung kann darin bestehen, eine komfortable Benutzeroberfläche für die Analyse zu entwickeln, um dem Nutzer auf diese Weise die Daten direkt in interaktiven Diagrammen anzeigen zu lassen. Diese sollte außerdem eine Benutzerverwaltung beinhalten.

Vorstellbar ist auch eine Optimierung des Transformators selbst. So wäre es möglich, die aus Cassandra ausgelesenen und im Arbeitsspeicher gehaltenen Datenmengen in Form einer CSV-Datei in Neo4j zu importieren, um schnellere Schreibgeschwindigkeiten zu erzielen.

Da es sich bei dem Testsystem, auf dem die Evaluierungen durchgeführt wurden, nicht um ein praxisadäquates System handelt und Cassandra sowie Neo4j nicht-verteilt getestet wurden, ist eine weitere Evaluierung unter faireren Bedingungen denkbar.

Die Möglichkeit darüber nachzudenken, ob ganz auf einen Transformator verzichtet werden kann. Stattdessen könnte der Graph direkt über den Spark-Job, der innerhalb MStreams die Vergleiche zwischen den aus Kafka konsumierten Massenspektren mit den theoretischen Spektren durchführt, erstellt werden. Somit wäre es möglich, bereits während der Messung einen Graphen entstehen zu lassen.

Alle aufgezeigten fortführenden Arbeiten haben wie bereits diese Arbeit zum Ziel, die Proteindatenbanksuche und Analyse noch schneller und komfortabler zu machen, um die kostbare Ressource Zeit effektiver zu nutzen.

# Literaturverzeichnis


[AM03] Ruedi Aebersold and Matthias Mann. Mass spectrometry-based proteomics. *Nature*, 422:198–207, 04 2003. (zitiert auf Seite 6)

[AXF+12] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. *Sigmetrics Performance Evaluation Review - SIGMETRICS*, 40, 06 2012. (zitiert auf Seite 12)

[Bre00] Eric Brewer. Towards robust distributed systems. page 7, 01 2000. (zitiert auf Seite 11)

[Bre12] Eric Brewer. Cap twelve years later: How the rrules"have changed. *Computer*, 45(2):23–29, Feb 2012. (zitiert auf Seite 11)

[Cai18] A.J. Cain. Taxonomy. Website, 2018. Online erhältlich unter https://www.britannica.com/science/taxonomy; abgerufen am 27. August 2019. (zitiert auf Seite 8)

[Cat11] Rick Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27, May 2011. (zitiert auf Seite 10)

[CB04] Robertson Craig and Ronald C. Beavis. TANDEM: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, 02 2004. (zitiert auf Seite 1 und 7)

[Con18] The UniProt Consortium. UniProt: a worldwide hub of protein knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, 11 2018. (zitiert auf Seite 7)

[DAC10] Mark W Duncan, Ruedi Aebersold, and Richard M Caprioli. The pros and cons of peptide-centric proteomics. *Nature biotechnology*, 28(7):659—664, July 2010. (zitiert auf Seite vii, 6, 7 und 8)

[Dar13] Costel Darie. Mass spectrometry and proteomics: Principle, workflow, challenges and perspectives. *Mod Chem appl*, 06 2013. (zitiert auf Seite 6)

[DCL18] Ali Davoudian, Liu Chen, and Mengchi Liu. A survey on nosql stores. *ACM Computing Surveys*, 51:1–43, 04 2018. (zitiert auf Seite 12)

[Deu12] Eric W. Deutsch. File formats commonly used in mass spectrometry proteomics. *Molecular & Cellular Proteomics*, 2012. (zitiert auf Seite 6 und 7)

[Fed11] Scott Federhen. The NCBI Taxonomy database. *Nucleic Acids Research*, 40(D1):D136–D143, 12 2011. (zitiert auf Seite 8)

[FGG+18] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1433–1445, New York, NY, USA, 2018. ACM. (zitiert auf Seite 38)

[Gar13] Nishant Garg. *Apache Kafka*. Packt Publishing, 2013. (zitiert auf Seite 18)

[GMK+04] Lewis Y. Geer, Sanford P. Markey, Jeffrey A. Kowalak, Lukas Wagner, Ming Xu, Dawn M. Maynard, Xiaoyu Yang, Wenyao Shi, and Stephen H. Bryant. Open mass spectrometry search algorithm. *Journal of Proteome Research*, 3(5):958–964, 2004. (zitiert auf Seite 7)

[GR15] Felix Gessert and Norbert Ritter. Polyglot persistence. *Datenbank-Spektrum*, 15(3):229–233, 2015. (zitiert auf Seite vii, 13, 14 und 28)

[HJ11] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*, pages 336–341, Dec 2011. (zitiert auf Seite 12)

[HLLP15] J.S. Hwang, S Lee, Y Lee, and S Park. A selection method of database system in bigdata environment: A case study from smart education service in korea. *International Journal of Advances in Soft Computing and its Applications*, 7:9–21, 01 2015. (zitiert auf Seite 14)

[HPCG13] Robert L. Hettich, Chongle Pan, Karuna Chourey, and Richard J. Giannone. Metaproteomics: Harnessing the power of high performance mass spectrometry to identify the suite of proteins that control metabolic activities in microbial communities. *Analytical Chemistry*, 85(9):4203–4214, 2013. (zitiert auf Seite 1)

[HSS11] J. Han, M. Song, and J. Song. A novel solution of distributed memory nosql database for cloud computing. In *2011 10th IEEE/ACIS International Conference on Computer and Information Science*, pages 351–355, May 2011. (zitiert auf Seite 9)

[HSZ+17] Robert Heyer, Kay Schallert, Roman Zoun, Beatrice Becher, Gunter Saake, and Dirk Benndorf. Challenges and perspectives of metaproteomic data analysis. *Journal of Biotechnology*, 261:24 – 36, 2017. Bioinformatics Solutions for Big Data Analysis in Life Sciences presented by the German Network for Bioinformatics Infrastructure. (zitiert auf Seite 1)

[Hun14] M. Hunger. *Neo4j 2.0: Eine Graphdatenbank für alle*. Schnell + kompakt. Entwickler.Press, 2014. (zitiert auf Seite 38)

[JABH09] Daniel J. Abadi, Peter Boncz, and Stavros Harizopoulos. Column oriented database systems. *PVLDB*, 2:1664–1665, 08 2009. (zitiert auf Seite 13)

[JHGJ11] Jing Han, Haihong E, Guan Le, and Jian Du. Survey on nosql database. In *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366, Oct 2011. (zitiert auf Seite 9 und 10)

[JPM02] Stefan Jablonski, Ilia Petrov, and Christian Meiler. Ein konzeptionelles architekturrahmenswerk für web-anwendungen. In *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen - Promise 2002, 9.-11. Oktober 2002, Potsdam*, pages 175–187, 2002. (zitiert auf Seite 40)

[Kle19] Manuel Kleiner. Metaproteomics: Much more than measuring gene expression in microbial communities. *mSystems*, 4(3), 2019. (zitiert auf Seite 5 und 7)

[KR15] K. Kaur and R. Rani. A smart polyglot solution for big data in healthcare. *IT Professional*, 17(6):48–55, Nov 2015. (zitiert auf Seite 56)

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010. (zitiert auf Seite 18 und 38)

[MBH+15] Thilo Muth, Alexander Behne, Robert Heyer, Fabian Kohrs, Dirk Benndorf, Marcus Hoffmann, Miro Lehtevä, Udo Reichl, Lennart Martens, and Erdmann Rapp. The metaproteomeanalyzer: A powerful open-source software suite for metaproteomics data analysis and interpretation. *Journal of Proteome Research*, 14(3):1557–1565, 2015. (zitiert auf Seite 8 und 43)

[MBR+13] Thilo Muth, Dirk Benndorf, Udo Reichl, Erdmann Rapp, and Lennart Martens. Searching for a needle in a stack of needles: challenges in metaproteomics data analysis. *Mol. BioSyst.*, 9:578–585, 2013. (zitiert auf Seite 8)

[MC15] Anil Kumar Meher and Yu-Chie Chen. Polarization induced electrospray ionization mass spectrometry for the analysis of liquid, viscous and solid samples. *Journal of Mass Spectrometry*, 50(3):444–450, 2015. (zitiert auf Seite 6)

[MGAOI14] Mohamed Mohamed, Obay G. Altrafi, and Mohammed O. Ismail. Relational vs. nosql databases: A survey. *International Journal of Computer and Information Technology (IJCIT)*, 03:598, 05 2014. (zitiert auf Seite 9 und 11)

[MRML07] Pierre-Alain Maron, Lionel Ranjard, Christophe Mougel, and Philippe Lemanceau. Metaproteomics: A new approach for studying functional microbial ecology. *Microbial ecology*, 53:486–93, 05 2007. (zitiert auf Seite 5)

[MVdJW+17] Bart Mesuere, Felix Van der Jeugt, Toon Willems, Tom Naessens, Bart Devreese, Lennart Martens, and Peter Dawyndt. High-throughput metaproteomics data analysis with unipept: A tutorial. *Journal of Proteomics*, 171, 05 2017. (zitiert auf Seite 8)

[MVM10] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Apache Maven*. Alpha Press, 2010. (zitiert auf Seite 37)

[Neo19] Inc. Neo4j. Neo4j produkte, 2019. Online erhältlich unter https://neo4j.com/product/; abgerufen am 30. August 2019. (zitiert auf Seite 37)

[NLIH13] Cory Nance, Travis Losser, Reenu Iype, and Gary Harmon. Nosql vs rdbms - why there is room for both. *SAIS 2013Proceedings*, 2013. (zitiert auf Seite 9 und 13)

[PA14] S. Prasad and S. B. Avinash. Application of polyglot persistence to enhance performance of the energy data management systems. In *2014 International Conference on Advances in Electronics Computers and Communications*, pages 1–6, Oct 2014. (zitiert auf Seite 56)

[PM00] Akhilesh Pandey and Matthias Mann. Proteomics to study genes and genomes. *Nature*, 405:837–846, 2000. (zitiert auf Seite 6)

[PPCC99] David N. Perkins, Darryl J. C. Pappin, David M. Creasy, and John S. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567, 1999. (zitiert auf Seite 7)

[PRS+11] Rabi Prasad Padhy, Manas Ranjan, Patra Suresh, Chandra Sekhar Satapathy, and Oracle India Pvt. Rdbms to nosql: Reviewing some next-generation non-relational database's. 2011. (zitiert auf Seite 11)

[PS13] S. Prasad and M. S. N. Sha. Nextgen data persistence pattern in healthcare: Polyglot persistence. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–8, July 2013. (zitiert auf Seite 55)

[RKD+19] Roman Zoun, Kay Schallert, David Broneske, Sören Falkenberg, Robert Heyer, Sabine Wehnert, Sven Brehmer, Dirk Benndorf, and Gunter Saake. Mstream: Proof of concept of an analytic cloud platform for near-real-time diagnostics using mass spectrometry data. Technical Report 002-2019, Otto-von-Guericke-University Magdeburg, 2019. (zitiert auf Seite 2, 17, 19 und 21)

[ROdVC16] Fábio Roberto Oliveira and Luis del Val Cura. Performance evaluation of nosql multi-model data stores in polyglot persistence applications. pages 230–235, 07 2016. (zitiert auf Seite 11)

[Rom18] Roman Zoun. Internet of metaproteomics. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1714–1718, April 2018. (zitiert auf Seite 5)

[SC05] M. Stonebraker and U. Cetintemel. ”one size fits all”: an idea whose time has come and gone. In *21st International Conference on Data Engineering (ICDE'05)*, pages 2–11, April 2005. (zitiert auf Seite 14)

[SCY04] Rovshan G. Sadygov, Daniel Cociorva, and John R. Yates. Large-scale database searching using tandem mass spectra: Looking up the answer in the back of the book. *Nature Methods*, 1:195–202, 2004. (zitiert auf Seite 6)

[SF12] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Pearson Education, 2012. (zitiert auf Seite 13 und 14)

[SSH11] Gunter Saake, Kai-Uwe Sattler, and Andreas Heuer. *Datenbanken: Implementierungstechniken*. mitp, 2011. (zitiert auf Seite 10)

[SST97] Gunter Saake, Ingo Schmidt, and Can Türker. *Objektdatenbanken - Konzepte, Sprachen, Architekturen*. International Thomson Publishing, 1997. (zitiert auf Seite vii und 9)

[Sto04] Reinhard Stockmann. Was ist eine gute evaluation? einführung zu funktionen und methoden von evaluationsverfahren, 2004. (zitiert auf Seite 43)

[TTC16] Stefka Tyanova, Tikira Temu, and Juergen Cox. The maxquant computational platform for mass spectrometry-based shotgun proteomics. *Nature Protocols*, 11:2301–2319, 10 2016. (zitiert auf Seite 1)

[V14] Manoj V. Comparative study of nosql document, column store databases and evaluation of cassandra. *International Journal of Database Management Systems*, 6:11–26, 08 2014. (zitiert auf Seite 12)

[VMZ+10] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: A data provenance perspective. volume 10, page 42, 01 2010. (zitiert auf Seite 13)

[VWA+14] Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. *Neo4J in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2014. (zitiert auf Seite 37)

[ZDS+18] Roman Zoun, Gabriel Campero Durand, Kay Schallert, Apoorva Patrikar, David Broneske, Wolfram Fenske, Robert Heyer, Dirk Benndorf,

and Gunter Saake. Protein identification as a suitable application for fast data architecture. In Mourad Elloumi, Michael Granitzer, Abdelkader Hameurlain, Christin Seifert, Benno Stein, A Min Tjoa, and Roland Wagner, editors, *Database and Expert Systems Applications*, pages 168–178, Cham, 2018. Springer International Publishing. (zitiert auf Seite 6, 7 und 18)

[ZSB+17] Roman Zoun, Kay Schallert, David Broneske, Robert Heyer, Dirk Benndorf, and Gunter Saake. Interactive chord visualization for metaproteomics. In *2017 28th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 79–83, Aug 2017. (zitiert auf Seite 1)

[ZSB+19] Roman Zoun, Kay Schallert, David Broneske, Wolfram Fenske, Marcus Pinnecke, Robert Heyer, Sven Brehmer, Dirk Benndorf, and Gunter Saake. Msdatastream - connecting a bruker mass spectrometer to the internet. In *Datenbanksysteme für Business, Technologie und Web*, pages 507 – 510. Gesellschaft für Informatik, March 2019. (zitiert auf Seite vii, 17 und 18)

[ZSJ+18] Roman Zoun, Kay Schallert, Atin Janki, Rohith Ravindran, Gabriel Campero Durand, Wolfram Fenske, David Broneske, Robert Heyer, Dirk Benndorf, and Gunter Saake. Streaming fdr calculation for protein identification. In András Benczúr, Bernhard Thalheim, Tomáš Horváth, Silvia Chiusano, Tania Cerquitelli, Csaba Sidló, and Peter Z. Revesz, editors, *New Trends in Databases and Information Systems*, pages 80–87, Cham, 2018. Springer International Publishing. (zitiert auf Seite 18)

[ZXW+16] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. (zitiert auf Seite 18)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 05. September 2019