Seminararbeit im Studiengang Angewandte Informatik (BSc)

Self-Adaptive Architecture in Stream Processing

Version 1.0 vom January 22, 2020
(Vor Abgabe entfernen)

Leon Meister

285631

meister@uni-hildesheim.de

Betreuer:
MSc Cui Qin, SSE

Arbeitsgruppe Software Systems Engineering • Institut für Informatik
Universität Hildesheim • Universitätsplatz 1 • D-31134 Hildesheim

Eigenständigkeitserklärung

Erklärung über das selbstständige Verfassen von "Self-Adaptive Architecture in Stream Processing"

Ich versichere hiermit, dass ich die vorstehende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der obigen Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich in jedem Fall durch die Angabe der Quelle bzw. der Herkunft, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet und anderen elektronischen Text- und Datensammlungen und dergleichen. Die eingereichte Arbeit ist nicht anderweitig als Prüfungsleistung verwendet worden oder in deutscher oder einer anderen Sprache als Veröffentlichung erschienen. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

Hildesheim, den January 22, 2020

Leon Meister

Kurzfassung

Eine kurze Zusammenfassung der Arbeit, die Interesse beim Leser wecken soll.

Abstract

Gerne zusätzlich oder alternativ in Englisch.

Contents

List of Figures

List of Tables

Quellcode-Verzeichnis

List of Abbreviations

DPS	Data Stream Processing
EDF	Elastic and Distributed DSP Framework
IDC	International Data Corporation
SPS	Stream Processing System

1 Introduction

THIS IS FINAL

In this chapter we will explore the motivation and goals of this seminar thesis, as well its structure. In order to do this, we will explain the motivation behind the topic as well as the research questions that we aim to answer in section 1.1 Motivation and Goals. Afterwards we will then explain the thesis's structure and contents in section 1.2 Structure.

1.1 Motivation and Goals

THIS IS NOT FINAL

This chapter delves into the motivation behind this thesis and furthermore defines research questions, which we aim to answer in the subsequent chapters.

The advancements in technology of the past decades has lead to enormous data creation. Technology has become ubiquitous, with the evolution of cell phones to smartphones, the digitilization of industrial processes, Industry 4.0 and the increasing amount of "smart" devices, causing the creation of information to grow exponentially. It is estimated that the Global Datasphere will reach the size of 175 zettabytes by 2025, as shown in figure 1.1.

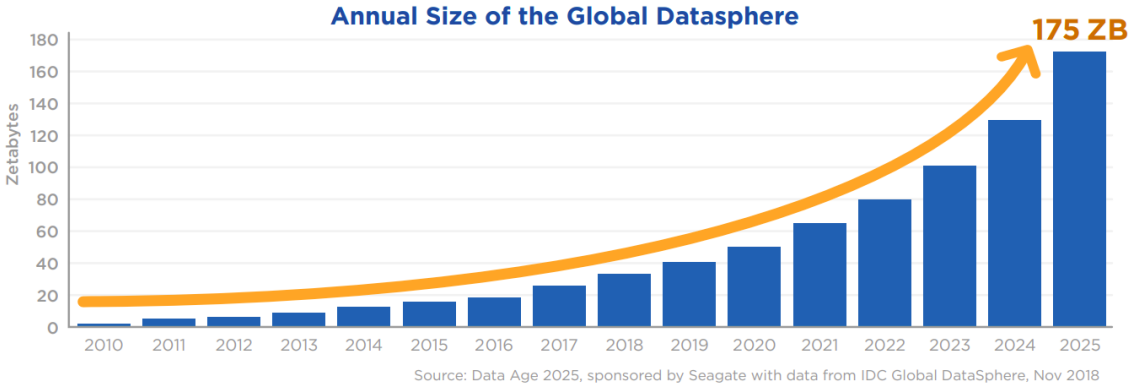

Figure 1.1: The Growth of the Global Datasphere [RGR18, p.6]

Data has become an important factor in decision making and optimization in virtually every industry, especially in finances. **TODO: Wieso? Vllt sogar diese Section ueberarbeiten** The financial market is dominated by data driven decisions, with emphasis on data processing, often in a (near) real-time fashion. However, real-time data is becoming of importance in multiple sectors; the International Data Corporation estimates that real-time data will be responsible for a share of 30 percent of the total global datasphere by 2025, as shown in figure 1.2.

A global study led by IBM in 2012 has shown that 71 percent of the firms in the financial market use information (including big data) in order to achieve an advantage over their competitors, compared to 36 percent, which IBM has found in an earlier study conducted in 2010. [TSS13, p.1]

As it is no longer feasible to save all the data before then analyzing it (in batches), due to computational cost and lack of storage capacity, a new approach was designed in order to handle data in a (near) real-time fashion, stream processing systems (SPS). A stream processing system takes in a continuous stream of data and processes each element of

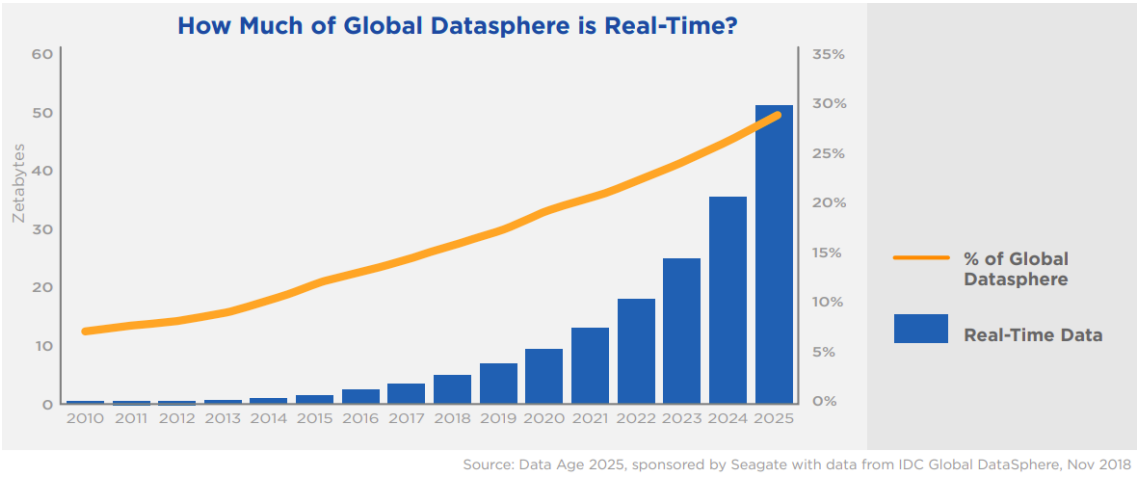

Figure 1.2: The growth of real-time data as part of the Global Datasphere [RGR18, p.13]

said stream, and then puts out a stream of processed data. These systems will further be explained in 2.1.2 Stream Processing Systems.

However the rate in which data is being streamed fluctuates, so one comes to the logical conclusion that SPSs should adapt to the velocity and volume of the stream. Besides the stream, the computing environment might also fluctuate, e.g. a resource failing in a distributed computing environment, and as such the system has to adapt. There are many different techniques of adaptation such a system might utilize, which can impact both the stream as well as the SPS. We will elaborate on this in 2.1.2 Stream Processing Systems.

In this thesis our goals are to define stream processing and to look at various adaptation techniques in stream processing. Furthermore we will clarify why self-adaptivity is needed in stream processing and what current architectures, that enable self-adaptivity in stream processing, look like.

1.2 Structure

THIS IS FINAL

In chapter 1 we will introduce the topic, motivate the thesis and explain its structure.

In chapter 2 we will go over the fundamental concepts regarding the general topic of stream processing and adaptive systems. In section 2.1 Stream Processing will be explained by first clarifying data stream management systems in 2.1.1, afterwards we will look into steam processing systems and understand how they work and the challenges they're facing. In 2.1.3 we will then discuss requirements as formulated by Stonebraker et al. and explore how they influence our decisions when talking about architecture. In section 2.2 we will then introduce the concept of self-adaptive systems and their design, including the MAPE-loop in 2.2.1.

In chapter 3 we then discuss several architectural approaches in designing self-adaptive streaming processing systems, starting with Dhalion in 3.1, followed by a decentralized approach by Cardellini et al. in 3.2. Each subsection of this chapter will contain a thorough description of the approache's architecture followed by a discussion, in which they will also be compared to their reference architecture, the MAPE-loop.

Chapter 4 will then provide a summary of our findings and conclude the thesis.

2 Fundamental Concepts

TODO: I -> We, no need to always say will, sometimes use passive to switch it up In this chapter I will lay out the fundamental concepts, which are necessary in order to understand the subsequent chapters. First I will explain Stream Processing in 2.1, afterwards the concept of the MAPE-K Loop will be presented and explained in subchapter **??** followed by a brief explanation of self-adaptive systems in 2.2 to conclude this chapter.

2.1 Stream Processing

In this section I will split the concept of Stream Processing into three further components. In 2.1.2 I will then define Stream Processing Systems, explain how they work and give examplary fields of application. Afterwards in 2.1.1 I will then move onto the topic of Data Stream Management Systems and finally in 2.1.3 I will talk about some requirements that SPSs should meet.

2.1.1 Data Stream Management Systems

THIS IS FINAL

NOT CLEAR ENOUGH, EXPLAIN LATENCY, CREATE DIAGRAM TO SHOW DIFFERENCE DBMS DSMS A Data Stream Management is a system designed to manage continuous data streams. Similar to a DBMS a DSMS can also be queried using a streaming query language, however, queries installed on a DSMS are continuous and will be executed as long as it is uninstalled. This leads to everchanging results, due to changing incoming data in contrast to querying a DBMS and receiving a static result.

Another important difference is that the data being fed into the system is not saved like in DBMSs but instead only synopses, in an attempt to summarize the data.

Most notably the data being fed into a DSMS varies greatly from the data being inserted into a DBMS; while a DBMS receives a finite predetermined amount of data, a DSMS could theoretically receive an infinite continuous stream. There are a few more functional differences, which Panigati, Schreiber and Zaniolo highlight, as shown in table 2.1.

Feature	**DBMS**	**DSMS**
Model	Persistent data	Transient Data
Table	Set or bag of tuples	Infinite sequence of tuples
Updates	All	Append only
Queries	Transient	Persistent
Query answers	Exact	Often approximate
Query evaluation	Blocking and non-blocking	Non-blocking
Operators	Fixed	Adaptive
Data processing	Synchronous	Asynchronous
Concurrency overhead	High	Low

Table 2.1: Functional comparison of DBMS and DSMS[PSZ15]

2.1.2 Stream Processing Systems

THIS IS NOT FINAL

TODO: Explain big data -> batch(latency low prio) and stream(latency high prio) In this subsection I will explain what SPSs are, what their input looks like, what SPSs are able to do and give an example. In the end of this subsection I will also go over the challenges that SPSs face.

Stream processing systems are fed continuous data streams, generated by data providers, for example GPS-sensors or EEG- and EKG-machines. Per definition of Panigati, Schreiber and Zaniolo[PSZ15] a data stream S consists of a infinite, countable series of four-tuples $s := < \nu, t^{app}, t^{sys}, b_{id} >, s \in S$, where:

TODO: Replace this with a more general definition, e.g. from fundamentals of stream processing book

- $\nu \in \Re$ is a relational tuple
- $t^{app} \in T$, with T being the time domain, is a partially ordered application time
- $t^{sys} \in T$, with T being the time domain, is a totally ordered system time
- $b_{id} \in B$ is a batch id; batch B is a finite subset of S where all $b \in S$ have an identical t^{app}

Each element s then is processed by one or more operators, each doing their own independent operation, e.g. filtering as shown in figure **?? (somehow ref wrong?)** and then put out into the operator's outgoing stream.

TODO: Implement some structure, not own sections but little headers maybe or new paragraphs or something -> explain windows

-> exanpe aggregation Tumbling

-> example aggregation sliding

-> elaborate Difference

In order to increase efficiency, an SPS can, if (computational) resources are available, create replicas of operators to introduce parallelity, as shown in figure 2.1.2 **Ref wrong?**. Conversely, if there is little input, it may also reduce the amount of replicas in order to save or free up additional resources.

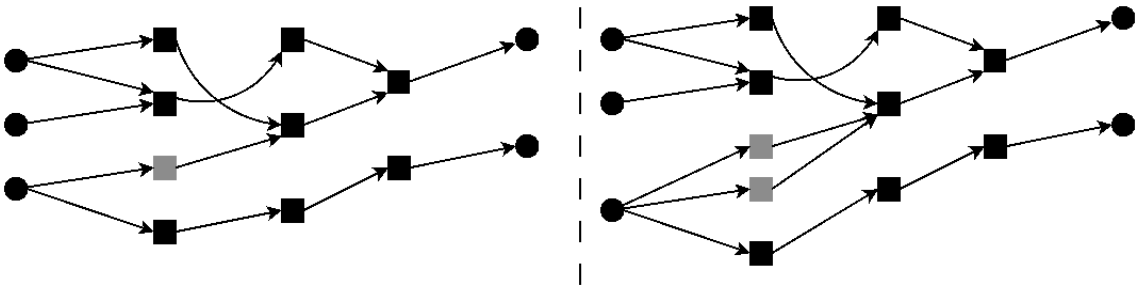

Figure 2.1: Left: An example for an SPS displayed as a directed acyclic graph. Right: Same SPS with introduced parallelity in one operator, marked gray for visibility. Circles are input/outputs, squares are operators, arrows are streams. TODO: EXPLAIN graph and sink, generator, processor!!

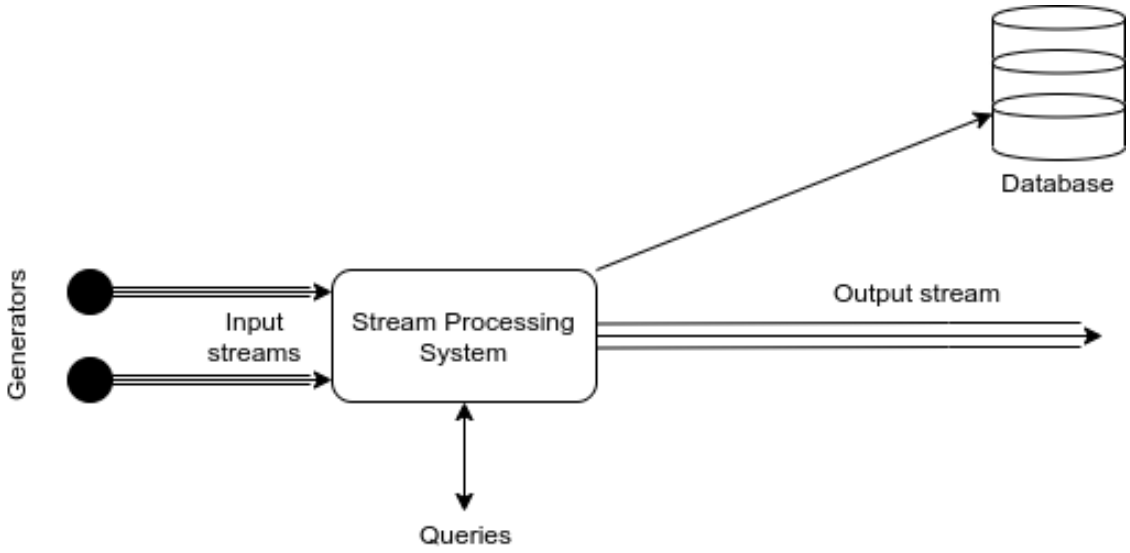

Figure 2.2: Overview of a basic Stream Processing System

2.1.3 Requirements for Stream Processing Systems

THIS IS NOT FINAL

Notes: What are the requirements, why do they matter to us (Elaborate on this)

Due to the nature of the fields in which SPS are used, there are important requirements that SPS should meet in order to be viable, which Stonebraker et al. point out in [ScZ05], of which the ones most important to us can be summarized as the following:

1. **Keep the Data Moving:** In order to minimize latency, data must not be stored, as these are costly operations.
2. **Handle Stream Imperfections:** Expecting only perfect data is utopian, so one must prepare the system with built-in mechanisms for data that might be missing or out-of-order.
3. **Integrate Stored and Streaming Data:** For an SPS to be able to perform comparisons between "predecessor" data and current data, operators must keep an efficiently manageable state.
4. **Guarantee Data Safety and Availability:** Recovering from a failure is detrimental for real-time data processing, so a system must be in place to guarantee the highest availability possible.
5. **Process and Respond Instantaneously:** Systems must be highly optimized in order to provide (near) real-time responses.
6. **Partition and Scale Applications Automatically:** Systems must be able to be split across multiple machines and threads. The system must also be able to automatically scale and distribute the load across the machines.

2.2 Self-Adaptive Systems

THIS IS FINAL?

2.2.1 MAPE-K Loop

THIS IS FINAL (Maybe add something underneath enum if not sufficient)

The MAPE-K Loop was introduced by IBM [KC03] and refers to a proposed solution for self-adaptive or autonomic systems. This model has since become the basis or reference architectural pattern for many self adaptive systems, which I will show in the third chapter. The acronym MAPE-K refers to the components that make up the model: **TODO: Add a diagram, make this a subsection of self-adaptive systems**

1. **M**onitor: The *Monitor* component gathers data about the system and its environment, aggregates and filters it.
2. **A**nalyze: The *Analyze* component analyzes the previously gathered data and determines whether or not an adaptation should be performed. The decision is made based on performance or cost gain and should include the adaptation cost as well. This component's analysis is influenced by the *Knowledge* base.
3. **P**lan: If the choice to adapt the system has been made, the *Plan* component then decides how to reconfigure the system. Once the decision has been made, the information is then forwarded to the *Execute* component.
4. **E**xecute: Given the *Plan* component's decision, the *Execute* component then executes said plan and the loop returns to the initial monitoring state.
5. **K**nowledge: Represents the knowledge base, which is shared between the other components. This base is created by the *Monitor* component and contains information in the form of metrics, policies, symptoms and logs.

Needs more; e.g why self-adaptive(already mentioned in intro), Software engineering perspective on self adaptivity, challenges Cheng et al. define self-adaptive systems as

> "[...] systems that are able to adjust their behaviour in response to their perception of the environment and the system itself [...]"[CLG+09, p.1].

Self-adaptive systems are oftentimes based on the **??** [p.**??**] pattern. Adaptive Systems have a wide variety of possible application areas: adaptable user interfaces, autonomic computing, multi-agent systems [CLG+09], biologically inspired computing, robotics [LSC18], streaming applications and a lot more.

An examplary application would be a scenario, in which population and food capacities are given and evolving over time, due to births, deaths, changes in demographics and changes in weather and harvest respectively. A system would have to adapt to these changes in its environment in order to ration the food properly.

3 Approaches for Self-Adaptive Architectures in Stream Processing

TODO: How does it work? FOCUS ON ARCHITECTURE IN THIS CHAPTER!! Explain that this chapter showcases a few select strategies, which are then elaborated on further in the subchapters Question: Even more approaches? e.g. Master-Slave pattern or Coordinated Control pattern (Both MAPE based)? **Add and explain a few more MAPE Based architectures**

3.1 Dhalion

Quick Introduction to Dhalion, this chapter will deal with the Dhalion paper. NOTE: Explain what Dhalion is, where its used

3.1.1 An Outline of Heron

Small outline of Heron, as Dhalion is built on top of Twitter's Heron.

3.1.2 Dhalion's Architecture

TODO: Diagram, explain it here Explanation of Dhalion's Architecture **KERNPUNKT DER SECTION DHALION**

3.1.3 Discussion of Dhalion

Discuss the approach and compare it to the reference architecture (Mape?) **TODO: Maybe discuss how they evaluate, look at metrics relevant to architecture**

3.2 Decentralized Self-Adaptation

TODO: Incorporate same structure as above NOTE: In this section I will explain the hierarchical control architecture as decribed by Cardellini..

3.2.1 Elastic and Distributed DSP Framework

TODO: Possibly change title of this, add diagram, explain thoroughly (decentralized etc.), extract their design patterns Explanation of the EDF, their architecture for elastic DSP apps **KERNPUNKT DER SECTION**

3.2.2 Discussion of EDF

3.3 Some Other Architecture

NOTE: In this chapter I will explain another architecture/approach to self-adaption, yet to be researched

3.4 Some Other Architecture2

NOTE: In this chapter I will explain another architecture/approach to self-adaption, yet to be researched

3.5 Title??

TODO: Discuss among all of them, critical thinking..

TODO: If enough material compare the architecture relevant metrics of the approaches

4 Summary And Conclusion

4.1 Summary

Summarize the paper

4.2 Conclusion

Conclude the paper

A Anhang

A.1 Beispiele

Zitat ohne Seitenangabe: [**?**]

Zitat mit Seitenangabe: [**?**, S.1]

Referenz eines Glossareintrags:

Eine normale Liste:

- Ein Punkt
- Ein anderer Punkt

Eine nummerierte Liste:

1. Erstens
2. Zweitens
3. ...

Eine Tabelle:

Table A.1: Eine Tabelle

Spalte 1	Spalte 2	Spalte 3
1	2	3

Fetter Text

Kursiver Text

Eine Referenz: Siehe Tabelle A.1: Eine Tabelle auf Seite 9.

Ein Bild:

Figure A.1: Das Logo der SUH

Eine abgesetzte Formel:

$$\sum_{n=0}^{3} n = 6 \tag{A.1}$$

Eine Formel im Fließtext: $2^2 = 4$

Ein Listing aus einer Datei:

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World!");
4     }
5 }
```

Listing A.1: HelloWorld

Ein 'on-the-fly' erstelltes Listing:

```
1 (Sun Sep 13 23:02:20 2009): ODBC Driver <system32>\wbemdr32.dll not present
2 (Sun Sep 13 23:02:20 2009): Successfully verified WBEM OBDC adapter (incompatible version
      removed if it was detected).
3 (Sun Sep 13 23:02:20 2009): Wbemupgd.dll Registration completed.
```

Listing A.2: Beispiel eines Log-Eintrags

Glossary

Global Datasphere The Global DataSphere quantifies and analyzes the amount of data created, captured, and replicated in any given year across the world [SOURCE https://www.idc.com/getdoc.jsp?co

Bibliography

[CLG+09] Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[LSC18] V. Seidita L. Sabatucci and M. Cossentino. The four types of self-adaptive systems: A metamodel. In R. J.Howlett G. De Pietro, L. Gallo and L. C. Jain, editors, *Intelligent Interactive Multimedia Systems and Services 2017*, pages 440–450, Cham, 2018. Springer International Publishing.

[PSZ15] Emanuele Panigati, Fabio A. Schreiber, and Carlo Zaniolo. *Data Streams and Data Stream Management Systems and Languages*, pages 93–111. Springer International Publishing, Cham, 2015.

[RGR18] David Reinsel, John Gantz, and John Rydning. The digitization of the world. November 2018.

[ScZ05] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4):42–47, December 2005.

[TSS13] David Turner, Michael Schroeck, and Rebecca Shockley. Analytics: The real-world use of big data in financial services. May 2013.