Polystore Systeme

Alina Böttcher

28. Februar 2019
Grundseminar Wintersemester 2018/2019
Betreuende Prüfer: Prof. Dr. Kai von Luck, Prof. Dr. Jan Sudeikat
Studiengang Master of Science Informatik
Department Informatik
Hochschule für Angewandte Wissenschaften Hamburg
alina.boettcher@haw-hamburg.de

Abstract. Wo bislang entstandenen Daten wieder gelöscht wurden, da werden diese heutzutage immer öfter abgespeichert. Dieses geschieht in der Hoffnung, dass noch nützliches Wissen gewonnen werden kann. Da diese Daten in der Regel von sehr unterschiedlicher Struktur sind, ist es sinnvoll diese auch in spezialisierten Datenbanken abzuspeichern. Häufig wird für die Analyse der Daten eine Verknüpfung zwischen all diesen unterschiedlichen Daten benötigt.
Diese Hausarbeit beschäftigt sich daher mit unterschiedlichen Systemen, die entwickelt wurden, um genau diese Problematik zu lösen. Im Laufe der Arbeit werden zuerst unterschiedliche Arten vorgestellt und im Anschluss eine Beispiel Implementierung eines Polystoren Systems genauer untersucht. Dabei wird zuerst der Aufbau und die Funktionsweise beschrieben. Anschließend wird etwas genauer auf die Ansätze eingegangen, wie so ein System mit Join-Befehlen über die unterschiedlichen Datenbanken umgeht. Dabei wird auch auf mögliche Optimierungen dieser Abläufe eingegangen.

Keywords: Polystore System · Multistore System · Polyglot System · Federated Database System

1 Einleitung

Heutzutage gibt es viele unterschiedliche Anwendungsgebiete für Datenbanken. Entsprechend der vielen unterschiedlichen Einsatzgebiete existieren auch viele unterschiedliche Datenbanken mit den unterschiedlichsten Schwerpunkten und Spezialisierungen.
Aus diesen Spezialisierungen ergibt sich das Problem, dass eine Datenbank nur dann eine gute Performance aufweist, solange man in ihrem Spezialgebiet bleibt. Es gibt aber auch Anwendungen, bei denen das Einsatzgebiet breiter gefächert ist und wo sich dann das Problem ergibt, dass die Performance leidet.
Um diesem Problem zu begegnen gibt es unterschiedliche Ansätze mehrere unterschiedliche Datenbanken zu verknüpfen.
Im ersten Teil dieser Hausarbeit wird sich mit den unterschiedlichen Ansätzen für

die Verknüpfung auseinandergesetzt. Im weiteren Verlauf soll dann der Ansatz der Polystoren Systeme, anhand eines Beispielsystems, genauer angesehen werden.

2 Unterschiedliche Ansätze

In [3] machen die Autoren einen Vorschlag, wie die unterschiedlichen Varianten eingeordnet werden können.
Dabei unterscheiden sie grundsätzlich zwei Hauptmerkmale. Zum einen, die Ähnlichkeit der benutzten Datenbanken und zum anderen, ob eine oder mehrere Anfragesprachen genutzt werden müssen bzw. können.
Daraus ergeben sich dann die folgenden 4 Möglichkeiten.

federated database system: Ähnliche Datenspeicher, die über eine einzelne Abfragesprache angesprochen werden

polyglot system: Ähnliche Datenspeicher, die über verschiedene Abfragesprachen angesprochen werden

multistore database: Unterschiedliche Datenspeicher, die über eine einzelne Abfragesprache angesprochen werden

polystore system: Unterschiedliche Datenspeicher, die über verschiedene Abfragesprache angesprochen werden

Systeme, die ähnlichen Datenspeicher besitzen, die alle über die gleiche Anfragesprache angesprochen werden können, werden in dem Paper federated systems genannt. Ziel ist es, dass die unterschiedlichen Datenspeicher dem Nutzer gegenüber wie ein einziger großer Datenspeicher vorkommen.
Bei den ähnlichen Datenspeichern, die über unterschiedliche Anfragesprachen angesprochen werden, geht es um den Ansatz, dass unterschiedliche Anfragesprachen unterschiedliche Vorteile besitzen. Durch die Möglichkeit unterschiedliche Sprachen nutzen zu können, soll ermöglicht werden, je nach Anfrage Situation zwischen den Sprachen wechseln zu können, um so z.B. das schnellste Ergebnis zu bekommen oder die komfortabelste Anfrage zu formulieren.
Beim dritten Punkt sind sehr unterschiedliche Datenspeicher im Einsatz, welche nach außen alle mit der gleichen Sprache angesprochen werden. Der Nachteil liegt in der begrenzten Auswahl der Datenspeicher, da bei multistore systems festgelegt ist, mit welchen Datenspeichern kommuniziert werden kann. Dieses ist Voraussetzung dafür, dass die Anfrage übersetzt werden kann.
Beim Letzten Punkt geht es dann um die Polystoren Systeme. Sie können aus vielen unterschiedlichen Datenspeichern bestehen, welche in ihrer eigenen Sprache angesprochen werden. Dadurch gehen die Spezialisierungen nicht verloren. Damit weiterhin die gleichzeitige Abfrage von mehreren Datenspeichern möglich ist,

werden die Sprachen innerhalb des Systems übersetzt und vereinheitlicht und können so kombiniert werden. Wie so ein System aussehen kann, soll in den folgenden Kapiteln geklärt werden. Dafür wird ein Testimplementierung durchgegangen, welche in [1] und [2] vorgestellt wird.

3 Aufbau eines Beispiel Polystoren Systems

Als Beispiel wird das Big Data Analytics Working Group (BigDAWG) verwendet, welches in [1] und [2] genauer beschrieben wird.

3.1 Grundgedanken bei der Entwicklung

Bei der Entwicklung lag der Fokus auf der Idee, dass es Anwendungen gibt, die unterschiedliche Datenspeicher benötigen aber nicht auf Performance und Echtzeitanalysen verzichten können. Darüber hinaus muss es möglich sein, komplexe Analysen über unterschiedliche Datenspeicher hinweg stellen zu können, ohne dass der Nutzer sämtlichen Einzelsprachen beherrscht.

3.2 Datenspeicher und Daten

Als Beispiel wird in [1] eine Intensivstation genommen. Dort gibt es Datensätze z.B. über die Patientendaten, die dazugehörenden Protokolle der behandelnden Ärzte oder auch Herzfrequenzkurven. Als Beispiel Datensatz wird der MIMIC II Datensatz genommen. [4] Dabei handelt es sich um einen öffentlich zugänglichen Datensatz mit realitätsnahen Medizinischen Daten.
Die Anforderungen, die an die Beispielanwendung gestellt werden, reichen von Standard SQL Anfragen, über komplexe Mathematische Berechnungen auf Kurvenverläufe und Echtzeit-Analysen bis hin zu Volltextsuchen auf die Protokolle. Insgesamt also ein großes Spektrum, welches Abgedeckt werden soll.
Folgende Datenspeicher werden laut [2] in dem Beispiel BigDAWG verwendet:

PostgreSQL: Als relationale Datenbank

Myria[5]: Für klinische Daten z.B. die Medikation

Apache Accumulo[6]: Die key-value Datenbank wird für Textdokumente, sowie graphische Analysen verwendet

SciDB[7]: Eine Array Datenbank für Wellen von unterschiedlichen Vorgängen wie z.B. die Herzfrequenz

S-Store[8]: Eine Streaming Datenbank für time-series Daten

3.3 Aufbau BigDAWG

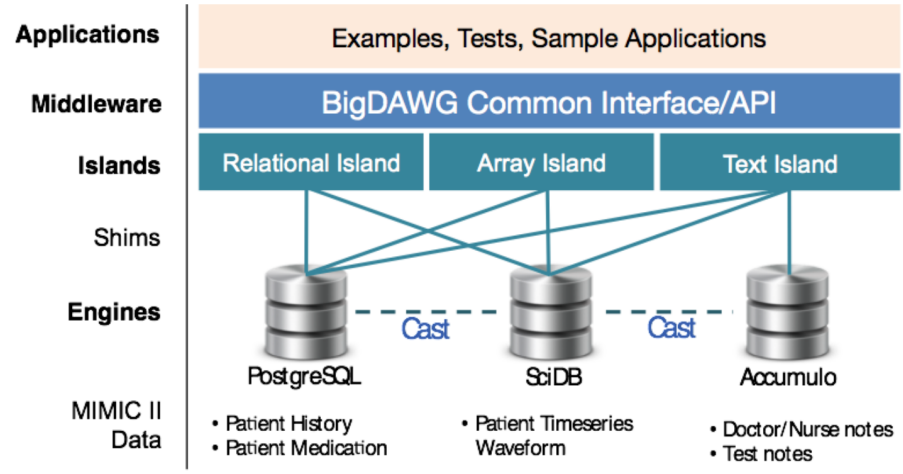

Abbildung 1: Aufbauschema des BigDAWG Polystor Systems [9]

In Abbildung 1 ist der Aufbau des Polystoren Systems gut zu erkennen. Die oberste Schicht setzt sich aus den unterschiedlichen Applikationen zusammen, die auf das System zugreifen. Da in dieser Arbeit das Polystore System im Vordergrund steht, wird auf diese Schicht nicht weiter eingegangen.
In dem folgenden Kapitel werden die danach folgenden Schichten genauer erläutert.

3.4 Middleware

Die Middleware ist für die Anfragen und die Historie zuständig. Der genaue Aufbau ist in Abbildung 2 zu sehen.

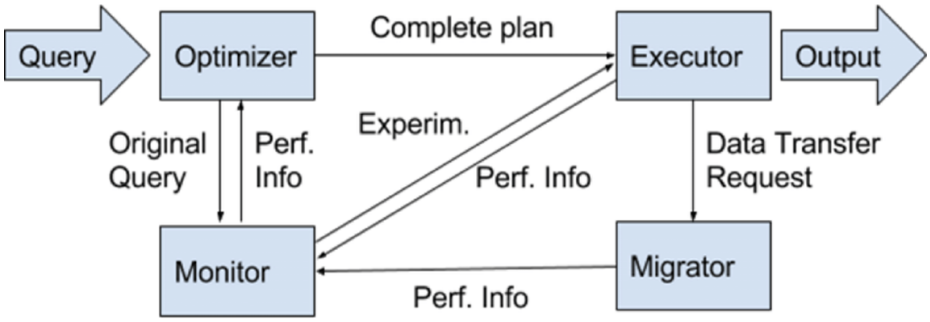

Abbildung 1: Aufbauschema der Middleware [2]

Die Middleware nimmt die Anfragen von einer Applikation entgegen. Der Optimizer parst die Abfrage, um eine Liste von brauchbaren Abfragebäumen zu bekommen. Diese Abfragebäume werden an den Monitor weitergeleitet. Dieser

hat Informationen über vorhergegangene Abfragen und wendet dieses Wissen an, um den leistungsstärksten Abfragebaum zu ermitteln. Eine wichtige Entscheidung dabei ist, aus welcher Datenbank welche Informationen geholt werden müssen.
Danach werden die Informationen an den Exekutor übergeben. Dieser entscheidet über die besten Joins und fragt gegebenenfalls beim Migrator an, wenn Daten zwischen den Datenbanken verschoben werden müssen.
Die Information, wie effektiv eine Anfrage letztendlich war, werden wieder an den Monitor zurückgesendet. So wird ein selbstlernendes System geschaffen, dass bei ähnlichen Abfragen immer effektiver wird.
Damit das System schnell antwortet, wenn eine echte Anfrage kommt, gibt es einen Trainingsmodus. Hier werden dem System realitätsnahe Anfragen mitgegeben und dieses testet alle Möglichkeiten durch. Dafür werden alle vorhandenen Ressourcen aufgewendet. Der Trainingsmodus kann bewusst gestartet werden oder wird automatisch gestartet, nachdem das System mit einer Abfrage konfrontiert wird, die nicht zur Historie passt. In diesem Fall wird ein Moment ohne Abfragen abgewartet und dann diese Abfrage durchgetestet, damit beim nächsten Mal Performance Informationen vorhanden sind.
Dieser Mechanismus hat den Vorteil, dass sich das System an neue Anforderungen anpassen lässt.

3.5 Islands

Die nächste Ebene unter der Middleware sind die Islands. Diese dienen der Kommunikation zwischen der Middleware und den unterschiedlichen Datenbanken.
Es gibt zwei unterschiedliche Typen von Islands. Zum einen die einfachen degenerate Islands. Diese beherrschen nur die Sprache einer einzigen Datenbank. Entsprechend gibt es für jede benutzte Datenbank eine entsprechende Island.
Zum anderen gibt es die komplexeren Islands, welche nicht nur mit einer, sondern mit mehreren Datenbanken sprechen können. In diesem Beispiel wurden zwei solcher komplexen Islands verwendet.

- D4M [10]
- Myria [5]

D4M unterstützt PostgreSQL, Apache Accumulo und SciDB.
Myria unterstützt Myria, PostgreSQL und SciDB.

Diese komplexen Islands beherrschen die Sprachen von mehreren Datenbanken und sind daher bestens dazu geeignet, wenn Informationen aus mehreren Datenbanken gleichzeitig verarbeitet werden sollen. Wenn ein Nutzer eine Anfrage an das System stellt, dann werden diese im Rahmen einer bestimmten Insel gestellt. Wenn dieses nicht möglich ist, so müssen die Befehle von mehreren Inseln mit dem SCOPE Befehl verbunden werden. Der Nutzer muss also eine Vorstellung davon haben, was er von welcher Datenbank wissen möchte, um zu entscheiden, wie er die Anfrage formuliert.

3.6 Shims

Nachdem die Islands mitgeteilt bekommen haben, an welche Datenbank sie welche Anfrage stellen sollen, werden diese Anfrage mittels der Shims übermittelt. Sie dienen als Wrapper zwischen den Islands und den eigentlichen Datenbanken.

3.7 Cast

Manchmal müssen Informationen zwischen den unterschiedlichen Datenbanken ausgetauscht werden. Damit das möglich ist, müssen die Daten semantisch verändert werden. Dafür ist der CAST Befehl zuständig. Dieser Befehl wird vom Migrator gesteuert.

4 Join über verschiedene Datenbanken

Nachdem jetzt der grobe Aufbau eines Beispiel Systems beschrieben wurde, soll noch ein vertiefender Blick auf die Abfragen an sich geworfen werden. Damit so eine Verknüpfung von unterschiedlichen Datenbanken Sinn macht, müssen Join-Befehle über die verschiedenen Datenbanken hinweg möglich sein. Im Paper [11] wird genauer analysiert, welche Schritte notwendig sind, um so einen Join mit einer akzeptablen Performance zu ermöglichen. Wie schon in Kapitel Middelware beschrieben, findet das Planen der Ausführungspläne in der Middleware statt. Dort ist der Exekutor für das Ausführen der erstellten Abfragepläne verantwortlich. Entsprechend finden auch die Wichtigen Entscheidungen, welche Datenbanken welche Teilaufgaben übernehmen, hier statt.

Ein Schwerpunkt bei der Performanten Ausführung der Joins ist die Fragestellung, wie möglichst viel parallel auf den unterschiedlichen Datenbanken gearbeitet werden kann. Wenn ein Exekutor einen Abfrageplan erhält, wird zuerst geschaut, ob Join-Befehle enthalten sind, die nur eine einzelne Datenbank betreffen. Falls dieses der Fall sein sollte, dann werden diese Abfragen-Bestandteile an die entsprechende Datenbank abgegeben und müssen für die Datenbank übergreifenden Joins nicht weiter beachtet werden.

4.1 Ablauf

Zuerst teilt die Optimierungssoftware die beteiligten Datensätze kleinen Join-Einheiten zu. Datensätze könne hier z.B. Zeilen in PostgreSQL oder Key-Value Paare in Accumulo sein. Diese Einteilung erfolgt anhand ihrer Attribute, die für den Join wichtig sind. Damit wird erreicht, dass ähnliche Werte für die Berechnung, zusammen auf einer Ausführungsdatenbank landen. Dieses ist Voraussetzung dafür, dass die Abfragezeit nicht zu lang wird.
Es sollte daran gedacht werden, dass die einzelnen Bestandteile einer Join-Einheit, vor der Zuteilung zu einer Einheit, auf ganz unterschiedlichen Datenbanken liegen können und zusammengebracht werden müssen.

Beim Ablauf werden grundsätzlich fünf Punkte abgearbeitet, wobei der Erste unter Umständen entfallen kann. Die Gesamtausführungszeit des Joins setzt sich hauptsächlich aus den unten aufgeführten Punkten zusammen.

1. Prüfung auf eventuell vorhandenen Versatz in den Datensätzen

2. Zuweisung zu den Join-Einheiten

3. Die Join-Einheiten werden physisch auf der zugewiesenen Engine zusammengeführt

4. Die Join-Abfragen werden innerhalb der Join-Einheiten durchgeführt

5. Die Teilergebnisse der einzelnen Join-Einheiten werden auf der Ziel-Datenbank zusammengetragen und vereint

Die Zuweisung und anschließende Zusammenführen der Join-Einheiten werden vom Migrator übernommen. Dieser wiederum gibt die Informationen nach Schritt drei an die entsprechende Datenbank weiter, wo dann der Join durchgeführt wird. Der Exekutor kümmert sich hauptsächlich um die Prüfung des Versatzes und die Zuweisung der Datenbanken.

4.2 Versatz-Prüfung

Im Ersten Schritt sammelt der Exekutor Informationen darüber, welche Datensätze überhaupt in Join-Einheiten zusammengefasst werden können. Die dafür nötigen Informationen müssen entweder anhand der Daten erstellt werden oder werden von den entsprechenden Datenbanken generiert.
Da die Daten in den Join-Einheiten möglichst ähnliche Werte aufweisen sollen, werden die Daten anhand der Werte in Gruppen eingeteilt. Eine Join-Einheit bekommt eine bestimmte Anzahl von diesen Gruppen zugewiesen, die sich dann im weiteren Verlauf auch nicht wieder verändert.

Falls die Informationen bereits von der Datenbank generiert wurden, dann bieten mache Datenbanken noch zusätzliche Informationen über die häufigsten Werte an. Falls dieses der Fall ist, dann werden diese Informationen ebenfalls an den Exekutor weitergeben.

Bei der Erstellung der Informationen gibt es zwei unterschiedliche Strategien.

1. Die komplette Tabelle wird gescannt

2. Aus der Tabelle wird nur eine feste Anzahl an Datensätzen angesehen und dann Rückschlüsse auf die Tabelle gezogen

Die Vorteile beim Ersten Punkt sind, dass die Zuweisung zu den Join-Einheiten genauer ist und somit die Geschwindigkeit bei den Joins höher ausfallen kann. Der Nachteil ist, dass das Scannen der kompletten Tabelle deutlich länger dauert. Beim zweiten Punkt ist das Scannen der Tabelle deutlich schneller, jedoch kann es aufgrund der Ungenauigkeiten später zu Verzögerungen während der Joins kommen. Welche Variante geschickter ist, hängt vom Einzelfall ab.

4.3 Join-Einheiten

Nachdem die Informationen gesammelt wurden, muss der Exekutor entscheiden, wie die Werte am besten gruppiert werden und auf welcher Datenbank sich die einzelnen Join-Einheiten treffen sollen. Die Entscheidung auf welcher Datenbank der Join ausgeführt werden soll, kann großen Einfluss darauf haben, wie schnell die Abfrage bearbeitet wird, da die Informationen von ganz unterschiedlich strukturierten Datenbanken kommen können.
Neben der Migration der Daten spielt auch das Netzwerk eine große Rolle. Häufig ist dieses der Flaschenhals in so einem System. Entsprechend sollte in die Entscheidung mit einfließen wie groß die Menge der zu verschiebenden Daten wird, wenn über die Ziel Datenbank nachgedacht wird.
Darüber hinaus kann eine Datenbank nicht gleichzeitig Daten migrieren und Joins darauf durchführen. Es muss also immer gewartet werden bis die Migration der Daten komplett abgeschlossen ist und erst dann kann mit dem Durchführen der Joins begonnen werden. Daraus folgt, dass die Übertragungsmenge der Daten nicht nur Einfluss auf die Übertragungsgeschwindigkeit, sondern auch auf die anschließende Join-Geschwindigkeit hat.
Da der abschließende Join die Zwischenjoins der einzelnen Join-Einheiten benötigt, legt die langsamste Datenbank in dem parallel arbeitenden System die Gesamtgeschwindigkeit fest.

4.4 Strategien zur Zeitoptimierung

Es zwei grobe Strategiebereiche, wie an die Problematik der zeitintensiven Migration und knappen Netzwerkkapazitäten herangegangen wird.

1. Die komplette Tabelle als eine Join-Einheit

2. Die Join-Attribute

Wenn sich die Join-Einheit aus nur einer Tabelle zusammensetzt, dann führt das zum einen dazu, dass die Versatzprüfung vernachlässigt werden kann. Es ist nicht notwendig, dass die Informationen vorhanden sind, da die Tabelle eh nicht mit anderen Datensätzen zu Join-Einheiten kombiniert wird.
Zum anderen ist der Migrationsaufwand begrenzt. Der große Nachteil liegt darin,

dass eventuell ungenutzte Datensätze mitgeschleppt werden, die eigentlich nicht gebraucht werden. In dieser Variante gibt es zwei Möglichkeiten vorzugehen.

1. Sämtliche Join-Einheiten, also sämtliche beteiligten Tabellen werden auf die Zieldatenbank übertragen und dort ein großer Join durchgeführt. Die Auslastung für das Netzwerk ist hoch und der Join auf der Zieldatenbank muss warten, bis alle Daten übertragen wurden. Dafür entfallen die Netzauslastungen und die Wartezeiten, die während des Zwischenjoins entstehen würden. Bei der Wahl der Zieldatenbank sollte aber darauf geachtet werden, dass es sich um eine Zieldatenbank handelt, die die Joins zügig bearbeiten kann, sodass keine weiteren Verzögerungen entstehen.

2. Es wird geschaut, welches die größte Tabelle ist. Alle kleineren Tabellen werden auf die Datenbank der größten Tabelle migriert. Dadurch wird versucht die Netzauslastung etwas zu verringern. Der große Nachteil kann sein, dass die Datenbank der größten Tabelle nicht die Datenbank ist, die vom Exekutor als Zieldatenbank angegeben wurde. In diesem Fall muss zum Schluss noch alles auf die Zieldatenbank migriert werden.

Der Zweite große Ansatz, wie versucht wird die Geschwindigkeit zu verbessern, setzt bei den Join-Attributen an. Grundsätzlich ist ja der Ansatz, die Datensätze nach gleichen Attributen in den Join-Einheiten zusammenzuführen. Um die Join-Einheiten zusammenzuführen, müssen evtl. sehr viele Migrationen durchgeführt werden. Jede Datenbank kann theoretisch zu jeder anderen Datenbank Datensätze migrieren und von allen anderen Datenbanken Datensätze erhalten.
Um diesen Umstand abzumildern wird versucht die Minimale Bandbreitenheuristik [12], welche aus SciDB stammt zu übernehmen.

Minimale Bandbreitenheuristik(MBH): Die MBH geht davon aus, dass die Zeit der Migration die Gesamtlaufzeit dominiert. Daher wird hier versucht die Datensätze dorthin zu migrieren, wo bereits die meisten dazugehörenden Datensätze der Join-Einheit liegen. Im Grunde ein ähnlicher Grundgedanke wie bei den Tabellen, jedoch auf die Join-Attribute bezogen.
Der Exekutor prüft, wo sich die meisten Datensätze befinden und legt diese Datenbank dann als Ziel für die Join-Einheit fest. Dadurch kann die Migration etwas minimiert werden und die Joins können schneller durchgeführt werden.

Ebenfalls in [12] wird eine weiterführende Verbesserung des MBH beschrieben. Hier wird zusätzlich noch untersucht, ob die Join-Einheit auf einer Datenbank gelandet ist, die vergleichsweise langsam ist. Sollte dieses der Fall sein, so wird versucht die darauf befindenden Join-Einheiten auf anderen Datenbanken zu verschieben, welche schneller sind. Das erhöht zwar wieder den Migrationsaufwand, kann sich aber rechnen, wenn die anschließenden Joins deutlich schneller durchgeführt werden können.

Sollte es zu einer Verschiebung auf eine andere Datenbank kommen, dann wird dieses in einer Liste vermerkt und somit nicht weiter beachtet. Diese "Tabu-Liste" gab dieser Erweiterung auch ihren Namen "Tabu-Search".

5 Schluss

5.1 Fazit

Systeme, die über unterschiedliche Datenbanken hinweg arbeiten, bieten großes Potenzial. Sie ermöglichen es, die Datenbanken zu verwenden, die für die aktuelle Problematik am besten geeignet sind und trotzdem muss nicht jedes Mal wieder eine neue Datenbank-Sprache gelernt werden. Für den Anwender bietet es daher eine große Erleichterung. Aber es konnte auch gezeigt werden, dass die Entwicklung noch nicht abgeschlossen ist. So gibt es im Bereich der Anfrageformulierung noch Potenzial. Ohne den SCOPE Befehl, der das Wissen über die Unterschiedlichen Datenbanken voraussetzt, wäre es für den Anwender viel einfacher das System zu bedienen. Darüber hinaus muss auch an der Anzahl der unterstützten Datenbanken weitergearbeitet werden, da genau hier die Stärke dieser Systeme liegt.

5.2 Weiteres Vorgehen

Nachdem in dieser Arbeit die Grundlagen eines Polystoren Systems untersucht wurden, soll das Thema nun weiter vertieft werden.
Die Vertiefung wird sich auf den Bereich der Datenmigration konzentrieren, welche in dieser Hausarbeit zwar häufig angesprochen aber nicht tiefergehend behandelt wurde. Es ist aber deutlich geworden, dass es sich bei der Datenmigration um ein wichtiges Feld im Bereich der Polystoren Systeme handelt, da ohne diese der komplette Datenaustausch zwischen den Datenbanken nicht möglich wäre.
Da es bei der Migration um die Datenumwandlung zwischen vielen unterschiedlichen und teilweise auch sehr neuen Datenbanken geht, wird es ein großes Themenfeld werden, bei dem sich wohl noch weitere konkrete Vertiefungen ergeben.

References

1. A. Elmore, J. Duggan, M. Stonebraker, M. Balazinska, U. Cetintemel, V. Gadepally, J. Heer, B. Howe, J. Kepner, T. Kraska, S. Madden, D. Maier, T. Mattson, S. Papadopoulos, J. Parkhurst, N. Tatbul, M. Vartak, S. Zdonik, A Demonstration of the BigDAWG Polystore System, Proceedings of the VLDB Endowment - Proceedings of the 41st International Conference on Very Large Data Bases, 2015, pp. 1908-1911.
2. Vijay Gadepally, Peinan Chen, Jennie Duggan, Aaron Elmore, Brandon Haynes, Jeremy Kepner, Samuel Madden, Tim Mattson, Michael Stonebraker, The BigDAWG Polystore System and Architecture, IEEE High Performance Extreme Computing Conference (HPEC), 2016, pp. 1-6.

3. Ran Tan, Rada Chirkova, Vijay Gadepally, Timothy G. Mattson, Enabling Query Processing across Heterogenous Data Models, IEEE International Conference on Big Data (BIGDATA), 2017, pp. 32113220.
4. Mohammed Saeed, Mauricio Villarroel, Andrew T. Reisner, Gari Clifford, Li-Wei Lehman, George Moody, Thomas Heldt, Tin H. Kyaw, Benjamin Moody, Roger G. Mark, Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): A public-access intensive care unit database, Critical Care Medicine, 2011, pp. 39:952-960.
5. Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu,Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang,Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, Dan Suciu, Demonstration of the Myria Big Data Management Service, Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data(ACM SIGMOD), 2014, pp. 881-884.
6. Dylan Hutchison, Jeremy Kepner, Vijay Gadepally, Adam Fuchs, Graphulo implementation of server-side sparse matrix multiply in the Accumulo database, IEEE High Performance Extreme Computing Conference (HPEC), 2015, pp. 1-7.
7. Michael Stonebraker, Paul Brown, Donghui Zhang, Jacek Becla, SciDB: A Database Management System for Applications with Complex Analytics, IEEE Computing in Science & Engineering, 2013, pp. 54 - 62.
8. Ugur Cetintemel, Jiang Du, Tim Kraska, Samuel Madden, David Maier,John Meehan, Andrew Pavlo, Michael Stonebraker, Erik Sutherland, Nesime Tatbul, Kristin Tufte, Hao Wang, Stanley Zdonik, S-Store: A Streaming NewSQL Systemfor Big Velocity Applications, Proceedings of the VLDB Endowment,Vol.7, No.13, 2014, pp. 1633 - 1636.
9. BigDAWG Polystore, https://bigdawg.mit.edu/. Last accessed 13 Februar 2019
10. Vijay Gadepally, Jeremy Kepner, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Lauren Edwards, Matthew Hubbell, Peter Michaleas, Julie Mullen, Andrew Prout, Antonio Rosa, Charles Yee, Albert Reuther, D4M: Bringing associative arrays to database engines, IEEE High Performance Extreme Computing Conference (HPEC), 2015, pp. 1 - 6.
11. Ankush M. Gupta, Vijay Gadepally, Michael Stonebraker, Cross-Engine Query Execution in Federated Database Systems, IEEE High Performance Extreme Computing Conference (HPEC), 2016, pp. 1 - 6.
12. Jennie Duggan, Olga Papaemmanouil, Leilani Battle, Michael Stonebraker, Skew-Aware Join Optimization for Array Databases, ACM SIGMOD '15: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 123 - 135.