FernUniversität Hagen

– Faculty of Mathematics and Computer Science –

Dynamically allocation of Datasets to Datastores in a Polystore Environment

Presented by

Daniel Langhann

Matriculation number: 3788687

Supervision : Prof. Dr. Ute Störl

Daniel Langhann: *Dynamically allocation of Datasets to Datastores in a Polystore Environment*, © 01. Feb 2025

DECLARATION

I declare that I have written the seminar paper independently and without any unauthorized assistance from third parties.

In doing so, I have only used the sources and resources specified and have identified the passages taken from these sources, either verbatim or in meaning, as such.

The assurance of independent work also applies to any drawings, sketches, or graphic representations contained therein.

The paper has not been submitted in the same or similar form to the same or any other examination authority nor has it been published.

By submitting the electronic version of the final paper, I acknowledge that it may be checked for plagiarism using a plagiarism detection service and will be stored exclusively for examination purposes.

Hagen, 01. Feb 2025

Daniel Langhann

ABSTRACT

In der vorliegenden Masterarbeit wird das Thema Dynamische Allokation von Datasets zu Datastores behandlet. Die Motivation für die Arbeit ist es, einen Vorschlag zu unterbreiten, wie in einem Umfeld aus heterogenen Datasets, für die mehr als ein Datenbank-System ideal ist, eine dynamische Zuweisung und Neuzuweisung von Datasets zu Datastores zu realisieren. In der Vergangenheit wurde bereits viel an dem Themenbereich Polystores geforscht. Polystores beinhalten mindestens zwei verschiedenartige Datastores also letztendlich Datenbank-Systeme um hetoregene Datasets, zum Beispiel transaktionsbasierte Daten auf der einen Seite und aggregierte Daten für die Analyse von zum Beispiel sogenannten Key Perfomance Indikatoren (KPI) auf der anderen Seite optimal verarbeiten zu können. Eine bislang ungelöste Herausforderung in einem polystoren Umfeld ist, die Reaktion auf sich verändernde Workloads in der Applikaton bzw. im Gesamtsystem. Wie reagiert man bezogen auf das oben genannte Beispiel, wenn in dem transaktionsbasierten Bereich des Gesamtsystems vermehrt analytische Abfragen entstehen, also GET Requests mit langen und sehr langen Laufzeiten. Dann wäre es wünschenswert, dass diese Abfragen zukünftig über den Datastore, der sich auf analytische Abfragen fokussiert, umgeleitet werden.

Genau hier setzt die vorligende Arbeit an. Es soll ein Vorschlag unterbreitet werden, wie man in einem polystoren Systemumfeld, dynamisch auf sich verändernde Workloads reagieren kann. Der erste Schwerpunkt beschäftig sich damit, bezogen auf gegebene Anforderungen an ein Gesamtsystem eine initiale Zuweisung von Datasets zu Datastores vorzuschlagen. Es wird ein Prototyp entwickelt, der algorithmisch eine entsprechende Zuweisung ermittelt. Der zweite Schwerpunkt fokussiert sich auf die permanante und dynamische Analyse von Workloads um bei sich verändernden Paramtern bezogen auf den initialen Zustand mit einer Re-Allokation der Datasets zu reagieren bzw. einen angepassten Vorschlag zu unterbreiten. Für diesen zweiten Schwerpunkt wird ebenfalls ein Prototyp entwickelt, der als Input die Analysedaten des Workloads erhält, und beim Erreichen bestimmter Schwellwerte einen Vorschlag für eine Neuzuweisung unterbreitet. Für die konkrete Ausführung der angepassten Zuweisungen wird ein theoretischer Vorschlag gemacht und diskutiert.

Die Arbeit schließt wiederum mit einer Zusammenfassung der Ergebnisse bezogen auf die Ausführung und Testung der Prototypen. Darüber hinaus sollen Vorschläge unterbereitet werden, an welchen Themen noch gearbeitet werden sollte, um tatsächlich ein komplett autonom arbeitendes Gesamtsystem auf der Basis von Polystores zu realisieren.

CONTENTS

ABKÜRZUNGSVERZEICHNIS

1

EINLEITUNG

The topic of polystores has been an active area of research for more than ten years. This is relatively easy to explain, as the digital world is becoming increasingly multifaceted and complex. The amount of data to be processed continues to grow. Whereas the goal used to be to design and implement database concepts that used storage space as sparingly as possible, the focus today tends to be on performance. Recommendation systems must be able to provide meaningful suggestions in real-time or very quickly based on inputs, for example in the context of a customer **C** bought item **A**, then they also bought item **B** or item **C** or . . . :

$$\forall c \in \mathcal{C} \quad \Big(\text{Bought}(c, A) \implies \big(\text{Bought}(c, B) \vee \text{Bought}(c, C) \vee \cdots\big)\Big)$$

The topic of polystores has been an active area of research for more than ten years. Such a request can only be meaningfully and efficiently answered if a suitable algorithm quickly receives input from an appropriate database. Such a database could be, for example, a graph database. In a web shop, for instance, where such a recommendation system is implemented, typical database transactions, known as Create, Read, Update, Delete (CRUD) operations, frequently occur. These transactions can often be better handled by a classical SQL database. Such a system can be referred to as a polystore because it requires at least two different database systems or data stores to function efficiently. Sometimes, the transitions are more fluid than in the aforementioned example of an e-commerce system with one database for the recommendation engine and another for the transactional part of the system. For example, so-called data streams can be handled just as well in a document-based database as classical CRUD operations. It is conceivable that both parts of a hypothetical system could be mapped using, for example, a MongoDB.
In this case, it would be sensible to measure whether a system consisting of two or more different services that share a data store or database system would benefit from reallocating services or datasets to a data store when faced with a dynamically changing workload. This could be beneficial, for example, if certain transactions or commits start taking more time, negatively impacting user experience or system performance.

Fundamentally, the issue is that in a system, there must be at least two different services **S**, each assigned to a data store **D** and having a certain workload **W**. If the workload **W** changes or certain thresholds **Th** are exceeded, the services or the affected service should be assigned to another, possibly more suitable, data store **D**. This should only happen if the service **S**, in conjunc-

tion with the newly assigned data store $\mathbf{D_n}$, exhibits an improved workload $\mathbf{W_i}$.In summary, the situation can be described as follows:

SERVICE REASSIGNMENT BASED ON WORKLOAD

We define the following:

- **Services:**

  $$\mathcal{S} = \{S_1, S_2, \ldots, S_n\} \quad (n \geq 2)$$

- **Datastores:**

  $$\mathcal{D} = \{D_1, D_2, \ldots, D_m\}.$$

- **Workload Function:**

  $$W : \mathcal{S} \times \mathcal{D} \to \mathbb{R}_{\geq 0},$$

  where $W(S, D)$ measures the workload of service S when using datastore D (lower is better).

- **Thresholds:** Let $Th \geq 0$ be the workload threshold, and let $\epsilon > 0$ be a parameter defining a significant change in workload over time.

Let $A : \mathcal{S} \to \mathcal{D}$ be the current assignment function, meaning that a service S is assigned to a datastore $A(S)$.

For a given service S, if either:

$$\left|W(S, A(S))_t - W(S, A(S))_{t-1}\right| \geq \epsilon$$

or

$$W(S, A(S)) \geq Th,$$

and there exists another datastore $D' \in \mathcal{D}$ such that:

$$W(S, D') < W(S, A(S)),$$

then we reassign service S to D'.

This rule can be written as:

$$A'(S) = \begin{cases} D', & \text{if } \left(\left|W(S, A(S))_t - W(S, A(S))_{t-1}\right| \geq \epsilon \text{ or } W(S, A(S)) \geq Th\right) \\ & \quad \text{and there exists } D' \in \mathcal{D} \text{ with } W(S, D') < W(S, A(S)); \\ A(S), & \text{otherwise.} \end{cases}$$

It is therefore necessary to constantly check how a system of datasets, services, and data stores should be structured in order to optimally meet the

requirements for this system. This is precisely the topic that the present work addresses.

1.1 MOTIVATION

Es existieren bislang nur wenig bis gar keine polystoren Systeme die es über den Status Prototyp geschafft haben. Das Projet was wohl am fortgeschrittensten ist ist Polyphenie DB. Ein Erklärungsansatz für den mangelhafte Verbreitung von Polystore Systemen und oder deren produktiven Einsatz ist, dass die Implmentierung immer noch schwierig und aufwendig ist.

Ein anderer Erklärungsansatz ist, dass die Vorteile eines Polystores erst zum tragen kommen, wenn dieses in einem dynamischen Umfeld bzw. System zum Einsatz kommt. In einem dynamischen Umfeld ändert sich der Workload und damit ist wie bereits beschrieben die initiale oder aktuelle Zuordnung von Datastores zu Datasets nicht mehr valide bzw. optimal.

Nun ist es schwer vorstellbar, dass in einem produktiven Betrieb permanent manuell Datasets anderen Datenbanken und Datenbank Systemen zugeordnet werden. Ein solcher Eingriff ist komplex, erfordert eine sorgfältige Vorbereitung und beansprucht in den meisten Fällen einen nicht unerheblichen Teil der Ressourcen (insbesondere Personal wie Datenbank Administratoren und Backend-Entwickler etc.). Des weiteren geht mit einer solchen Umstellung ein nicht zu unterschätendes Risiko einher. Dieses Risiko kann durch bestimmte Verfahren wie das Testen der Umstellung in einer gekapselten Testumgebung, einer schrittweisen Umstellung etc. minimiert werden, dennoch sind Anpassungen am Datenmodell bzw. an der Persistenz-Schicht eines Systems immer heikel und unterliegen häufig einem sehr statischen Setup.

Des Weiteren stellt sich immer die Frage wenn festgestellt wird, dass eine Zuordnung Datasets zu Datastores nicht mehr optimal ist, was die bessere Lösung wäre und wie sich der Workload durch eine Unmstellung potentiell verändern würde. Eine manuelle Überprüfung der Workloads ist zwar denkbar aber schwer mit dem Alltag von Entwicklungs-Teams vereinbar. Wenn man feststellt, dass die Performance aber auch andere parameter wie Verfügbarkeit, also letztendlich Service Level Agreements (SLA) nicht mehr den Anforderungen entsprechen, stellt sich die Frage, wie das System umgestellt werden sollte oder umgestellt werden kann um die Anforderungen wieder zu erfüllen.

Die weiterführende Vision bzw. Motivation dieser Arbeit ist, eine Polystore System welches auf Basis von Workloads, Applikationsparametern und SLA automatisch eine initial Zuordnung von Datasets zu Datastores vornimmt und dieses Setup kontinuierlich und automatisiert überwacht und im Fall von Abweichungen sich selbst optimiert. Ein solches System wäre gerade für komplexe Anwendungen mit heterogenen Anforderungen an die entsprechenden Datastores ein Mehrwert und trägt der Erkenntnis Rechnung das im Bereich Datastores und Datenbank eine One Fits All Lösung nicht bzw. bislang niht existiert. So entsteht für den Benutzer ein opti-

miertes Nutzererlebnis und das Entwicklungs-Team kann sich voll auf die Implmentierung von User Interfaces und Business Logik fokussieren.

1.2 ZIELE DER ARBEIT

Die Ziele der Arbeit sind wie folgt:

1. Einen theoretischen Überblick zu lieferen, wie ein dynamisches polystore basiertes System beschaffen sein muss bzw. beschaffen sein sollte, damit dieses in produktiven Umgebungen eingesetzt werden kann.

2. Konkret zu beschreiben welche Schritte ein dynamisches polystore basiertes System immer wieder durchlaufen muss um sich selbst aktuell zu halten und ein möglichst optimalen fit zwischen Datasets und Datastores zu generieren.
3. Einen Prototyp zu implementieren, der auf Basis von definierten Inputs eine intiale Zuordnung von Datasets zu Datastores herstellt.
4. Auf Basis des Prototyp aus 3. einen Algorithmus zu implementieren, der den Workload von Datastores analysiert und dynamisch neue Zuordnungsvorschläge generiert, wenn sich der Workload verändert.
5. Theoretisch zu beschreiben welche Schritte unternommen werden müssen, um eine dynamisch errechnete neue Zuordnung von Datasets zu Datastores zu exekutieren.

1.3 GLIEDERUNG

Die Arbeit ist wie folgt gegliedert:

Nach der Einleitung werden in Chapter2 TODO Terminologische Grundlagen die für das Verständnis der Arbeit wichtig sind erörtert.

In Chapter3 TODO beginnt der Hauptteil der Arbeit, der sich mit den Zielen 1. und 2. aus Chapter TODO beschäftigt. In diesem Kapitel soll vor allem ein Überblick vermittelt werden, wie dynamisch polystore basiertes System funktionieren kann und welche Kernelemente und Schnittstellen dieses enthalten muss.

Chapter4 beschäftigt sich mit der Implmentierung der Prototypen in Verbindung mit den Zielen 4 und 5 TODO. Es soll Konkret beschrieben werden wie der Prototyp aufgebaut ist bzw. der Algorithmus zur dynamischen Neuzuteilung funktioniert. Zusätzlich werden die Ergebnisse aus dem Testbetrieb beschrieben und gegebenfalls sinnvolle Ergänzungen und oder Einschränkungen bezogen auf den praktischen Einsatz dokumentiert.

Chapter5 beschreibt die konkreten Schritte zur Umsetzung einer dynamischen Neuzuteilung von Datasets zu Datastores. Wie können Schnittstellen aussehen oder müssen diese beschaffen sein um Datasets einen neuem Datastore zuzuweisen? Welche Rolle spielen Konzepte wie Objekt Relationale Mapper (ORM)? Welche konkreten Arbeitsschritte müssen durchlaufen wie durchlaufen werden um die neue Zuordnung so umzusetzen, dass die zu

Grunde liegende Anwendung oder das zu Grunde liegende System weiter arbeitsfähig ist? Welche Service Level Agreements müssen wie eingehalten werden und was bedeuten zum Beispiel Anforderungen wie eine Zero Downtime für ein solches System?

In Chapter 6 werden die Ergebnisse aus dem Hauptteil (TODO Chapter3 bis Chapter5) zusammengefasst. Zusätzlich soll ein Ausblick gegeben und Empfehlungen gegeben werden, wie das Thema dynamische Polystores weiter entwickelt werden können. Die Arbeit endet mit einem Fazit bezogen auf die zusammengefassten Ergebnisse und den Ausblick.

2

TERMINOLOGISCHE GRUNDLAGEN

Nach der Einleitung sollen nun in diesem Kapitel wichtige Terminologische Grundlagen geklärt werden, um ein einheitliches Verständnis zu generieren.

Dataset

A dataset is a structured collection of related data that is organized for efficient storage, retrieval, and analysis. In the context of datasets and datastores, it typically refers to:

Organization: Data arranged in a systematic format (e.g., tables, files, or arrays) that can be easily queried or processed.

Purpose: Aimed at supporting tasks like analysis, reporting, or machine learning model training.

Metadata: Often accompanied by metadata that describes the structure, source, and meaning of the data, enhancing its usability.

Storage: May be stored in various datastores such as databases, file systems, or cloud storage services, ensuring accessibility and scalability.

Datastore

A datastore is a system or repsitory designed for the storage, management, and retrieval of data, including datasets and other data objects. In the context of datasets and datastores:

Storage Mechanism: A datastore provides the underlying nfrastructure to physically or logically store data. This can be implemented as a database, file system, data warehouse, or cloud storage service

Management and Retrieval: It offers methods for efficiently accessing, querying, and updating the stored data. This might involve indexing, transaction management, or specialized query languages.
Data Integrity and Security: Datastores often include mechanisms for ensuring data accuracy, consistency, and protection against unauthorized access.
Scalability: They are built t handle varying amounts of data, supporting growth and high performance as data volume increases.

Datasets and Datastores

In summary, while a dataset is the structured collection of data used for analysis or processing, a datastore is the system that houses and manages these datasets, ensuring that data is organized, accessible, and secure.

Nach dem die grundsätzlichen Begriffe Datasets und Datastores geklärt sind, werden die beiden Begriffe Mulltistore und Polyglot Persistenz definiert

und voneinander abgegrenzt. Beide Konzepte honorieren, dass eine One Size fits all Lösung für verschiedenartige Datassets nicht existiert und pro Dataset er jeweils best passende Datastore genutzt werden sollte.

Multistore

Ein Multistore besteht aus einer einzigen öffentlichen Schnittstelle und übersetzt die Anfragen an diese Schnittstelle zu Query Interface der verschiedenen und verschiedenartigen Datastores Dij wobei i den jeweiligen Datastore identifiziert und j den Typ des Datastores abbildet:

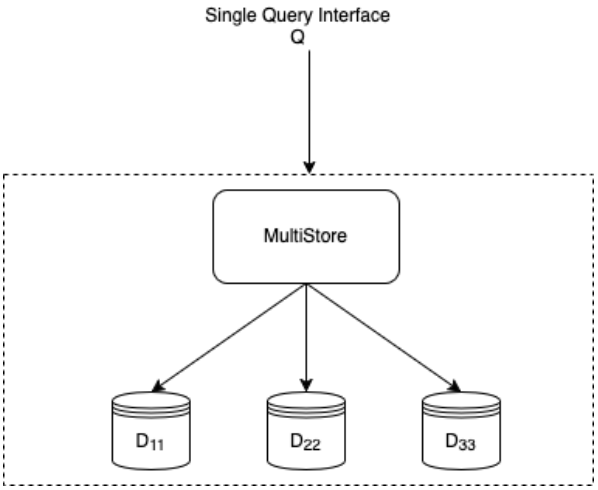

Figure 2.1: Concept of a Multistore

Polyglot Persistenz Ein Polystore wiederum bietet verschiedene öffentliche Schnittstellen Q1,...,QN und leitet die Anfragen an die entsprechenden Datastores 1,...,N weiter.

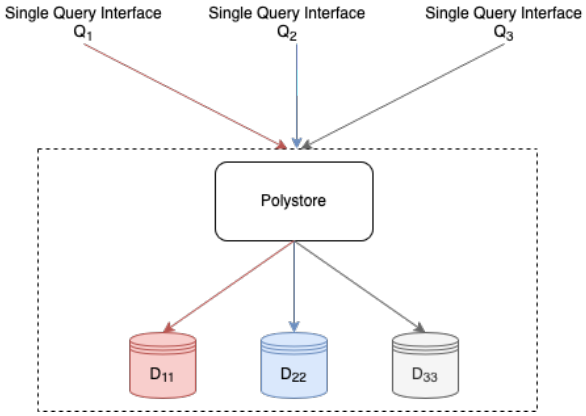

Figure 2.2: Concept of a Multistore

Polystore Ein Polystore ist eine Kombination aus Multistore und Polyglot Persistenz.

3

INITIALE ZUORDNUNG VON DATASETS ZU DATASTORES

Nach der Klärung wesentlicher Begriffe und Terminologien beginnt nun im Chapter3 der Hauptteil der Arbeit. Dieser beginnt mit der initialen Zuordnung von datasets zu Datastores. Es wird im weiteren von einem Zustand auf der so genannten grünen Wiese ausgegangen. Das heißt eine beliebige Applikation bzw. System wurde entwickelt und soll nun deployed bzw. bereitgestellt werden. Hierfür ist es notwendig, dass die initiale Zuordnung von datasets zu datastores feststeht. Grundsätzlich ist also festzuhalten, dass der initale Zuordnungsprozess optimalerweise in eine Deployment Pipeline eingebunden wird. Ein Vorteil dieser Vorgehensweise ist, dass die gemachte Zuordnung über ein Staging-System getestet werden kann.

3.1 INITIALE ZUORDNUNG VON DATASSETS ZU DATASTORES

In diesem Kapitel wird der Arbeitsschritt des Programms die initiale Zuordnung von Datasets zu Datastores beschrieben. In Kapitel TODO wird der grundsätzliche Aufbau des Systems, referenzierend auf die Beschreibung aus Kapitel TODO beschrieben.
In Kapitel TODO wird der o.g. erste Arbeitsschritt praktisch beschrieben, also die konkrete Implmentierung eines Programms für die intiale Zuordnung, eingebettet in das in dem vorangegangenen Kapitel TODO beschriebene Gesamtsystem.

3.1.1 *Aufbau des Gesamtsystems*

Der Abbildung 3.1 ist der Aufbau des Gesamtsystems zu entnehmen:

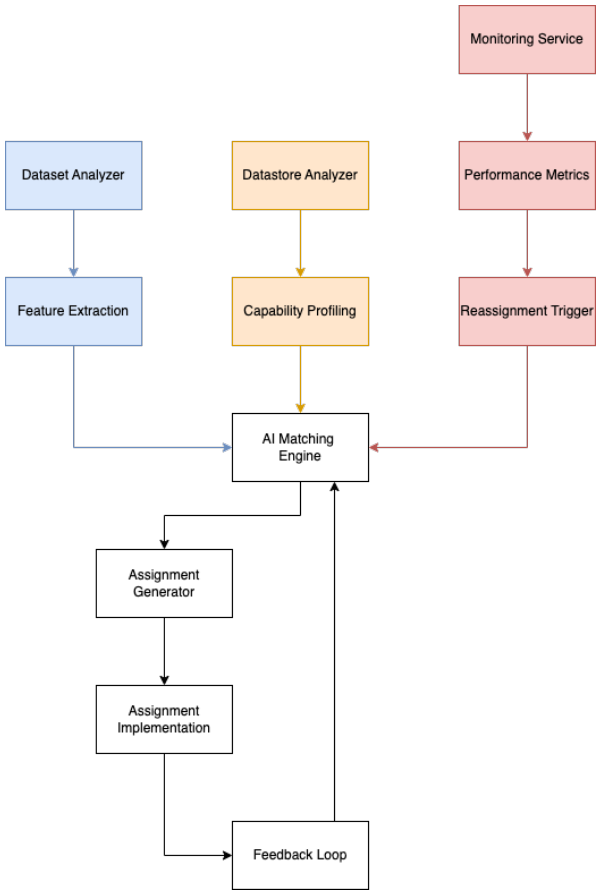

Figure 3.1: System Architecture

Das Gesamtsystem bzw. dessen Architektur kann in verschiedene Module eingeteilt werden.

3.1.1.1 *Dataset Analysis Module*

Das erste Modul ist das Dataset Analysis Module (DAM). Das DAM (blaue Färbung in 3.1) hat die folgenden drei Hauptaufgaben:

FEATURE EXTRACTION: Analyse von eigehenden Datasets um relevante Features zu extrahieren.

DETERMINE DATATYPES: Determines datatypes, structure, size, access patterns, query frequency.

FEATURE VECTOR: Creation a of feature vector for each dataset.

Zusammenfassend kann man sagen, dass das DAM Datassets als Input entgegennimmt und diese klassifiziert.

3.1.1.2 *Datastore Profiler*

Das nächte Modul ist der Datastore Profiler (DP, oranger Bereich in 3.1). Die Hauptaufgaben des DP sind die folgenden:

KATALOGISIERUNG VON DATASTORES: Catalogs available datastores with their capabilities.

PERFORMANCE MESSUNG: Measures performance characteristics for different data types

DATASTORE CAPABILITY MATRIX: Creation and maintenance of Capability Matrix for each Datastore.

Zusammenfassend ist der DP dafür verantwortlich alle verfügbaren Datastores zu katalogisieren und eine Matrix zu erstellen und zu warten, derer man die entsprechenden Fähigkeiten aber auch umgekehrt auch deren Limitationen entnehmen kann.

3.1.1.3 *Dynamic Reassignment Module*

Das Dynamic Reassignment Module (DRM, roter Bereich in 3.1) monitort das zugrunde liegende Gesamtsystem aus Datasets und Datastores. Als Input erhält das DRM System Conditions und Performance Metrics. Auf Basis des oben genannten Inputs werden Veränderungen dedektiert und reassignment getriggert, werden entsprechende Grenzwerte über- oder unterschritten werden. Kompakt sind die drei Hauptfunktionen des DRM die folgenden

SYSTEM MONITORING: Monitoring of system conditions and performance metrics.

CHANGE DETECTION: Detection of changes that warrant reassignment, based in thresholds.

DATASTORE CAPABILITY MATRIX: Triggers reassignments with minimal disruption.

3.1.1.4 *AI Matching Engine*

Das dritte Modul ist die AI Matching Engine (AME). Die AME ist dafür verantwortlich, den Input aus dem DAM (3.1.1.4), dem DP (3.1.1.2) und dem DRM (3.1.1.3) um Datassets zu Datastores zuzuordnen.

3.1.2 *System Input parameters*

Nach dem die Architektur des Gesamtsystems im Kapitel 3.1.1 erklärt wurde, sollen nun für das System relevante Inputparamter beschrieben werden.

3.1.2.1 *Dataset related Input*

Bezogen auf Datasets sind im wesentlichen die folgenden parameter von entscheidener Bedeutung:

DATA STRUCTURE : Cluster structured versus unstructured data and schema complexity.

SIZE : Current size and growth rate of the dataset.

ACCESS PATTERNS : Read-heavy versus write-heavy usage of the dataset.

QUERY COMPLEXITY : Cluster simple lookups versus complex analytics.

CONSISTENCY REQUIREMENTS : Cluster strong versus eventual consistency.

LATENCY REQUIREMENTS : Evaluate the needed response time.

DATAS RELATIONSHIPS : Check for connections to other datasets.

Auf die einzelnen Inputs wird ab dem Kapitel TODO im Detail eingegangen. Nach der Aufzählung der Dataset bezogenen Input parameter, werden nun die Input parameter bezogen auf Datastores aufgezählt.

3.1.2.2 *Datastore related Input*

Für die Beurteilung ob ein Dataset mit entsprechenden Kriterien (Vergleiche Chapter 3.1.2.1) zu einem potentiellen Datastore passt, ist eine hinreichende Beschreibung der Beschaffenheit und der characteristics der zur Verfügung stehenden Datastores erforderliche. Nachfolgend werden die Kriterien aufgezählt die in der vorliegenden Arbeit Berücksichtigung finden. An dieser Stelle sei erwähnt, dass die Inputparameter für Datassets (Chapter 3.1.2.1) aber auch die parameter bezogeb auf Datastores noch erweitert werden können. Datastore bezogenene für diese Arbeit relevante Inputparameter sind die folgenden:

TYPE OF DATASTORE : A distinction is made between Relational, document, key-value, graph based datastores.

PERFORMANCE PROFILE : Evaluation of the datastore related speed characteristics.

COST MODEL : Storage and operation costs of the related datastore.

SCALABILITY : Horizontal and vertical scaling capabilities of the related datastore.

4

DYNAMISCHE ALLOKATION UND RE-ALLOKATION VON DATASETS ZU DATASTORES

5

DATENMODELL UND DATENBANKSEITIGE UMSETZUNG VON DATASTORE RE-ALLOKATIONEN

6

SCHLUSSBETRACHTUNG