Bachelorarbeit

13. August 2013

Benchmarking Big Data Streams

Joins in Informationsstrom-Verarbeitungssystemen

Livio Hobi

of Mels, Schweiz (10-719-938)

supervised by

Prof. Abraham Bernstein, PhD

Dr. Thomas Scharrenbach

University of Zurich
Department of Informatics

University of Zurich
Department of Informatics

Bachelorarbeit

Benchmarking Big Data Streams

Joins in Informationsstrom-Verarbeitungssystemen

Livio Hobi

University of Zurich
Department of Informatics

University of Zurich
Department of Informatics

Bachelorarbeit

Author: Livio Hobi, livio.hobi@uzh.ch

Project period: 14.03.2013 - 14.09.2013

Department of Informatics, University of Zurich

Disclaimer. *This thesis template is provided by the s.e.a.l. group (http://seal.ifi.uzh.ch) of the University of Zurich, Department of Informatics (ifi) as a service to ifi students on an "as-is, as-available" basis for (master/bachelor/diploma) thesis layouting purposes only. We assume no responsibility for any errors or omissions in the template and make no commitment to update the template on a regular basis.*
ifi students may download and use the thesis template for personal and non-commercial use only, without altering or removing any copyright or other notice from the template. For the proper use of the template, the Frutiger Condensed font of the University of Zurich has to be installed.
Thesis template © 2009, s.e.a.l – University of Zurich, Switzerland

Acknowledgements

An dieser Stelle bedanke ich mich bei allen, die mich bei dieser Arbeit unterstützt haben. Ein herzliches Dankeschön gebührt Prof. Abraham Bernstein, PhD, der es mir ermöglicht hat, meine Bachelorarbeit in seiner Forschungsgruppe zu schreiben. Mein Betreuer, Dr. Thomas Scharrenbach, hat mich mit sehr vielen Tipps, fachlichem Input und sehr grossem Engagement unterstützt. Bei anstehenden Fragen wusste er stets Rat und konnte weiterhelfen. Dafür bin ich sehr dankbar. Ein spezieller Dank geht an meine Schwester, die sich die Mühe gemacht hat, das Endprodukt dieser Arbeit zu korrigieren. Du bist die Beste. Zu guter Letzt danke ich meiner Familie und meinen engsten Freunden für ihr Interesse an meinem Thema und ihre moralische Unterstützung. Sie halfen mir vorübergehend abzuschalten, um später mit frischer Energie und Motivation die Arbeit fortzuführen. Herzlichen Dank an alle, die mich während meiner Bachelorarbeit begleitet haben.

Abstract

This thesis investigates an approach for systematically stress-testing Information flow processing (IFP) systems with sequential joins. The main contribution of this work is not only the result of the evaluation of a fife-way sequential join but also the provided methodic approach going through the properties and challenges of joins in the context of IFP systems, the described data preparation and the statistical evaluation based on some defined key performance indicators. This approach can simplify future work. Furthermore this work emphasizes the immense importance of knowing and selecting the dataset for a benchmark. For example if the window operator of a query is too small, the system may never output anything but possibly has a great throughput. Therefore the developer of a benchmark needs to know some statistics about the dataset and then choose the parameters that fit the defined requirements.

In contrast to traditional database systems, the data in IFP applications are rather dynamic than static. The inputs of such systems, called Information flows, are infinite and change over the time. The data in IFP systems are typically never stored permanently. They enter the system, get processed and leave the system. Thanks to window operators in the query languages of IFP engines, the input data can be splitted into pieces such that they can be processed. IFP systems have the requirement to respond in near real-time. Different kinds of IFP systems meet different challenges. This, in particular, makes it hard to compare such systems. Where are their strengths and - more important - where are their weaknesses?

There already exist some benchmarks to compare IFP systems. Scharrenbach *et al.* (2013) state that these benchmarks for systems with semantically annotated data have not been developed with respect to all the properties and challenges of such systems. Key aspects are missing or not completly integrated. This thesis takes the generic commandments of benchmarking semantically flow processing systems from Scharrenbach *et al.* (2013) and uses them to develope an approach for systematically stress-testing joins based on their properties and challenges in IFP systems. The different kinds, properties and challenges of simple, temporal and sequential joins are analyzed. Furthermore the thesis contains an evaluation of an IFP system tested with a five-way sequential join with different windows on a real word dataset of IPTV user information. The results of the evaluation show that different sizes of windows in the query have a remarkable impact on the throughput, the recall, the precision and the response time. Furthermore it demonstrates that a high datarate with more possible query results can stress a system. A high datarate has thus an impact on the throughput and the response time because it can lead to more computation costs.

Zusammenfassung

Diese Arbeit beschreibt ein methodisches Vorgehen, um sequentielle Joins in Informationsstrom-Verarbeitungssystemen (ISV-Systeme) systematisch zu testen. Als Hauptbeitrag dieser Arbeit ist nicht nur das Resultat des durchgeführten Experiments eines fünffachen sequentiellen Joins, sondern das methodische Vorgehen von den Eigenschaften und Herausforderungen der Joins in ISV-Systemen, bis hin zur Datenaufbereitung und der statistischen Analyse zu betrachten. Dieses Vorgehen und die dazugehörigen Python-Skripte können für zukünftige Forschungen wiederverwendet werden. Weiter betont diese Arbeit, wie wichtig die Auswahl des Datensatzes für eine Benchmark ist (gerade wenn sequentielle Joins getestet werden sollen). Beispielsweise bringt eine Abfrage mit einem achtfachen sequentiellen Join wenig, wenn im Datensatz nur Fünfer-Ketten vorhanden sind. Dasselbe gilt für andere Parameter wie die Wahl des Fenster-Operators etc. Die Ergebnisse des Experiments zeigen, dass unterschiedliche Grössen der Zeitfenster einen beachtlichen Einfluss auf den Durchsatz, die Trefferquote, die Präzision und die Antwortzeit haben. Weiter setzt sich diese Arbeit mit den Herausforderungen von schwankenden Datenraten in ISV-Systemen auseinander. Die schwankende Datenrate im verwendeten Datensatz konnte das System insofern stressen, als dass es durch eine höhere Datenrate mehr Resultate zu berechnen gab, was eine höhere Antwortzeit und einen tieferen Durchsatz ergibt.

Die unendlichen, ständig wechselnden Informationen, welche solche Systeme über die Zeit verarbeiten müssen, verlangen nach einem Ausführungsmodell, das sich stark von traditionellen Datenbanksystemen unterscheiden muss. Die Informationen werden prinzipiell nicht permanent gespeichert, sondern fliessen durch das System. Spezielle Fenster-Operatoren in den Abfragesprachen für ISV-Systeme helfen, die Informationsströme in Teilen zu verarbeiten. Vielfach besteht in solchen Systemen die Anforderung, Antworten in Echtzeit zu liefern. Die verschiedenen Arten der Systeme, ihre Herausforderungen und die Heterogenität in den Anwendungszwecken machen es schwer, ISV-Systeme zu vergleichen. Welche System haben worin ihre Stärken, oder noch wichtiger, ihre Schwächen? Wann und wie gerät ein System an seine Grenzen?

Es bestehen bereits einige Benchmarks um ISV-Systeme zu vergleichen. Gemäss Scharrenbach *et al.* (2013) wurden diese Benchmarks jedoch nicht systematisch, d.h. basierend auf den speziellen Eigenschaften und Herausforderungen von ISV-Systemen entwickelt bzw. werden wichtige Aspekte nicht berücksichtigt und integriert. Es fehlt eine methodische Herangehensweise. Diese Arbeit nimmt die generischen Vorschläge für eine Evaluation von ISV-Systemen mit semantischen Daten von Scharrenbach *et al.* (2013) auf und wendet diese auf die speziellen Eigenschaften und Herausforderungen von Joins im Kontext der Informationsstrom-Verarbeitung an. Dazu werden die Arten, Eigenschaften und Herausforderungen von einfachen, temporalen und sequentiellen Joins analysiert und beschrieben. Es wird ein Experiment mit einem fünffachen sequentiellen Join mit unterschiedlichen Zeitfenstern und einem realen Datensatz von IPTV-Benutzerlogs durchgeführt. Der Datensatz wird ausgiebig statistisch analysiert und die Parameter in dem durchgeführten Experiment basierend auf diesen Erkenntnissen gewählt.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Ziele der Arbeit . . . 2
1.2 Hypothesen, um sequentielle Joins zu testen . . . 2
1.3 Aufbau . . . 3

2 Methoden **5**
2.1 Informationsstromverarbeitung . . . 5
2.1.1 Übersicht . . . 5
2.1.2 Terminologie . . . 6
2.1.3 Architektur . . . 6
2.1.4 Datenstrom-Verarbeitungssystem . . . 6
2.1.5 Verarbeitungssystem für komplexe Events . . . 8
2.1.6 Eigenschaften . . . 8
2.2 Arten von Joins . . . 10
2.2.1 Join-Operationen und Joins . . . 10
2.2.2 Einfache Joins . . . 11
2.2.3 Temporale Joins . . . 11
2.2.4 Sequentielle Joins . . . 12
2.3 Eigenschaften von Joins in ISV-Systemen . . . 12
2.4 Herausforderungen . . . 13
2.4.1 Herausforderungen bei einem Join mit schwankenden Datenraten der Informationsströme . . . 13
2.4.2 Herausforderungen bei sequentiellen Joins . . . 14
2.5 Key Performance Indicators . . . 15

3 Evaluation **17**
3.1 Verwendete Technologien und Infrastruktur . . . 17
3.1.1 Streams-Esper . . . 17
3.1.2 PostgreSQL . . . 18
3.1.3 Kraken . . . 18
3.2 Output eines Tests mit Streams-Esper für spätere Evaluationen . . . 18
3.3 Experimente mit ViSTA-TV-Daten . . . 18
3.3.1 Was ist ViSTA-TV? . . . 19
3.3.2 Datenaufbereitung und Statistiken . . . 19
3.3.3 Fünffacher sequentieller Join . . . 23

4 Diskussion **31**
4.1 CEP-Systeme 31
4.2 Auswahl und Analyse eines Datensatzes für eine Benchmark 31
4.3 Experiment des fünffachen sequentiellen Joins 32
4.4 Stand der Forschung 33
4.5 Limitationen der Arbeit 35

5 Zusammenfassung und Ausblick **37**

6 Anhang **39**

A Output von Streams-Esper **41**

B Datenaufbereitung und Statistiken **43**
B.1 Datenaufbereitung 43
B.2 Weitere Statistiken zum verwendeten Datensatz 47

C Fünffacher sequentieller Join **49**
C.1 Aufbau des Experiments 49
C.2 Auswertungsskript 54
C.3 Weitere Statistiken zu den Resultaten 54
C.3.1 Ausführungszeiten 54
C.3.2 Durchsatz 57
C.3.3 Ausgegebene Resultate und Trefferquote 58

Abbildungsverzeichnis

2.1 Architektur eines ISV-Systems. (Quelle: Cugola and Margara (2012)) 7
2.2 Typische Architektur eines DSMS. (Quelle: Cugola and Margara (2012)) 8
2.3 Typische Architektur eines CEP-Systems. (Quelle: Cugola and Margara (2012)) . . 8
2.4 Allans 13 mögliche Beziehungen zwischen Intervallen. (Quelle: Allen (1983)) . . . 12
2.5 Dreifacher sequentieller Join. 13

3.1 Verteilungen der Zeitdifferenzen zwischen zwei Join-Partnern in Datenzeit für alle Resultate, die der fünfte Join liefert. x-Achse: Zeitdifferenz zwischen zwei Join-Partner in Sekunden, y-Achse: Prozent. 21
3.2 Verteilung der Anzahl ankommender Tupel/Sekunde zwischen zwei Join-Partnern in Datenzeit für die Joins 1-5 und insgesamt. x-Achse: Tupel/Sekunde, y-Achse: Prozent. 23
3.3 Anzahl vorhandener Events im verwendeten Datensatz pro Minute in Datenzeit, welche für die Joins relevant sind. Da ab 12:50 Uhr keine weiteren Benutzerdaten mehr im Datensatz sind, stoppt die x-Achse nach 741 Minuten, was der Uhrzeit 12:21 entspricht. 24
3.4 Verteilung der Antwortzeiten in Systemzeit der Joins 1-5 und gesamt. X-Achse: Sekunden in Systemzeit, y-Achse: Prozent. 27
3.5 Ausgegebene Tupel/Sekunde in Systemzeit für den Job mit einer Trefferquote von 100%. X-Achse: Sekunden in Systemzeit beschriftet mit dem ausführenden Join, y-Achse: Anzahl ausgegebener Resultate. 29

Tabellenverzeichnis

2.1 Verschiedene Fensterarten. 10
2.2 Verschiedene einfache Joins. 11

3.1 Anzahl Resultate pro Join. 20
3.2 Deskriptive Statistiken der Zeitdifferenzen zwischen zwei Join-Partnern in Datenzeit für die Joins 1-5 und insgesamt. 21
3.3 Deskriptive Statistiken über die ankommenden Tupel/Sekunde zwischen zwei Join-Partnern in Datenzeit für die Joins 1-5 und insgesamt. 22
3.4 Deskriptive Statistiken über die Antwortzeiten in Systemzeit der 1'024 Jobs in Sekunden. 26
3.5 Die 5 grössten und kleinsten Ausreisser bezüglich Antwortzeiten zusammen mit dem gesamten Zeitfenster in Sekunden. 26
3.6 Die fünf grössten und kleinsten Ausreisser verarbeitet bzw. ausgegeben zusammen mit dem gesamten Zeitfenster in Sekunden. 28

List of Listings

3.1 SQL-Abfrage in PostgreSQL des dritten sequentiellen Joins. 20
3.2 SQL-Abfrage für die Anzahl zu verarbeitender Tupel für Join 1 22
3.3 Esper-Abfrage des zweiten Joins. 25

Kapitel 1

Einleitung

Über die letzten Jahre entstand eine wachsende Anzahl an Informationsstrom-Verarbeitungssystemen (ISV-Systemen). Diese Systeme kommen zum Einsatz, wenn die zu verarbeitenden Daten aufgrund beschränkter Ressourcen nicht permanent gespeichert werden können, oder durch die Speicherung der Daten kein Informationsgewinn erzielt werden kann und diese deshalb nicht gesammelt werden müssen. Beispiele für Anwendungsfälle reichen von Online-Zeitungen, Blogs und sozialen Netzwerken bis hin zu Sensornetzwerken, welche die Umwelt überwachen, Wettervoraussagen, Verkehrsanalysen und finanziellen Analysen der Aktienmärkte (Cugola and Margara, 2012). In solchen Systemen sind die Daten nicht statisch, sondern dynamisch. Da die Datenströme unendlich gross sind und unterschiedliche Datenraten aufweisen können, sind diese Systeme mit komplexen Herausforderungen und Problemen konfrontiert. Im Gegensatz zu traditionellen Datenbanksystemen mit fixen Daten und wechselnden Abfragen, haben es ISV-Systeme mit dynamischen Daten und wiederholenden Abfragen zu tun (Scharrenbach *et al.*, 2013).

Die Anforderungen an ISV-Systeme sind schwer zu meistern. Es werden Echtzeit-Antworten auf Anfragen erwartet. Die Optimierung dieser Abfragen erweist sich als schwierig, weil die Daten sich über die Zeit ändern, nicht gespeichert und indexiert werden können und die Datenrate vielfach unbekannt ist. Vorhandene Ressourcen sind beschränkt. Was macht man also mit Daten, wenn der Hauptspeicher oder die Festplatte voll ist? Welche Daten sind für weitere Ergebnisse noch relevant? Welche können vergessen werden? Wie werden komplexe, zusammenhängende Ereignisse entdeckt?

Zeit spielt in diesen Systemen eine bedeutende Rolle. Sie ermöglicht, Ereignissen eine Reihenfolge zuzuordnen, Sequenzen von Ereignissen zu entdecken und zu verarbeiten. Traditionelle Datenbanksysteme unterstützen dies vielfach nicht.

Mit dem Aufkommen des Semantic Web werden zunehmend auch Systeme entwickelt, welche Datenströme mit semantischen Daten in der Form von RDF-Tripeln verarbeiten können (Scharrenbach *et al.*, 2013). Scharrenbach *et al.* (2013) nennen diese Art von Verarbeitungssystemen Semantic Flow Processing (SFP) Systeme. Dadurch, dass die Daten eine Semantik besitzen, können komplexere Berechnungen wie zum Beispiel Schlussfolgerungen (engl. inference) durchgeführt werden.

Es ist schwierig, solche heterogene Systeme mit unterschiedlichen Einsatzzwecken zu vergleichen und herauszufinden, welches System worin seine Stärken und Schwächen hat. Weder eine methodische und empirische Evaluation der Leistung noch das Verständnis der Eigenschaften und Herausforderungen solcher Systeme sind vollständig erforscht (Scharrenbach *et al.*, 2013). In bestehenden Leistungsbenchmarks gilt meist nur der Durchsatz (Anzahl verarbeiteter Tupel pro Zeiteinheit) als Key Performance Indicator (KPI) (Scharrenbach *et al.*, 2013). Ein hoher Durchsatz ist jedoch kein Indikator für korrekte Resultate oder schnelle Antwortzeiten. Zusätzlich werden die Eigenschaften und Funktionalitäten der Systeme nicht analysiert (bzw. berücksichtigt).

Es fehlen Ansätze, eine Benchmark systematisch aufzubauen. Scharrenbach *et al.* (2013) haben dies erkannt und eine generische Methode für die Evaluation der Leistung von SFP-Systemen entwickelt. Es werden die Eigenschaften von SFP Systemen beschrieben, deren Herausforderungen identifiziert und Key Performance Indicators (KPIs) definiert. Weiter werden sieben Vorschläge veröffentlicht, wie die Leistung von SFP Systemen anhand der KPIs evaluiert werden können. Ein Leitfaden, um eine Benchmark methodisch aufzubauen, ist gegeben. Die Implementierung fehlt jedoch noch. Die vorliegende Arbeit versucht einen Teil dieser Lücke zu schliessen, indem einfache, temporale und sequentielle Joins in ISV-Systemen analysiert werden. Wie kann man ein System damit testen? Wie muss eine Abfrage gestaltet werden, um ein System an seine Grenzen zu bringen bzw. zu stressen?

1.1 Ziele der Arbeit

Diese Arbeit vertieft die Thematik der Joins von Informationsströmen. Ziel ist, eine Methode zu finden, einen Stresstest für ein ISV-System mit Hilfe von einfachen, von temporalen und von sequentiellen Joins zu entwickeln. Dazu werden die verschiedenen Arten von Joins identifiziert und deren Eigenschaften analysiert. Basierend darauf werden Herausforderungen abgeleitet und Hypothesen für einen Stresstest aufgestellt. Diese sollen in einem Experiment getestet und ausgewertet werden. Eine abschliessende Diskussion reflektiert die Resultate des Experiments und stellt dessen Nützlichkeit und Limitationen dar.

1.2 Hypothesen, um sequentielle Joins zu testen

Dieses Kapitel führt Hypothesen ein, um sequentielle Joins in ISV-Systemen zu testen. Die Hypothesen basieren auf den Grundlagen der Eigenschaften von Joins in Kapitel 2.3 und der Herausforderungen eines sequentiellen Joins in Kapitel 2.4.
Hypothese:
Ein Informationsstrom-Verarbeitungssystem kann über folgende Parameter mit sequentiellen Joins gestresst werden:

- *Datenrate der Informationsströme*
- *Datenmenge der Informationsströme*
- *Rechenaufwand der einzelnen Joins*
- *Unterschiedliche Zeitfenster*
- *Anzahl sequentieller Joins*

Anhand der KPIs (Kapitel 2.5) soll analysiert werden, wie sich diese verändern, wenn an den beschriebenen Parametern herumgeschraubt wird. Damit letztlich Rückschlüsse auf die Auswirkungen eines Stresstests gezogen werden können, sollten diese systematisch komplexer gemacht werden. Zu Beginn z.B. mit einer schwankenden Datenrate in einem der sequentiellen Joins; später in mehreren Joins mit einer erhöhten Anzahl an Informationsströmen pro Join usw.

Systematischer Test Ein systematischer Test enthält folgende Elemente:

- Schwankende Datenrate eines, mehrerer, aller Informationsströme
  Regelmässig, konstant oder unregelmässig hohe Datenraten
- Anzahl Informationsströme eines, mehrerer, aller Joins erhöhen
- Komplexere Join-Konditionen eines, mehrerer, aller Joins einführen
- Anzahl sequentieller Joins erhöhen
- Zeitfenster eines, mehrerer, aller Joins variieren

Unter einer regelmässig hohen Datenrate ist zu verstehen, dass die Datenrate immer nach einer bestimmten Zeit steigt. Interessant dabei ist zu sehen, ob das System dies erkennt und aus den gewonnen Erkenntnissen die vorhandenen Ressourcen besser einsetzen kann.

Da ein vollständiger, systematischer Test den Aufwand dieser Arbeit überschreiten würde, wird in Kapitel 3 ein ISV-System, Streams-Esper (Kapitel 3.1.1), mit einem temporalen, sequentiellen fünffachen Join getestet, welcher fünf Datenströme mit schwankenden Datenraten sowie variablen Zeitfenster enthält.

1.3 Aufbau

Die Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden die für die Arbeit relevanten Methoden beschrieben. Diese führen über eine generelle Einleitung über ISV-Systeme mit einer Charakterisierung der verschiedenen Arten und Eigenschaften der Systeme (Kapitel 2.1), zum eigentlichen Thema dieser Arbeit: Joins (ab Kapitel 2.2). Einfache, temporale und sequentielle Joins werden eingeführt und deren Eigenschaften (Kapitel 2.3) und Herausforderungen (Kapitel 2.4) im Kontext von ISV-Systemen geschildert. Anschliessend folgt die Definition der KPIs in Kapitel 2.5, anhand derer die Experimente im Evaluationskapitel 3 ausgewertet werden. Im Kapitel 3.1 werden die verwendeten Technologien und die Infrastruktur beschrieben. Das Kapitel 3.3.2 zeigt die Datenaufbereitung und verschiedene, für das Experiment interessante Statistiken der Daten. Im Anschluss folgen der Aufbau des Experiments eines fünffachen sequentiellen Joins (Kapitel 3.3.3) zusammen mit den Resultaten. In Kapitel 4 werden Systeme für die Verarbeitung komplexer Event im Allgemeinen (Kapitel 4.1), die Wichtigkeit des verwendeten Datensatzes für eine Benchmark (Kapitel 4.2) und die Resultate des fünffachen sequentiellen Joins (Kapitel 4.3) mit Bezug zu den Aufgestellten Hypothesen in Kapitel 1.2 diskutiert. Anschliessend werden die drei Benchmarks Linear Road, SRBench und SLD-Bench beschrieben und kommentiert sowie die Limitationen der Arbeit aufgezeigt. Das Kapitel Zusammenfassung und Ausblick (Kapitel 5) rundet die Arbeit ab.

Kapitel 2

Methoden

Dieses Kapitel dient als Grundlage für die Hypothesen in Kapitel 1.2. Es werden verschiedene ISV-Systeme (Kapitel 2.1) sowie verschiedene Arten von Joins (Kapitel 2.2) analysiert und beschrieben. Zudem werden Eigenschaften von Joins im Kontext der Informationsstromverarbeitung diskutiert (Kapitel 2.3) und deren Herausforderungen geschildert (Kapitel 2.4). Zum Schluss werden in Kapitel 2.5 drei Key Performance Indicators eingeführt, anhand derer eine Auswertung der Tests in Kapitel 3 erfolgt. Dieses Kapitel basiert auf den Erkenntnissen von Scharrenbach *et al.* (2013), der Arbeit von Cugola and Margara (2012) sowie dem Datenbankbuch "Encyclopedia of Database Systems" von Liu and Özsu (2009).

2.1 Informationsstromverarbeitung

Dieses Kapitel soll eine Einführung in ISV-Systeme geben. Nach einer kurzen Übersicht in Kapitel 2.1.1 wird in Kapitel 2.1.2 die Terminologie, wie sie in dieser Thesis verwendet wird, erklärt. Anschliessend wird eine generische Architektur für alle ISV-Systeme eingeführt (Kapitel 2.1.3), Datenstrom-Verarbeitungssysteme (Kapitel 2.1.4) und Systeme für die Verarbeitung komplexer Events (Kapitel 2.1.5) beschrieben und spezielle Eigenschaften solcher Systeme erläutert (Kapitel 2.1.6).

2.1.1 Übersicht

Ein ISV-System hat die Aufgabe, Informationen, die aus unterschiedlichen Quellen in das System kommen, zu verarbeiten und die Ergebnisse, wiederum Informationen, auszugeben. Ein Schlüsselaspekt in solchen Systemen ist, dass die ankommenden Informationen, im Gegensatz zu traditionellen Datenbanksystemen, prinzipiell nicht gespeichert und indexiert werden. Sie strömen durch das System (Cugola and Margara, 2012).

Im Kontext der Informationsstromverarbeitung sind Informationsströme dynamische, ständig wechselnde und unendliche Ströme von Daten (z.B. Tupel) (Cugola and Margara, 2012). Unendliche Datenströme stellen solche Systeme vor grosse Herausforderungen. Wie soll man mit den beschränkten Ressourcen umgehen? Wann werden Zwischenergebnisse nicht mehr benötigt; wann und wie lange sollen sie behalten werden? Da sich die Daten über die Zeit ändern, müssen Resultate des Systems auch fortlaufend geändert werden (Cugola and Margara, 2012). Während in relationalen Datenbanksystemen die Daten als fix mit wechselnden Abfragen gesehen werden, arbeitet ein System für Informationsstromverarbeitung mit fixen Abfragen und wechselnden Daten (Scharrenbach *et al.*, 2013).

Cugola and Margara (2012) unterscheiden zwei verschiedene Arten von ISV-Systemen: Systeme für Datenstromverarbeitung (engl. Data Stream Management Systems, DSMS) und Systeme für die Verarbeitung komplexer Events (engl. Complex Event Processing Systems, CEP-Systeme). CEP-Systeme definieren Regeln, um aus der Kombination einzelner auftretender Muster von Events übergeordnete Informationen zu generieren (Scharrenbach *et al.*, 2013). DSMSs entstammen den traditionellen Datenbanksystemen. Mit Fenster-Operatoren begrenzen sie die ankommenden Datenströme, um diese mithilfe relationaler Algebra zu transformieren (Scharrenbach *et al.*, 2013).

Scharrenbach *et al.* (2013) führen den Begriff Semantic Flow Processing System (SFP-System) ein. Solche Systeme erweitern die beiden Arten von ISV-Systemen, indem die Daten basierend auf dem RDF-Datenmodell mit Semantik versehen werden. Mittels dieser Erweiterung unterstützen diese Systeme Inferenz-Mechanismen (Scharrenbach *et al.*, 2013).

2.1.2 Terminologie

Dieses Kapitel klärt die Verwendung der Begriffe und ihre Bedeutung. *Systeme für Informationsstromverarbeitung* oder *Informationsstrom-Verarbeitungssysteme* (ISV-Systeme) werden in dieser Arbeit als übergeordnete Begriffe für alle Arten solcher Systeme verwendet. Gleichzeitig stellt ein *Informationsstrom* einen generischen Strom dar, welcher dem Input und Output solcher Systeme entspricht. In *Datenstrom-Verarbeitungssystemen* (oder *Systemen für Datenstromverarbeitung*, engl. *Data Stream Management Systems*, *DSMSs*) bestehen die Informationsströme aus Datenströmen (z.B. aus Tupel oder in *SFP-Systemen* aus RDF-Tripel). Der Input von *Systemen für die Verarbeitung komplexer Events* (oder engl. *Complex Event Processing Systems*, *CEP-Systeme*) besteht aus *Events* (oder *Ereignissen*). Zusätzlich werden für Vergleiche auch traditionelle Datenbanksysteme (engl. *Database Management Systems*, *DBMSs*) erwähnt.

2.1.3 Architektur

Eine grobe Architektur eines ISV-Systems ist in Abbildung 2.1 dargestellt. Sowohl DSMSs als auch CEP-Systeme haben gemeinsam, dass Informationsströme (*Information flows*) aus verschiedenen Quellen in das Verarbeitungssystem (*Information Flow Processing Engine*) fliessen. Das System verarbeitet die Informationen basierend auf den Verarbeitungsregeln (*Processing Rules*) welche durch die Benutzer des Systems (*Rule managers*) definiert wurden. Die Regeln bestimmen, wann und wie ankommende Informationsströme verarbeitet werden sollen. Im Anschluss verlassen wiederum Informationsströme das System in sogenannte Waschbecken (*Sinks*), durch welche die produzierten Informationen weiterverwendet werden können.

Wie bereits erwähnt, geschieht die Informationsverarbeitung prinzipiell ohne Speicherung der Daten. Die Ströme fliessen durch das System, wobei die alten Informationen verarbeitet und anschliessend gelöscht und neue Informationen generiert und weitergegeben werden (Cugola and Margara, 2012).

2.1.4 Datenstrom-Verarbeitungssystem

Um die speziellen Anforderungen von ISV-Systemen zu meistern, wurden Datenstrom-Verarbeitungssysteme von der Datenbank-Community entwickelt. Diese besitzen ein ähnliches Ausführungsmodell wie traditionelle Datenbanksysteme (Scharrenbach *et al.*, 2013). DSMSs unterscheiden sich nach Cugola and Margara (2012) in drei wesentlichen Punkten von traditionellen DBMSs:

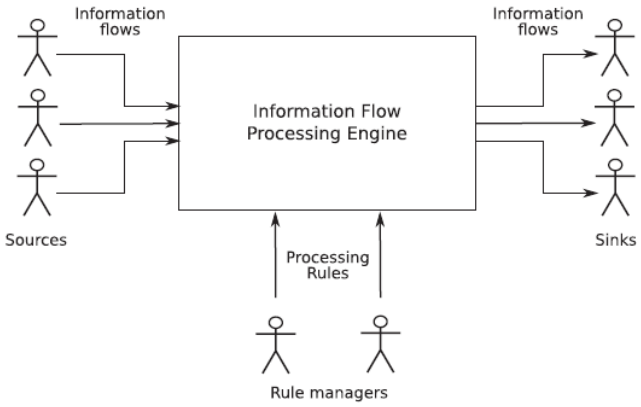


Abbildung 2.1: Architektur eines ISV-Systems. (Quelle: Cugola and Margara (2012))

1. Die Datenströme sind unendlich.
2. Die Daten haben eine unbestimmte Ankunftsreihenfolge.
3. Die Datenmengen und die Zeitbedingungen machen es schwierig, die Daten zu speichern und mehr als einmal zu verarbeiten. Datenströme werden typischerweise einmalig verarbeitet.

Durch die Benutzer von DSMSs werden Abfragen hinzugefügt, welche anschliessend Resultate produzieren bis diese wieder entfernt werden. Dies stellt einen Gegensatz zu traditionellen DBMSs dar, bei welchen die Abfragen variieren und die Daten fix sind. Produzierte Resultate können entweder genau oder, wenn die Abfrage mehr Ressourcen braucht als vorhanden, eine Approximation sein. Eine Abfrage kann fortlaufend eigenständige Resultate liefern oder ein bereits ausgegebenes Resultat ständig verändern (z.B. bei einer Aggregation) (Cugola and Margara, 2012).

Abbildung 2.2 zeigt die typische Architektur eines DSMS. Im Zentrum steht das Abfragesystem (*Q* für Queries) mit den von Benutzern hinzugefügten Abfragen. Der Input besteht aus verschiedenen Datenströmen (*Stream n*) und der Output kann die folgenden vier unterschiedliche Formen annehmen (Cugola and Margara, 2012):

1. Antwort-Datenstrom (*Stream*): Erzeugte Abfrageresultate, welche einmal berechnet wurden und vom System nicht mehr gebraucht werden.
2. Zwischenspeicher für Antwortteile (*Store*): Antwortteile, welche in der Zukunft vielleicht noch verändert oder gelöscht werden.
3. Zwischenspeicher für Daten, welche vielleicht Teil der Antwort werden (*Scratch*): Beispielsweise ein Tupel, welches noch auf einen möglichen Join-Partner wartet.
4. Papierkorb (*Throw*): In den Papierkorb kommen Daten, die nicht mehr verwendet werden.

DSMSs fokussieren sich auf das Antworten von Abfragen, die gemäss den ständig wechselnden Eingangsdaten fortlaufend aktualisiert werden. Eine Limitation besteht jedoch im Erkennen von komplexen Mustern in den Daten, wie Sequenzen oder Ordnungsrelationen (Cugola and Margara, 2012).

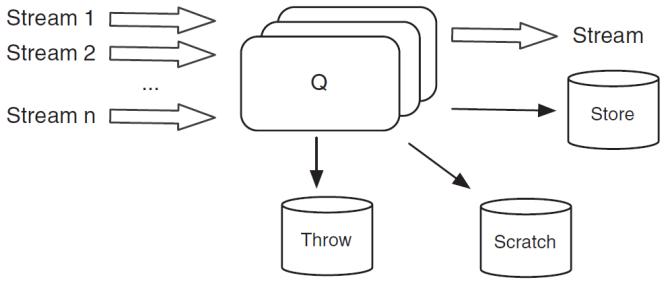


Abbildung 2.2: Typische Architektur eines DSMS. (Quelle: Cugola and Margara (2012))

2.1.5 Verarbeitungssystem für komplexe Events

Im Gegensatz zu DSMSs haben Verarbeitungssysteme für komplexe Events (CEP-Systeme) ihre Kernkompetenz im Erkennen von komplexen Mustern in den Daten. Wie Abbildung 2.3 zeigt, gibt es Beobachter (*Event observers*), welche Ereignisse (*Events*) in der Aussenwelt beobachten. Das System selbst (*Complex Event Processing Engine*) nimmt beobachtete Ereignisse wahr und verarbeitet diese zu übergeordneten Informationen, die an die Ereignis-Konsumenten (*Event consumers*) weitergereicht werden.

Das älteste Verarbeitungssystem für Events ist *Publish/Subscribe* (Cugola and Margara, 2012). In diesem System kann ein Benutzer sein Interesse für ein Themengebiet bekannt geben (*Subscribe*). Wird eine Information zu einem Themengebiet veröffentlicht (*Publish*), so werden dynamisch alle Verfolger (*Subscriber*) dieses Themas über diese Veröffentlichung benachrichtigt. *Publish/Subscribe-Systeme* unterstützen jedoch keine zusammengesetzten Events, bei welchen ein Event von anderen Events abhängt.

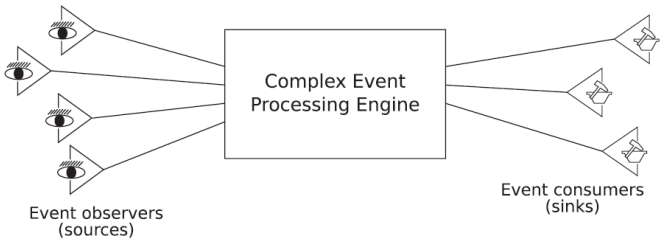


Abbildung 2.3: Typische Architektur eines CEP-Systems. (Quelle: Cugola and Margara (2012))

Die Hauptfähigkeit von CEP-Systemen liegt im Erkennen von komplexen Mustern in hereinkommenden Daten. Dazu gehören Sequenzen von Ereignissen und Reihenfolgen in Ereignisketten.

2.1.6 Eigenschaften

Dieses Kapitel beschäftigt sich mit speziellen Eigenschaften von ISV-Systemen, im Besonderen mit Eigenschaften von SFP-Systemen, welche in Scharrenbach *et al.* (2013) aufgeführt sind. Zuerst wird die Rolle von Hintergrunddaten diskutiert. Anschliessend folgen Erläuterungen über

Inferenz-Unterstützung; es werden Qualitätsanforderungen eingeführt, Zeitmodell und Zeitsemantik beschrieben, Aspekte der Abfragesprache dargelegt und die Verteilung der Systeme besprochen.

Support von Hintergrunddaten Unterstützung für Hintergrunddaten bedeutet, im Verarbeitungsprozess sowohl die Informationsströme als auch gespeicherte Hintergrunddaten berücksichtigen zu können. Ein ISV-System kann Hintergrunddaten entweder berücksichtigen oder ganz weglassen. Die Hintergrunddaten können als fix betrachtet oder Veränderungen können zugelassen werden.

Support von Inferenz Inferenz kann nur von SFP-Systemen unterstützt werden, in denen die Informationsströme aus semantischen Daten bestehen (Scharrenbach *et al.*, 2013).

Qualitätsanforderungen Ein System kann verschiedenen Qualitätsanforderungen genügen. Qualitätsanforderungen können eine bestimmte Performanz oder Antwortzeit, den Anteil aller korrekten Antworten (engl. completeness) oder jenen an inkorrekten Antworten (engl. soundness) betreffen (Scharrenbach *et al.*, 2013).

Zeitmodell Ein Zeitmodell gibt den ankommenden Informationen eine Ordnung (Cugola and Margara, 2012). So kann beispielsweise ermittelt werden, dass $Ereignis1$ vor $Ereignis2$ geschehen ist. Die Zeit kann explizit im Informationsstrom vorhanden sein oder implizit durch einen Zeitstempel des Systems hinzugefügt werden (Scharrenbach *et al.*, 2013). In DSMSs wird die Zeit kaum berücksichtigt, während sie in CEP-Systemen eine grosse Rolle spielt (Cugola and Margara, 2012).

Zeitsemantik Die Zeitsemantik in Informationsströmen kann entweder in einem Zeitpunkt oder einem Zeitintervall dargestellt werden. Zeitpunkte haben keine Dauer (z.B. das Auftreten eines Events oder das Eintreffen eines Tupels im System), während Zeitintervalle über eine Zeitspanne verfügen (Scharrenbach *et al.*, 2013).

Abfragemodell Das Abfragemodell eines ISV-Systems ist von grosser Bedeutung. Es beschreibt, wie mit der speziellen Eigenschaft von unendlichen Informationsströmen umgegangen wird. Die Ausführung von Abfragen kann entweder reaktiv, immer dann, wenn neue Informationen ankommen, oder periodisch sein (Scharrenbach *et al.*, 2013).

Die Abfragesprachen von ISV-Systemen teilen Cugola and Margara (2012) in zwei Kategorien ein: Transformierende Sprachen (engl. transforming langugage) und entdeckende bzw. musterbasierte Sprachen (engl. detecting or pattern-based languages).

Transformierende Sprachen definieren Regeln und Operatoren, mit welchen die ankommenden Informationsströme verarbeitet werden, ähnlich wie in DBMSs. Zu den Operatoren gehören Filter, Joins, Aggregationen, Vereinigungen, Projektionen und Ähnliches. Transformierende Sprachen werden normalerweise in DSMSs verwendet (Cugola and Margara, 2012) und orientieren sich an SQL.

Musterbasierte Sprachen führen Regeln zur Entdeckung gewisser Zustände in der Beobachtungsumwelt ein. Wird ein solcher Zustand entdeckt und sind die definierten Konditionen erfüllt, wird die Verarbeitung dieser Informationen gestartet. Die Konditionen bestehen aus logischen Operationen, Inhalts- und Zeitbeschränkungen. Musterbasierte Abfragesprachen werden vermehrt in CEP-Systemen verwendet. Diese Systeme suchen nach einem Auftreten relevanter Events und verarbeiten solche dann (Cugola and Margara, 2012).

Es kommt vor, dass die beiden Kategorien der Abfragesprachen vermischt sind. So können SQL-ähnliche Abfragen mit Konstrukten aus musterbasierten Sprachen erweitert werden. Die Event Processing Language (EPL) von Esper [1], welche für die Evaluation in Kapitel 3 verwendet wurde, ist ein Beispiel für eine vermischte Abfragesprache.

Den Abfragesprachen werden Konstrukte hinzugefügt, um eingehende Ströme in Teilen verarbeiten zu können. Diese Beschränkungen werden Fenster (engl. windows) genannt (Cugola and Margara, 2012). Fenster definieren, welchen Teil des Inputs für einen Operator zu verwenden sind (Cugola and Margara, 2012). Vielfach werden Fenster auch als semantische Beschränkungen verwendet. Um in einem Sensorsystem von Wetterstationen die durchschnittliche Temperatur eines Tages zu ermitteln, bedarf es eines Zeitfensters von einem Tag, während dem die Temperaturdaten gesammelt werden und daraus am Schluss die Durchschnittstemperatur berechnet wird.

Cugola and Margara (2012) fassen verschiedene Arten von Fenstern zusammen. Drei davon sind in der Tabelle 2.1 beschrieben.

Fensterbezeichnung englisch	Beschreibung	Beispiele
Fixed Window	Fenster mit fixer Ober- und Untergrenze	2013-07-01-00:00 bis 2013-07-02-00:00
Landmark Window	Fenster mit fixer Untergrenze	Alle Tupel seit 2013-07-01-00:00
Sliding Window	Fenster mit fixer Grösse, die Grenzen bewegen sich, wenn neue Daten ankommen	eine Stunde, ein Monat, 100 Tupel, 10 Events

Tabelle 2.1: Verschiedene Fensterarten.

Zentralisierte oder verteilte Systeme Verteilte Systeme werden wichtiger, sobald die Datenmengen sehr gross sind oder weite geographische Abstände zwischen Informationsquellen bestehen. Einerseits kann ein verteiltes System mehr Abfragen gleichzeitig verarbeiten. Andererseits kann die Ausführung einer einzelnen Abfrage über mehrere Maschinen verteilt werden, um die Antwortzeit zu verkürzen. Interessant ist auch der Ansatz, Filter-Operationen möglichst nahe bei den Informationsquellen durchzuführen, um die Netzlast des Systems zu verringern (Scharrenbach *et al.*, 2013).

2.2 Arten von Joins

Dieses Kapitel bietet einen Überblick über verschiedene Join-Varianten. Es führt von einfachen Joins (Kapitel 2.2.2) zu spezifischeren temporalen (Kapitel 2.2.3) und sequentiellen Joins (Kapitel 2.2.4).

2.2.1 Join-Operationen und Joins

Diese Arbeit unterscheidet folgendermassen zwischen Join-Operationen und Joins: Ein Join ist die abstrakte Methode, zwei Relationen zu kombinieren. Join-Operationen hingegen bezeichnen die effektiv ausgeführten Operationen eines spezifischen Joins.

[1] esper.codehaus.org/

2.2.2 Einfache Joins

Unter einem einfachen Join ist die Kombination der Tupel zweier Relationen oder Datenströme zu verstehen, welche eine gewisse Join-Kondition erfüllen (Liu and Özsu, 2009). Die Tabelle 2.2 gibt einen kurzen Überblick über verschiedene einfache Joins in Anlehnung an Liu and Özsu (2009).

Join-Name	Beschreibung
Equi Join	Ein Equi Join kombiniert die Tupel zweier Relationen, wenn die Join-Kondition der Gleichheit von zwei Attributwerten erfüllt ist.
Natural Join	Der Natural Join ähnelt dem Equi Join. Im Gegensatz zu jenem ist jedoch die Join-Kondition nicht anzugeben. Der Natural Join verbindet zwei Relationen, wenn die Attribute denselben Namen und denselben Wert aufweisen.
Inner Join	Ein Inner Join verbindet zwei Tupel, wenn die Join-Kondition erfüllt ist. Im Gegensatz zum Equi und zum Natural Join muss die Kondition nicht Gleichheit sein. Ein beliebiger Vergleichsoperator kommt in Frage.
Left/Right Join	Das Resultat eines Left Joins zwischen zwei Relationen enthält alle Tupel, die die Join-Konditionen erfüllen, und zusätzlich die Tupel der linken Join-Relation, aufgefüllt mit Null-Werten für die Attribute der rechten Join-Relation. Der Right Join funktioniert analog zum Left Join.
Full Outer Join	Ein Full Outer Join zwischen zwei Relationen beinhaltet alle Tupel, welche die Join-Kondition erfüllen, und zusätzlich alle Tupel beider Relationen, aufgefüllt mit Null-Werten.

Tabelle 2.2: Verschiedene einfache Joins.

2.2.3 Temporale Joins

Die Join-Konditionen bei temporalen Joins beziehen sich auf Attribute der Join-Relationen, die einen zeitlichen Aspekt aufweisen. Dies kann beispielsweise ein einzelner Zeitpunkt oder ein Zeitintervall darstellen. Es existieren verschiedene Arten von temporalen Konditionen. Allen (1983) veröffentlichte 13 mögliche Beziehungen zwischen zwei Intervallen. Ein Tupel $T1$ der Relation $R1$ mit Zeitstempeln $t1$ und $t2$ kann strikt vor oder nach einem Tupel $T2$ der Relation $R2$ mit Zeitstempeln $t1'$ und $t2'$ gültig sein (*before*, $t2 < t1'$). Sie können genau gleichzeitig starten und enden (*equal*, $t1 = t1'$ und $t2 = t2'$), $T2$ kann genau anschliessend an das Ende von $T1$ starten (*meets*), sie können überlappend sein (*overlaps*) oder $T1$ kann während $T2$ stattfinden (*during*). Weiter kann der Startzeitpunkt oder der Endzeitpunkt von beiden Tupeln derselbe sein (*starts/finishes*). Abbildung 2.4 zeigt eine Übersicht über die 13 möglichen Beziehungen in der Intervallalgebra. Durch die Möglichkeit, sechs Beziehungen zwischen zwei Intervallen umzukehren (z.B. x vor y und y nach x) entstehen die 13 unterschiedlichen Beziehungen (die Gleichheit kann nicht invertiert werden).

Um temporale Joins auszuführen, müssen einerseits die Informationsströme mit Zeitsemantik versehen sein (siehe Zeitsemantik in Kapitel 2.3), andererseits muss das Abfragemodell (siehe Abfragemodell in Kapitel 2.3) Konstrukte unterstützen, welche mit der Zeitsemantik umgehen können. Daten in traditionellen Datenbanksystemen haben beispielsweise keine Reihenfolge. Das zeitlich nächste Tupel kann nur über aufwendige Konstrukte wie "WHERE NOT EXISTS" bestimmt werden.

Relation	Symbol	Symbol for Inverse	Pictoral Example
X *before* Y	<	>	XXX YYY
X *equal* Y	=	=	XXX YYY
X *meets* Y	m	mi	XXXYYY
X *overlaps* Y	o	oi	XXX YYY
X *during* Y	d	di	XXX YYYYYY
X starts Y	s	si	XXX YYYYY
X finishes Y	f	fi	XXX YYYYY

Abbildung 2.4: Allans 13 mögliche Beziehungen zwischen Intervallen. (Quelle: Allen (1983))

2.2.4 Sequentielle Joins

Ein sequentieller Join besteht aus einer geordneten Reihe von aufeinanderfolgenden Joins, wobei der erste Join strikt vor dem zweiten Join und der zweite vor dem dritten etc. ausgeführt wird. Die einzelnen Joins können dabei unterschiedliche Join-Konditionen haben. Abbildung 2.5 zeigt beispielhaft, wie ein dreifacher sequentieller Join aussehen würde. $Join1$ wird vor $Join2$ ausgeführt, wobei die Resultate von $Join1$ ein Input für $Join2$ darstellen usw. Mithilfe von Allans 13 möglichen Beziehungen zwischen Intervallen können in ISV-Systemen komplexe, zeitabhängige Informationen gewonnen werden. Die Zeitinformation kann den ankommenden Tupeln entweder durch das System hinzugefügt werden (Systemzeit), oder das Tupel selbst kann über eine Zeitinformation verfügen, welche dem Tupel eine semantische Zeitinformation gibt (Datenzeit). Diese Arbeit betrachtet Allens Intervallalgebra als Methode temporale und sequentielle Joins in ISV-Systemen zu erstellen. Wie Scharrenbach *et al.* (2013) bemerkt haben, setzen sequentielle und temporale Joins in den Informationsströmen sowohl eine Ordnung, als auch eine Abfragesprache, welche die Intervallalgebra von Allen berücksichtigt, voraus. Die Eigenschaften und Herausforderungen von sequentiellen und von temporalen Joins werden in den nächsten beiden Unterkapiteln (2.3 und 2.4) genauer betrachtet.

2.3 Eigenschaften von Joins in ISV-Systemen

Da die Informationsströme spezielle Eigenschaften aufweisen, werden in diesem Kapitel die Eigenschaften von Joins im Kontext der Informationsstromverarbeitung beschrieben. Dies erfolgt in Anlehnung an Scharrenbach *et al.* (2013). Die vorliegende Arbeit unterscheidet zuerst, ob es sich um einen Join zwischen Informationsströmen untereinander, oder zwischen Informationsströmen und gespeicherten Hintergrunddaten handelt. Sind Hintergrunddaten involviert, stellt sich die Frage, ob diese im Verlauf der Ausführung der Abfrage konstant oder dynamisch sind (siehe Abschnitt Hintergrunddaten in Kapitel 2.3). Dynamische Hintergrunddaten müssen sorgfältig behandelt werden, da bereits ausgegebene Resultate auf einmal nicht mehr korrekt sein könnten. Ein Join kann aus einem Selbstjoin mit nur einem Informationsstrom oder aus einem

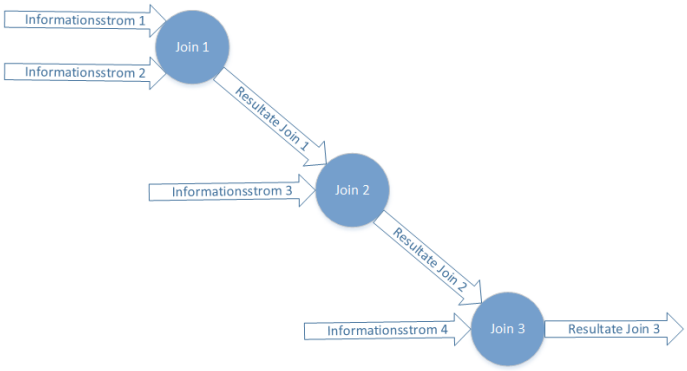


Abbildung 2.5: Dreifacher sequentieller Join.

Join mit mehreren Informationsströmen bestehen.

Weiter kann ein ISV-System auf einer einzelnen Maschine oder auf mehreren Maschinen in Betrieb sein (siehe Abschnitt Zentralisierte oder verteilte Systeme in Kapitel 2.3). Mehrere Maschinen stellen weitere Herausforderungen an das System. Falls die Datenströme für einen Join auf unterschiedlichen Maschinen ankommen, muss das System dies koordinieren.

Andere Eigenschaften beziehen sich auf die verfügbaren Ressourcen. Haben die Daten für die Auswertung einer Abfrage im Hauptspeicher Platz, oder ist eine Zwischenspeicherung auf der Festplatte nötig? Ist die Datenrate der Datenströme konstant oder schwankt sie?

Ein Join kann immer ausgeführt werden, wenn ein Tupel ankommt, oder in bestimmten Zeitintervallen.

2.4 Herausforderungen

Basierend auf den Erkenntnissen von Join-Eigenschaften in ISV-Systemen (Kapitel 2.3), werden in diesem Kapitel die Herausforderungen von Joins diskutiert.

2.4.1 Herausforderungen bei einem Join mit schwankenden Datenraten der Informationsströme

Eine schwankende Datenrate stellt ein ISV-System vor verschiedene Herausforderungen. Die folgende Aufzählung bietet einen Überblick über die Herausforderungen, wenn ein Informationsstrom eine schwankende Datenrate aufweist.

- Korrektheit und Vollständigkeit von Antworten (Qualitätsanforderungen, Kapitel 2.1.6): Kann das System die Anfragen trotz schwankender Datenrate korrekt und vollständig beantworten?
- Lastverteilung: Wie gut verteilt das System die Berechnungen auf die verfügbaren Ressourcen?
- Antwortzeit: Wie stark leidet die Antwortzeit, wenn die Datenrate schwankt? Wie verhält sich die gesamte Ausführungszeit des Joins bei konstanter Datenrate im Vergleich zu einer schwankenden Datenrate?

- Durchsatz: Wie gross ist der Durchsatz bei konstanter/schwankender Datenrate?
- Wie sieht das optimale Zeitfenster aus, wenn es nicht durch die Anfrage vorgegeben ist? Ist es zu gross, können die verfügbaren Ressourcen aufgebraucht werden oder die Antwortzeit kann darunter leiden. Bei einem zu kleinen Zeitfenster hingegen, werden vielleicht nur wenige Resultate ausgegeben.

2.4.2 Herausforderungen bei sequentiellen Joins

Ein ISV-System muss für die Ausführung von sequentiellen Joins verschiedene Herausforderungen meistern. Diese Arbeit unterteilt die Herausforderungen in drei verschiedene Kategorien: Erstens, die Anzahl der aufeinanderfolgenden Joins; zweitens, der Rechenaufwand jedes einzelnen Joins und schliesslich die Art der Datenrate der Informationsströme.

Anzahl der aufeinanderfolgenden Joins

Die erste Kategorie beschäftigt sich mit der Anzahl an Joins, die ein Informationsstrom-Verarbeitungssystem in einer Abfrage durchführen muss. Je mehr Join-Operationen ein System pro Zeiteinheit durchführen muss, desto mehr Ressourcen (CPU, Arbeitsspeicher) werden für die Ausführung einer Abfrage benötigt. Ziel ist es, die Qualitätsanforderungen an das System zu erfüllen. Dies kann beispielsweise durch eine gleichmässige Lastenverteilung über die vorhandenen Prozessoren bzw. im Falle eines verteilten Systems über die vorhandenen Maschinen zu erreichen sein oder indem Flaschenhälse (engl. Bottlenecks) identifiziert und vermieden werden. Da ein ISV-System möglicherweise keine Informationen über die ankommenden Informationen hat (wie Datenrate oder Grösse des Resultats), kann eine gleichmässige Lastenverteilung eine grosse Herausforderung darstellen.

Aufwand für einen Join

Der Rechenaufwand eines Joins kann verschieden sein und somit unterschiedlich viele Ressourcen beanspruchen. Einerseits kann ein Join, wenn die Join-Partner generell sehr weit auseinander liegen, sehr viel Zwischenspeicher beanspruchen und somit den Arbeitsspeicher stark belasten. Problematisch ist es, wenn die Datenrate prinzipiell hoch ist, da dies zu einem Überlauf des Arbeitsspeichers führen kann. Andererseits können sehr viele Join-Partner eines Informationsstroms innerhalb kurzer Zeit eintreffen, was zu vielen Join-Operationen führt und damit die Rechenleistung eines Systems stark beansprucht. Des Weiteren kann ein einzelner Join nicht nur aus zwei Join-Partnern bestehen, sondern mehrere Informationsströme verwenden, was eine zusätzliche Herausforderung darstellt. Auch mehrere Join-Konditionen sind möglich, welche bei der Ausführung mehr CPU-Zeit beanspruchen. Zusätzlich spielt die Grösse der Zeitfenster, in denen ein Join zwischen zwei Informationsströmen gültig ist, eine Rolle. Ist das Zeitfenster gross, müssen mehr Zwischenergebnisse gespeichert werden.

Datenrate der Informationsströme

Die Datenrate stellt bei sequentiellen Joins ein Problem dar. Ist zum Beispiel die Datenrate der beiden Informationsströme beim ersten Join einer Folge von sequentiellen Joins sehr hoch, kann dies zu einem Flaschenhals führen und die Antwortzeit der gesamten Abfrage stark verlangsamen. Im Extremfall kann das System überlastet werden. Es muss dann entweder hereinkommende Tupel blockieren oder abweisen, zwischengespeicherte Tupel freigeben oder die Abfrage abbrechen. Hier entsteht ein Zielkonflikt zwischen Antwortzeit und Vollständigkeit der Antwort. Manchmal

kann es sinnvoll sein, zugunsten einer schnelleren Antwortzeit auf die Vollständigkeit zu verzichten Scharrenbach *et al.* (2013).

2.5 Key Performance Indicators

Dieses Unterkapitel führt drei Key Performance Indicators (KPIs) ein, welche sich an Scharrenbach *et al.* (2013) orientieren. Diese Leistungskennzahlen dienen als Massstab für eine Evaluation der Systemleistung eines ISV-Systems. Einen Benchmark für ein System zu entwickeln bedeutet, den verfügbaren oder erstellten Datensatz zu analysieren und, basierend auf Erkenntnissen aus diesen Daten sowie Änderungen an Testparametern, die beste, die mittlere und die schlechteste Leistung des Systems anhand der definierten KPIs zu messen (Scharrenbach *et al.*, 2013). Im Folgenden werden die drei KPIs erläutert:

- Antwortzeit (engl. response time) über alle Abfragen (Mittelwert, Perzentile, Maximum).
- Maximaler Durchsatz: Anzahl Tupel, die das System in einer gewissen Zeit verarbeiten kann (engl. maximum input throughput).
- Minimale Zeit, die korrekten Resultate zu berechnen und minimale Ausführungszeit aller Abfragen. Dies schliesst Präzision (Anzahl ausgegebener, korrekter Resultate / Anzahl ausgegebener Resultate), Trefferquote (Anzahl ausgegebener korrekter Resultate / Anzahl aller korrekten Resultate) und die Fehlerquote in Relation zu der Ausführungszeit ein (engl. minimum time to accuracy and the minimum time to completion).

Verschiedene Stresstests können anhand dieser drei KPIs an verschiedenen Systemen implementiert und ausgewertet werden. Dadurch werden die Reaktionen auf die Stresstests vergleichbar gemacht.

Kapitel 3

Evaluation

Dieses Kapitel beschreibt, wie sich Streams-Esper, ein System für die Verarbeitung komplexer Events, gegeben eines Datensatzes von IPTV-Benutzerlogs, bei einem fünffachen sequentiellen Join verhält. Die Evaluation testet die aufgestellten Hypothesen aus Kapitel 1.2. Es soll überprüft werden, ob diese Daten für einen Stresstest eines CEP-Systems geeignet sind. Ferner setzt sich dieses Experiment mit dem Zusammenhang zwischen Output und unterschiedlich grossen Zeitfenstern in der Abfrage auseinander. Schwankende Datenraten werden im Datensatz identifiziert und deren Auswirkungen auf das Experiment dargelegt. Die Frage ist, wann und wieso ein spezifischer Join länger dauert als andere.

Das Kapitel ist folgendermassen aufgebaut: Zu Beginn werden in Kapitel 3.1 die verwendeten Technologien und Infrastrukturen erläutert. Anschliessend wird das ViSTA-TV-Projekt, welchem die Daten entstammen, vorgestellt. Kapitel 3.3.2 beschreibt, wie die Daten für das Experiment aufbereitet worden sind sowie einige interessante Statistiken der Daten. Im Anschluss wird das Experiment eines fünffachen sequentiellen Joins beschrieben (Kapitel 3.3.3) und die Resultate (Kapitel 3.3.3) präsentiert.

3.1 Verwendete Technologien und Infrastruktur

Dieses Unterkapitel gibt einen kurzen Überblick über die verwendeten Technologien und die Infrastruktur.

3.1.1 Streams-Esper

Esper[1] ist ein quelloffenes Event Stream Processing System (ESP-System) und CEP-System geschrieben in Java. Zusammen mit der unterstützten Event Processing Language (EPL) ist Esper hoch skalierbar, speichereffizient und erlaubt nahezu Echtzeit-Abfragen in SQL-ähnlichem Muster. Die EPL von Esper ist mit sehr ausgefeilten, auf Eventströme ausgerichteten Funktionalitäten ausgestattet.

Streams[2] ist ein in Java geschriebenes Framework für eine einfache Verarbeitung von Informationsströmen. Die Idee hinter der Streams-Bibliothek ist, eine Ausführungsumgebung bereitzustellen, sodass die Anwender die Datenstromprozesse in XML-Dateien implementieren können.

Streams-Esper kombiniert die beiden Frameworks Esper und Streams. Streams agiert sozusagen als Schicht zwischen den XML-Ausführungsdateien und dem CEP-System von Esper. Durch

[1] esper.codehaus.org/
[2] http://www-ai.cs.uni-dortmund.de/SOFTWARE/streams/

diese Verbindung entsteht eine System, das von der sehr ausdrucksstarken und mächtigen EPL von Esper profitiert und mittels XML-Dateien unkomplizierte Ausführungen erlaubt.

3.1.2 PostgreSQL

Für die qualitativen Auswertungen und die Verifikation des Outputs von Streams-Esper wurde die bekannte quelloffene, relationale Datenbank PostgreSQL[3]unter der Version 9.2 verwendet.

3.1.3 Kraken

Kraken ist ein Cluster der Universität Zürich[4]. Es besteht aus zwölf Maschinen (Debian 64 bit) mit je 24 Kernen und 64 GB Hauptspeicher. Jobs werden mittels Torque-Ressourcenplanungssystem[5] koordiniert.

3.2 Output eines Tests mit Streams-Esper für spätere Evaluationen

Dieses Unterkapitel erläutert kurz, wie und wann ein Test in Streams-Esper etwas ausgibt, um später die KPIs (Kapitel 2.5) zu berechnen. Streams-Esper schreibt grundsätzlich in drei Fällen einen Log in den Standard-Output:

- Beim Starten des Tests
- Wenn eine Abfrage ein Resultat in den Zwischenspeicher schreibt
- Beim Beenden des Tests

Bei jedem Output wird die Systemzeit mittels $System.currentTimeMillis()$ zusätzlich ausgegeben. Speichert eine Abfrage ein Resultat im Zwischenspeicher, so werden alle Attribute des Resultats zusammen mit der Systemzeit ausgegeben.

Die Antwortzeit berechnet sich anschliessend aus der Systemzeit beim Start und beim Ende des Tests (gesamte Laufzeit des Tests) respektive aus der Systemzeit des ersten und letzten ausgegebenen Resultats (Abfragezeit). Der Durchsatz kann ermittelt werden, indem die Anzahl Tupel der im Test verwendeten Eventströme durch die Antwortzeit dividiert wird. Für den letzten KPI - minimale Zeit für die korrekten Resultate und minimale Ausführungszeit in Relation zur Präzision, Treffer- und Fehlerquote - wird die statische Ausführung des Tests in PostgreSQL für die Verifikation der Systemresultate verwendet.

3.3 Experimente mit ViSTA-TV-Daten

Dieses Unterkapitel beschreibt das Experiment eines fünffachen sequentiellen Joins mit ViSTA-TV-Daten ausgeführt in Streams-Esper. Zuerst werden das ViSTA-TV-Projekt sowie die Datenaufbereitung und die Auswertung der Datenstatistiken charakterisiert. Im Anschluss folgen Beschreibungen zum Aufbau des Experiments im Streams-Esper-Framework und die produzierten

[3]postgresql.org/
[4]https://wiki.ifi.uzh.ch/kraken/
[5]adaptivecomputing.com/products/open-source/torque/

Resultate. Grundsätzlich wird im weiteren Verlauf der Arbeit zwischen zwei Zeiten unterschieden: *Datenzeit* und *Systemzeit*. Während die Datenzeit Auskunft darüber gibt, zu welchem Zeitpunkt ein spezifisches Event stattgefunden hat (z.B. 2012-08-01 11:25 hat Benutzer x eine Sendung auf Pro7 geschaut), informiert die Systemzeit, wann ein Tupel vom System verarbeitet wurde.

3.3.1 Was ist ViSTA-TV?

ViSTA-TV[6] ist ein Forschungsprojekt der Universität Zürich, der technischen Universität Dortmund, der Rapid-I GmbH, der Zattoo Europa AG, der Vrije Universität Amsterdam und der BBC. Es verfolgt das Ziel, basierend auf den Benutzerlogs von IPTV-Applikationen (z.B. Zattoo), das Verhalten der Benutzer in Echtzeit zu analysieren. Zusammen mit den Informationen zum aktuellen TV-Programm und zum gespeicherten Benutzerverhalten sollen den Benutzern Vorschläge für weitere TV-Sendungen gemacht werden. Ausserdem können auf diese Weise interessante Marktforschungsinformationen über das Verhalten der Benutzer gewonnen werden.

3.3.2 Datenaufbereitung und Statistiken

Datenaufbereitung

Die Daten sind dem ViSTA-TV-Projekt entnommen. Für dieses Experiment wurden die Daten vom 01.08.2012 verwendet, welche insgesamt 138'000 Tupel ergeben. Der Grossteil der Tupel enthält dabei den Startzeitpunkt der Sendung (tStart), den Endzeitpunkt (tEnd), den Benutzeridentifikator (uID) und eine Bezeichnung der Sendung selbst (pID). Ein kleiner Teil von 10'000 Tupel beinhaltet TV-Programminformationen mit den Attributen Startzeitpunkt, Endzeitpunkt, Programmidentifikator und Titel der Sendung.

Ein Tupel-Beispiel eines Benutzerlogs sieht wie folgt aus: <tStart, tEnd, uID, pID>

Zu erwähnen ist, dass der Startzeitpunkt dem Endzeitpunkt gleichgesetzt wurde, da die einzelnen Events keine Dauer haben und es in diesem Experiment um aufeinanderfolgende Events und nicht um deren jeweiligen Dauer geht.

Damit Streams-Esper die Daten verwenden kann, wurden der Startzeitpunkt und der Endzeitpunkt mittels eines Python-Skripts in Millisekunden seit der Epoche umgewandelt. Anschliessend wurden die Daten nach dem Startzeitpunkt der Events aufsteigend sortiert, damit daraus ein geordneter Eventstrom gemacht werden kann. Die Tupel wurden dann in eine CSV-Datei geschrieben, sodass diese als CSV-Eventstrom im Streams-Esper-Framework verwendet werden können.

Statistiken der Daten

Um die Resultate des Experiments auszuwerten und den Output des Systems auf Korrektheit zu verifizieren, wurde das Experiment statisch in PostgreSQL nachproduziert.

[6] vista-tv.eu/

Eine Beispielabfrage des dritten Joins in PostgreSQL sieht folgendermassen aus:

```
SELECT
 W1.uID, W1.startTime1, W1.startTime2, W1.startTime3,
 W2.startTime AS startTime4
FROM
 resultsJoin2 as W1, 20120801vistatv as W2
WHERE
 W1.uID = W2.uID AND
 W1.startTime3 < W2.startTime
AND NOT EXISTS (
 SELECT * FROM watched AS W3
 WHERE W3.uID = W1.uID AND W3.startTime > W1.startTime3
 AND W3.startTime < W2.startTime) ;
```

Listing 3.1: SQL-Abfrage in PostgreSQL des dritten sequentiellen Joins.

Die Abfrage sucht nach dem zeitlich nächsten Tupel des jeweiligen Benutzers in den Daten. Sie basiert auf einem Join zwischen den Resultaten der vorherigen Joins (Relation resultsJoin2 mit Attributen Benutzeridentifikation, Startzeiten 1-3) und den ViSTA-TV-Daten (Relation 20120801vistatv mit Attributen Benutzeridentifikation, Startzeit, Endzeit und Sendung).

Die Abfrage wurde in Teilabfragen zerlegt, damit einerseits die Resultate für jeden Join vorhanden sind und andererseits die Ausführungszeiten der Teilabfragen in Grenzen gehalten werden kann. Die statischen Auswertungen wurden lokal auf einem Rechner mit zwei Kernen und 4 GB Hauptspeicher ausgeführt.

Im Folgenden werden verschiedene Statistiken der Daten präsentiert.

Anzahl Resultate pro Join, wobei die Ursprungsrelation aus 138'000 Tupeln besteht:

	Anzahl Resultate
nach dem ersten Join	80'000
nach dem zweiten Join	55'000
nach dem dritten Join	41'000
nach dem vierten Join	32'000
nach dem fünften Join	26'000

Tabelle 3.1: Anzahl Resultate pro Join.

Die Anzahl der Resultate nimmt nach jedem Join ab. Anzumerken ist, dass ein Benutzer mit mehr als sechs Events auch mehr Resultate nach dem fünften Join liefert. Ein Beispiel: Hat ein Benutzer acht aufeinanderfolgende Events im Datensatz, resultieren daraus drei Resultat-Tupel (Startzeiten 1-6, 2-7, 3-8).

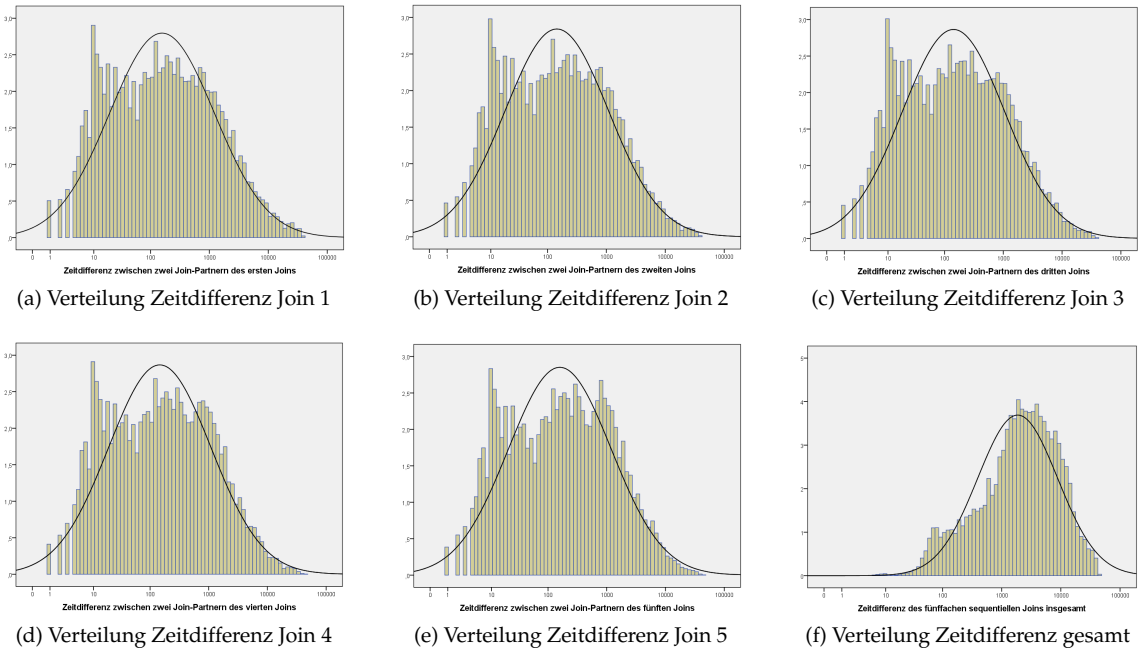
Die deskriptiven Statistiken über die Zeitdifferenzen in Datenzeit zwischen zwei Join-Partnern sind wichtig, um überhaupt die Grösse der Zeitfenster in den Abfragen des CEP-Systems bestimmen zu können. Diese werden in der Tabelle 3.2 für alle Resultate des fünften Joins (26'000 Tupel) zusammengefasst. Interessant sind die grossen Unterschiede zwischen Median und Mittelwert von ca. 770 Sekunden über alle fünf Joins hinweg. Dies deutet einerseits auf einige sehr grosse Ausreisser, andererseits auf eine klar linksschiefe Verteilung hin. In Abbildung 3.1 sind die verschiedenen Verteilungen der Zeitdifferenzen von zwei Join-Partnern dargestellt. Zu beachten gilt es, dass die x-Achsen die Zeitdifferenzen zwischen zwei aufeinanderfolgenden Events logarithmisch abbildet, weshalb die Schiefe nicht sofort ersichtlich ist.

		Zeitdifferenz zwischen zwei Join-Partnern des ersten Joins	Zeitdifferenz zwischen zwei Join-Partnern des zweiten Joins	Zeitdifferenz zwischen zwei Join-Partnern des dritten Joins	Zeitdifferenz zwischen zwei Join-Partnern des vierten Joins	Zeitdifferenz zwischen zwei Join-Partnern des fünften Joins	Zeitdifferenz des fünffachen sequentiellen Joins insgesamt
N		25'936	25'936	25'936	25'936	25'936	25'936
Mittelwert		1'049	908	867	867	912	4'603
Median		153	141	140	148	167	2'240
Standardabweichung		2'974	2'559	2'405	2'366	2'382	6'207
Minimum		1	1	1	1	1	6
Maximum		39'840	39'840	39'840	44'209	44'209	45'217
Perzentile	60	283	260	257	272	304	3'289
	70	542	494	483	514	579	4'815
	80	1'055	949	933	961	1'042	7'376
	90	2'440	2'131	2'019	2'025	2'195	11'988
	99	15'508	12'710	11'846	11'626	11'499	31'767

Zeitdifferenzen in Sekunden

Tabelle 3.2: Deskriptive Statistiken der Zeitdifferenzen zwischen zwei Join-Partnern in Datenzeit für die Joins 1-5 und insgesamt.

Bemerkenswert sind auch die hohen Spannweiten in den Zeitdifferenzen. Diese reichen beim Join vier und fünf von minimal einer Sekunde zu maximal 12,3 Stunden. Hätte das Experiment die Anforderung auf Vollständigkeit, müssten die einzelnen Zeitfenster der Joins gemäss den Maximaldifferenzen in der Abbildung 3.2 gewählt werden.



(a) Verteilung Zeitdifferenz Join 1 (b) Verteilung Zeitdifferenz Join 2 (c) Verteilung Zeitdifferenz Join 3

(d) Verteilung Zeitdifferenz Join 4 (e) Verteilung Zeitdifferenz Join 5 (f) Verteilung Zeitdifferenz gesamt

Abbildung 3.1: Verteilungen der Zeitdifferenzen zwischen zwei Join-Partnern in Datenzeit für alle Resultate, die der fünfte Join liefert. x-Achse: Zeitdifferenz zwischen zwei Join-Partner in Sekunden, y-Achse: Prozent.

In einem nächsten Schritt wurden Statistiken über die Anzahl hereinkommender Tupel pro Zeiteinheit in Datenzeit für einen Join erstellt. Dies ist aufschlussreich, da eine hohe Anzahl Tupel pro Zeiteinheit eine hohe Datenrate darstellt und ein ISV-System vor verschiedene Herausforderungen stellen kann (siehe Kapitel 2.4).

Dazu wurden die ViSTA-TV-Daten zuerst nach den verschiedenen Startzeiten gruppiert und pro Startzeit die Anzahl vorhandener Events gezählt. Mithilfe dieser Tabelle wurde im Folgenden die Anzahl hereinkommender Tupel zwischen zwei Join-Partnern bestimmt. Zusammen mit den Zeitdifferenzen zwischen zwei Events ergibt sich daraus die Anzahl hereinkommender Tupel pro Sekunde.

Eine Beispielabfrage für die Anzahl zu verarbeitender Tupel für den ersten Join in PostgreSQL sieht folgendermassen aus:

```
SELECT
j1.uID, j1.startTime1, j1.startTime2, sum(t1.count) as nrOfTuples
FROM
results5WayJoin as j1, countTuplesPerTimestamps AS t1
WHERE
t1.startTime BETWEEN j1.startTime1 AND j1.startTime2
GROUP BY
j1.uID, j1.startTime1, j1.startTime2;
```

Listing 3.2: SQL-Abfrage für die Anzahl zu verarbeitender Tupel für Join 1

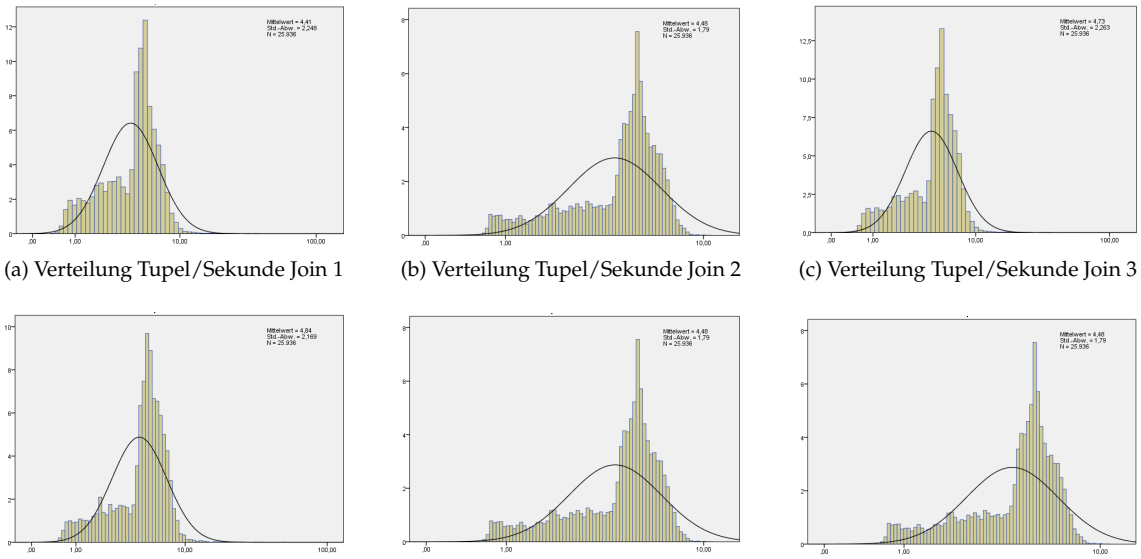
Die Relation $results5WayJoin$ enthält dabei alle Resultate, welche der fünfte Join liefert (also 26'000 Tupel). In der Relation $countTuplesPerTimestamps$ ist hingegen für jeden Zeitstempel gespeichert, wie viele Tupel dazu in den ViSTA-TV-Daten vorhanden sind. Wird die Anzahl hereinkommender Tupel zwischen zwei Join-Partnern, $nrOfTuples$, durch die Differenz zwischen startTime2 und startTime1 dividiert, resultiert daraus die Anzahl hereinkommender Tupel pro Zeiteinheit.

Tabelle 3.3 stellt verschiedene deskriptive Statistiken für die Anzahl zu verarbeitender Tupel pro Sekunde dar. Mittelwert und Median liegen nah beieinander, wobei der Median jeweils um ca. 0,3 Tupel/Sekunde höher ist. Wie bereits erwähnt sind die hohen Ausreisser interessant, da diese hohe Datenraten für das System darstellen. Ungefähr ein Prozent der Werte sind Ausreisser und mit einem Maximalwert von 101 Tupel/Sekunde teilweise auch beachtlich hoch im Vergleich zum Mittelwert bzw. Median.

Ankommende Tupel/Sekunde zwischen zwei Join-Partnern in Datenzeit		Join 1	Join 2	Join 3	Join 4	Join 5	über den fünffachen sequentiellen Join gesamt
N		25'936	25'936	25'936	25'936	25'936	25'936
Mittelwert		4,4	4,6	4,7	4,8	4,9	4,5
Median		4,7	4,8	5,0	5,1	5,2	4,9
Standardabweichung		2,2	2,3	2,3	2,2	2,1	1,8
Minimum		0,4	0,4	0,4	0,4	0,3	0,6
Maximum		101,0	101,0	101,0	51,3	51,3	10,6
Perzentile	60	5,0	5,2	5,3	5,4	5,5	5,2
	70	5,4	5,5	5,7	5,9	6,0	5,5
	80	6,0	6,2	6,3	6,4	6,5	5,9
	90	7,0	7,1	7,2	7,2	7,2	6,6
	99	9,5	9,6	9,7	9,7	9,6	7,9

Tabelle 3.3: Deskriptive Statistiken über die ankommenden Tupel/Sekunde zwischen zwei Join-Partnern in Datenzeit für die Joins 1-5 und insgesamt.

Wie in Abbildung 3.2 ersichtlich, sieht die Verteilung für die einzelnen Joins ziemlich ähnlich aus. Der grosse Teil bewegt sich von ca. 0,5 bis 10 Tupel pro Sekunde. Die Abbildungen enthalten Ausreisser, die klar unter einem Tupel pro Sekunde liegen. Aus diesen kann man schliessen, dass die ViSTA-TV-Daten nicht für jede Sekunde einen Event enthalten. Ausserdem sind auch einige Ausreisser stark oberhalb der 10er-Marke vorhanden.



(a) Verteilung Tupel/Sekunde Join 1 (b) Verteilung Tupel/Sekunde Join 2 (c) Verteilung Tupel/Sekunde Join 3

(d) Verteilung Tupel/Sekunde Join 4 (e) Verteilung Tupel/Sekunde Join 5 (f) Verteilung Tupel/Sekunde gesamt

Abbildung 3.2: Verteilung der Anzahl ankommender Tupel/Sekunde zwischen zwei Join-Partnern in Datenzeit für die Joins 1-5 und insgesamt. x-Achse: Tupel/Sekunde, y-Achse: Prozent.

Interessant ist zu betrachten, wie sich die Events eines Datenstroms über die Datenzeit verteilen. Für jeden Zeitstempel wurde die Anzahl Events gezählt, welche für die Joins relevant sind und mittels eines Python-Skripts in Anzahl ankommender Events pro Minute umgewandelt. Abbildung 3.3 stellt dies für die Events des Datenstroms dar. Die Datenrate steigt zu Beginn tendenziell an und bricht ab 750 Minuten (um 12:30 Uhr) zusammen. Da ab 12:50 Uhr am 01.08.2012 keine Benutzerlogs mehr in dem verwendeten Datensatz vorkommen, sondern noch sporadische Events über Programminformationen, sind nur die für die Joins relevanten Events bis 12:30 Uhr in der Grafik aufgezeigt. Dass ab 12:50 Uhr keine Benutzerlogs mehr vorhanden sind, hat jedoch keinen Einfluss auf die Resultate des Experiments. Die darauffolgenden Programminformationen stellen lediglich einen kleinen Mehraufwand für das System dar. Bemerkenswert sind die vielen unregelmässigen und kurzen Ausreisser, welche immer wieder auftreten. Diese Verteilung bis zum Mittag kann damit erklärt werden, dass bei den ViSTA-TV-Daten des 01.08.2012 tendenziell mehr Benutzer im Verlauf des Tages TV gesehen haben und die Event-Dichte deshalb gegen Mittag zunehmend stieg.

3.3.3 Fünffacher sequentieller Join

Dieses Experiment testet Streams-Esper mit einem fünffachen sequentiellen Join. Die Abfrage sucht dabei in den ViSTA-TV-Daten vom 01.08.2012 nach sechs aufeinanderfolgende Events eines

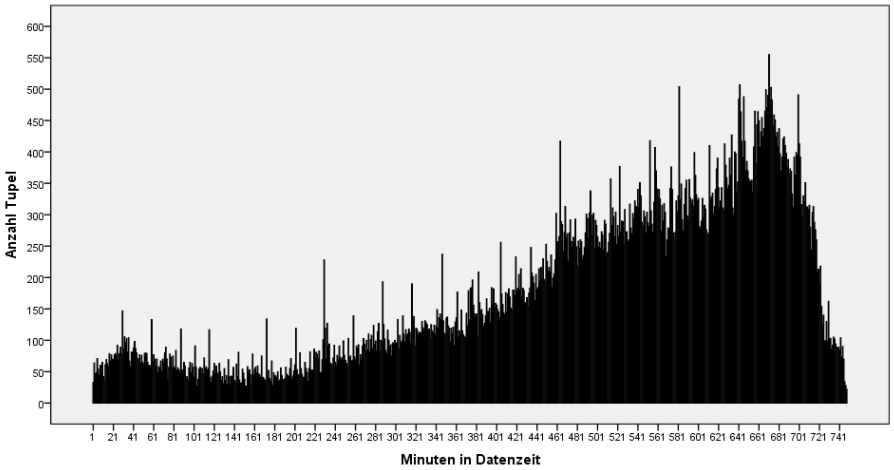


Abbildung 3.3: Anzahl vorhandener Events im verwendeten Datensatz pro Minute in Datenzeit, welche für die Joins relevant sind. Da ab 12:50 Uhr keine weiteren Benutzerdaten mehr im Datensatz sind, stoppt die x-Achse nach 741 Minuten, was der Uhrzeit 12:21 entspricht.

Benutzers. Die fixe Komponente der Abfrage ist die Anzahl der aufeinanderfolgenden Events. Variabel ist jedoch die Grösse der jeweiligen Zeitfenster bei jedem der Joins. Das Experiment enthält fünf identische Eventströme, die in Kapitel 3.3.2 beschrieben wurden. Der erste Join besteht dabei aus einem Selbstjoin mit dem ersten Eventstrom.

Die getesteten Zeitfenster sind 10 Sekunden, 100 Sekunden, 1000 Sekunden (ca. 17 Minuten) und 10'000 Sekunden (ca. 2,7 Stunden), was insgesamt zu 1'024 verschiedenen Jobs (fünf Joins, vier verschiedene Zeitfenster, 4^5) führt.

Ziel dieses Tests ist es, herauszufinden, ob diese Daten für eine Benchmark eines ISV-Systems verwendet werden können. Die Frage ist, inwiefern die verschiedenen Zeitfenster, eine schwankende Datenrate und die Anzahl Joins sich auf die Ausführungszeit und die Anzahl produzierter Resultate gesamt oder pro Join auswirken - unter Berücksichtigung von den Statistiken der Eventströme.

Experiment in Streams-Esper

Die XML-Konfigurationsdatei in Streams-Esper ist wie folgt aufgebaut (eine Beispiel XML-Konfigurationsdatei ist im Anhang zu finden):

- Zuerst werden die verschiedenen Eventströme definiert. Ein Eventstrom kann aus mehreren anderen Eventströme bestehen (ein sogenannter Multistream). Für dieses Experiment wurde ein sequentieller Multistream, bestehend aus fünf identischen CSV-Eventströme, erzeugt. Sequentiell bedeutet, dass die Ströme nacheinander in das System hereinkommen. Dies macht Sinn, da zuerst alle Resultate des ersten Joins vorhanden sein müssen, um mit dem zweiten Join fortzufahren. Sowohl der Pfad zur CSV-Datei als auch die Attribute der CSV-Ströme werden angegeben.

- Anschliessend werden die Datenstrukturen definiert, in denen Streams-Esper die Resultate der Abfragen im Hauptspeicher zwischenspeichern soll. In diesem Fall werden fünf Java-LinkedBlockingQueues[7] instanziert. Erreichen diese ihr Kapazitätslimit, so wird gewartet,

[7] http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html

bis wieder Speicher frei ist - deshalb der Name BlockingQueue. Sie funktionieren nach dem FIFO-Prinzip und ihre Knoten sind untereinander einfach verlinkt. In diesem Experiment werden fünf Queues erzeugt, eine für die Resultate jedes Joins.

- In einem nächsten Schritt wird der Hauptprozess definiert, der als Input den Multistream bekommt. Dieser besteht aus fünf Esper-Prozessoren. Im ersten werden die Systemeinstellungen sowie die interne, Java-basierte Darstellung der Eventstromattribute bestimmt. Da die ViSTA-TV-Eventströme über eigene Zeitstempel verfügen, wird der interne Timer des Systems deaktiviert und die Startzeiten der Events als Zeitstempeleigenschaften präzisiert. Die Esper-Prozessoren enthalten zusätzlich eine Angabe über die Outputqueue, in die ihre Zwischenresultate geschrieben werden sowie die eigentlichen Esper-Abfragestatements.

Im Folgenden wird die Esper-Abfrage am Beispiel des zweiten Joins erklärt:

```
1 <statement output="queue2">
2 INSERT INTO process2
3 SELECT event1.ID, event1.startTime1,
4 event1.startTime2, event2.startTime FROM
5 PATTERN [EVERY event1 = process1 -> event2 = SecondStream
6 (ID = event1.ID,event1.startTime2 < startTime,
7 startTime-event1.startTime2 < 10000) ]
8 </statement>
```

Listing 3.3: Esper-Abfrage des zweiten Joins.

Auffallend ist das SQL-ähnliche Muster des Statements. Zeile 1 beschreibt, in welche Queue dieses Statement seine Resultate schreibt. In Zeile 2 wird ein neues Event $process2$ definiert, damit dieses im nächsten Statement wiederverwendet werden kann. Das Event $process2$ besteht aus der Benutzeridentifikation sowie den Startzeiten eins bis drei. In den Zeilen 5-7 handelt es sich um ein spezielles Esper-Konstrukt, das wie folgt gelesen werden kann: Suche für jedes hereinkommende Event des ersten Prozesses ein darauffolgendes Event des zweiten Eventstroms, wobei der Benutzer derselbe sein muss, die Startzeit des zweiten Events des ersten Prozesses kleiner als die Startzeit des zweiten Eventstroms und die Differenz der zwei Startzeiten kleiner als 10'000 Millisekunden. Die zehn Sekunden agieren als Zeitfenster für das Statement. Dies ist die variable Komponente in diesem Experiment. Diese reichen von 10 Sekunden bis zu 10'000 Sekunden. Das Pattern-Konstrukt von Esper sucht zudem nach dem nächsten auftretenden Event, das die Konditionen erfüllt. Dies stellt gleichzeitig einen temporalen Aspekt in der Abfrage dar, da ein Event nach einem anderen Auftreten muss (siehe Kapitel 2.2.3).

Auswertung

Mittels eines Pyton-Skripts wurden die 1'024 XML-Ausführungsdateien mit den verschiedenen Zeitfenstern erzeugt. Diese wurden im Folgenden auf dem Kraken-Cluster der Universität Zürich ausgeführt. Ein einzelner Job hatte vier Kerne und 2 GB Hauptspeicher zur Verfügung. Die 1'024 Ausgabedateien der Jobs wurden dann mit einem anderen Python-Skript so bearbeitet, dass eine einzelne Datei mit verschiedenen Auswertungen gemäss den definierten KPIs in Kapitel 2.5 zu jedem Job erstellt wurde. Diese Datei gilt im weiteren Verlauf als Grundlage für die statistischen Auswertungen mit dem Statistikpaket SPSS[8].

[8] http://www-01.ibm.com/software/ch/de/analytics/spss/products/statistics/

Resultate

Dieses Unterkapitel gibt Auskunft über die Resultate des Experiments anhand der definierten KPIs in Kapitel 2.5.

Antwortzeiten Die Antwortzeiten in Systemzeit haben einen Mittelwert von 27 Sekunden und einen Median von 26,3 Sekunden. Obwohl die Differenz zwischen Median und Mittelwert nur 0,7 Sekunden beträgt, lässt dies auf einige hohe Ausreisser schliessen. Das bestätigt auch die beachtliche Spannweite von 19,2 Sekunden. Der höchste Wert beträgt 42,2 Sekunden und braucht damit fast doppelt so lange wie der tiefste Wert mit 23 Sekunden. In Tabelle 3.4 sind die Werte zusammengefasst.

Ausführungszeit gesamt in Sekunden	Mittelwert	27,0
	Median	26,3
	Standardabweichung	3,2
	Minimum	23,0
	Maximum	42,2
	Spannweite	19,2

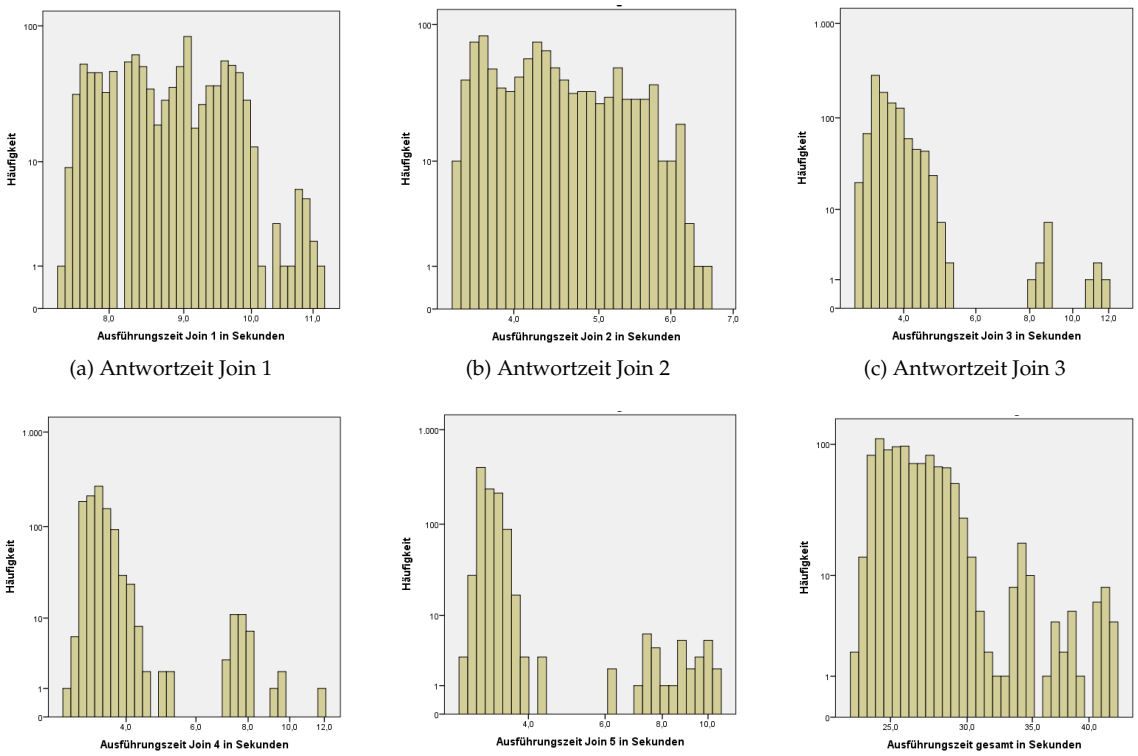
Tabelle 3.4: Deskriptive Statistiken über die Antwortzeiten in Systemzeit der 1'024 Jobs in Sekunden.

Abbildung 3.4 zeigt die verschiedenen Verteilungen der Ausführungszeiten einzelner Joins und der Ausführungszeit insgesamt. Interessant sind die Gruppierungen, die beim dritten, vierten und fünften Join auftreten (sichtbar in (c), (d) und (e)).

Da vor allem die Aussreisser spannend für diese Arbeit sind, werden die fünf grössten und fünf kleinsten in Tabelle 3.5 zusammen mit der Summe der Zeitfenster eines Jobs abgebildet. Bemerkenswert dabei ist, dass weder der Wert mit dem grössten gesamten Zeitfenster (50'000 Sekunden oder ca. 14 Stunden) noch der Wert mit dem kleinsten gesamten Zeitfenster (50 Sekunden) unter den Top 5 der Ausreisser zu finden ist. Der erste Join hat mit einem Mittelwert von 8,8 Sekunden am längsten gedauert. Der Mittelwert wird anschliessend mit jedem weiteren Join kleiner, wobei der fünfte Join einen Mittelwert der Ausführungszeit von 3,4 Sekunden aufweist. Zudem haben der erste und der zweite Join in den Ausführungszeiten keine Ausreisser.

			Zeitfenster gesamt in Sekunden	Wert
Ausführungszeit gesamt in Sekunden	Grösste Werte	1	32000	42,2
		2	32000	42,1
		3	23000	42,1
		4	41000	42
		5	40100	41,8
	Kleinste Werte	1	11030	23
		2	1310	23,1
		3	20030	23,2
		4	2120	23,2
		5	140	23,3

Tabelle 3.5: Die 5 grössten und kleinsten Ausreisser bezüglich Antwortzeiten zusammen mit dem gesamten Zeitfenster in Sekunden.



(a) Antwortzeit Join 1 (b) Antwortzeit Join 2 (c) Antwortzeit Join 3

(d) Antwortzeit Join 4 (e) Antwortzeit Join 5 (f) Antwortzeit gesamt

Abbildung 3.4: Verteilung der Antwortzeiten in Systemzeit der Joins 1-5 und gesamt. X-Achse: Sekunden in Systemzeit, y-Achse: Prozent.

Durchsatz Das CEP-System Streams-Esper konnte im Mittel 26'000 Tupel pro Sekunde verarbeiten. Vom Minimum mit 16'000 Tupeln pro Sekunde zum Maximum mit 30'000 Tupeln pro Sekunde ergibt sich eine Spannweite von 14'000.

In Abbildung 3.6 sind jeweils die Top 5 der Ausreisser der verarbeiteten Tupel pro Sekunde und der ausgegebenen Tupel pro Sekunde zusammen mit der Summe der jeweiligen Zeitfenster des Jobs dargestellt. Während die maximale und die minimale Zeitfensterdauer bei den ausgegebenen Tupeln pro Sekunde vorhanden sind, ist dies bei den verarbeiteten Tupeln pro Sekunde nicht der Fall. Bemerkenswert ist jedoch die Erkenntnis, dass bei den verarbeiteten Tupeln pro Sekunde die grössten Werte eins bis drei Mal das Zeitfenster von 10 Sekunden enthalten; bei den kleinsten Werten ein 10-Sekunden-Zeitfenster jedoch nicht auftaucht. Bis auf eine Ausnahme bei einem der grössten Werte bei den ausgegebenen Tupeln pro Sekunde ist dies hier genau umgekehrt. Die kleinsten Werte enthalten mindestens vier Mal das Zeitfenster von 10 Sekunden, gleichzeitig kommen die 10 Sekunden nur einmal bei den grössten Werten vor. Interessanterweise trifft dies genau bei dem Job zu, welcher die höchste Outputrate mit knapp 5'600 Tupeln pro Sekunde aufweist.

			Zeitfenster gesamt in Sekunden	Wert
Anzahl verarbeiteter Tupel/Sekunde	Grösste Werte	1	11030	29979
		2	1310	29880
		3	20030	29766
		4	2120	29691
		5	11030	29606
	Kleinste Werte	1	32000	16358
		2	23000	16395
		3	32000	16404
		4	41000	16413
		5	22100	16492

(a) Ausreisser verarbeitete Tupel/Sekunde

			Zeitfenster gesamt in Sekunden	Wert
Anzahl produzierter Resultate/Sekunde	Grösste Werte	1	31010	5589
		2	50000	5341
		3	32000	5320
		4	23000	5246
		5	31100	5239
	Kleinste Werte	1	10040	225
		2	10040	229
		3	140	233
		4	1040	233
		5	50	233

(b) Ausreisser ausgegebene Resultate/Sekunde

Tabelle 3.6: Die fünf grössten und kleinsten Ausreisser verarbeitet bzw. ausgegeben zusammen mit dem gesamten Zeitfenster in Sekunden.

Korrektheit der Resultate Alle Resultate, die Streams-Esper in diesem Experiment ausgibt, sind korrekt. Die Präzision (siehe Kapitel 2.5) beträgt somit 100% und die Fehlerquote 0%. Die Trefferquote unterscheidet sich je nachdem, wie gross die Zeitfenster eines Jobs sind. Mit einem gesamten Zeitfenster von ca. 14 Stunden (50'000 Sekunden) wird eine Trefferquote von 93% erreicht, wobei die Ausführungszeit hier über 41 Sekunden beträgt. Im Minimum liegt die Trefferquote bei 0,4% mit einer Antwortzeit von 24 Sekunden. Eine Trefferquote von 100%, mit einer Fehlerquote von 0% und 100% Präzision, wird erreicht, wenn das Zeitfenster jedes Joins genau der maximalen Zeitdifferenz zwischen den zeitlich am weitesten auseinanderliegenden Join-Partnern entspricht (siehe Abbildung 3.2 in Kapitel 3.3.2). Die minimale Zeit (es wurde nur ein Test mit den entsprechenden Zeitfenstern gemacht), alle korrekten Resultate auszugeben, beläuft sich auf 35 Sekunden. Die minimale Ausführungszeit beträgt 23 Sekunden mit einer Trefferquote von 1,1%.

Abbildung 3.5 zeigt die ausgegebenen Tupel pro Sekunde in Systemzeit für den Job mit einer Trefferquote von 100% (maximale Zeitdifferenz zwischen zwei Join-Partnern als Zeitfenster bei jedem Join). Interessant ist die angegebene Unterteilung in Ausführungssekunden der verschiedenen Joins. Die Anzahl der ausgegebenen Resultate steigt bei jedem Join zu Beginn an. Bemerkenswert sind auch die drei Sekunden zwischen Join 3 und Join 4, in denen das System keine Resultate ausgibt. Die Menge an aggregierten Resultaten pro Join sinkt von Join 1 bis zu Join 5 fortlaufend.

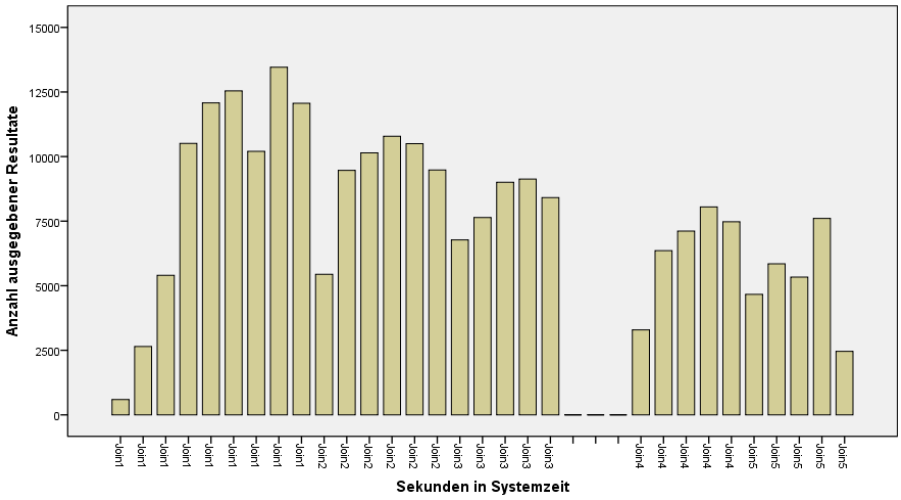


Abbildung 3.5: Ausgegebene Tupel/Sekunde in Systemzeit für den Job mit einer Trefferquote von 100%. X-Achse: Sekunden in Systemzeit beschriftet mit dem ausführenden Join, y-Achse: Anzahl ausgegebener Resultate.

Kapitel 4

Diskussion

Dieses Kapitel bespricht die Resultate aus der Evaluation (Kapitel 3). Zu Beginn werden CEP-Systeme im Allgemeinen (Kapitel 4.1) und die Wichtigkeit des Datensatzes für eine Benchmark kommentiert (Kapitel 4.2), dann wird auf das Experiment des fünffachen sequentiellen Joins und die Hypothesen aus Kapitel 1.2 eingegangen (Kapitel 4.3). Anschliessend geht es darum, andere Forschungen im Bereich von Benchmarking ISV-Systeme zu präsentieren (Kapitel 4.4) und schliesslich die Limitationen dieser Arbeit darzulegen (Kapitel 4.5).

4.1 CEP-Systeme

Ein grosser Vorteil von CEP-Systemen ist, dass die Ereignisse geordnet ankommen. Zusammen mit Konstrukten, welche diese Eigenschaft berücksichtigen (z.B. das "PATTERN"-Konstrukt im Experiment, Kapitel 3.3.3), ergeben sich weit mehr Möglichkeiten als in traditionellen DBMSs. Es wurde versucht, den fünffachen sequentiellen Join in einer Abfrage in PostgreSQL auszuführen. Nach über vier Stunden wurde die Ausführung jedoch abgebrochen und die Abfrage in einzelne Teile zerlegt. Zum Vergleich: Das Experiment in Streams-Esper läuft auf derselben Maschine in weniger als einer Minute durch.

4.2 Auswahl und Analyse eines Datensatzes für eine Benchmark

Wie bereits in Kapitel 3 erwähnt, ist eine statistische Analyse des Datensatzes für eine Benchmark von grosser Bedeutung. Es muss bei der Auswahl oder Erstellung eines Datensatzes darauf geachtet werden, was überhaupt getestet werden soll. Ist das Zeitfenster in einer Abfrage nicht durch deren Logik vorgegeben, ist es wichtig, die Zeitfenster in Abfragen basierend auf den Statistiken der Daten zu wählen. Bestimmt die Qualitätsanforderung, dass alle Resultate erhalten werden müssen, so gilt es, das Zeitfenster gemäss der maximalen Distanz zwischen zwei Join-Partnern in jedem Join zu wählen. Sind Join-Partner im Datensatz weit auseinander, kann das Verhalten des Systems bei höherer Nutzung von Zwischenspeicher betrachtet werden. Sind hingegen sehr viele Join-Partner nah beieinander, wird mehr Prozessorleistung beansprucht. Dies sind Beispiele, welche zeigen, wie wichtig die Auswahl des Datensatzes für eine Benchmark ist.

4.3 Experiment des fünffachen sequentiellen Joins

Das Experiment des fünffachen sequentiellen Joins wurde systematisch aufgebaut. Ausgehend von der Analyse verschiedener Join-Arten (Kapitel 2.2) sowie den Eigenschaften der Joins im Kontext von ISV-Systemen (Kapitel 2.3) und deren Herausforderungen (Kapitel 2.4) wurden die Hypothesen (Kapitel 1.2) definiert. Zur Erinnerung werden die aufgestellten Hypothesen noch einmal aufgeführt:

Ein Informationsstrom-Verarbeitungssystem kann über folgende Parameter mit sequentiellen Joins gestresst werden:

- *Datenrate der Informationsströme*
- *Datenmenge der Informationsströme*
- *Rechenaufwand der einzelnen Joins*
- *Unterschiedliche Zeitfenster*
- *Anzahl sequentieller Joins*

Ein Teil der Parameter, nämlich schwankende Datenraten, unterschiedliche Zeitfenster und die Anzahl sequentieller Joins, wurde im Experiment des fünffachen sequentiellen Joins aufgegriffen und getestet (Kapitel 3). Anhand der definierten Key Performance Indicators in Kapitel 2.5 wurden die Resultate ausgewertet (Kapitel 3.3.3). Die 1'024 unterschiedlichen Jobs mit Zeitfenstern von 10, 100, 1'000 und 10'000 Sekunden ergeben sehr unterschiedliche Ergebnisse bezüglich Antwortzeiten, Durchsatz, Trefferquoten u.a. Dies bedeutet grundsätzlich, dass ein Teil der Hypothese nicht grundfalsch ist. Die ausgegebenen Resultate von Streams-Esper sind in diesem Experiment immer korrekt. Folglich ist die Präzision bei allen Jobs 100% bei einer Fehlerquote von 0%. Die Abfragen wurden also so gestaltet, dass es keine Fehler geben konnte, die Anzahl der Resultate jedoch von den verschiedenen Zeitfenstern abhängt. Die Antwortzeiten reichen von 23 Sekunden bis über 42 Sekunden, der Durchsatz von 16'000 Tupeln/Sekunde bis zu 30'000 Tupeln/Sekunde und die Trefferquote von 0,4% bis zu 93%. Im Folgenden werden diese Unterschiede unter Berücksichtigung der Hypothesen diskutiert.

Hypothese: Schwankende Datenraten können ein ISV-System stressen. In Kapitel 3.3.2 wurde gezeigt, dass der fünffach verwendete Eventstrom über eine schwankende Datenrate verfügt. Dies ist auch in der Abbildung 3.3 zu sehen. Die Datenrate steigt beim verwendeten Datensatz des ViSTA-TV-Projekts bis etwa zur Hälfte an und bricht dann ein, da ab 12:50 Uhr des 01.08.2012 keine Benutzer-Ereignisse mehr in den Daten sind, sondern nur noch Programminformationen. Interessant ist die Abbildung 3.5, die zeigt, dass die einzelnen Joins zu Beginn der Ausführung weniger Resultate liefern als im weiteren Verlauf. Dies widerspiegelt sich auch im verwendeten Datensatz. Da die Ausgabe von vielen Resultaten in kurzer Zeit auch mehr Rechen- und Zeitaufwand benötigt, kann ein ISV-System mit schwankenden Datenraten prinzipiell gestresst werden. Interessant wäre es, die Datenrate fortlaufend künstlich zu erhöhen, um zu sehen, wie das System darauf reagiert bzw. wann es an seine Grenzen stösst.

Hypothese: Mit steigender Anzahl aufeinanderfolgender sequentieller Joins können ISV-Systeme gestresst werden. Der Mittelwert der Ausführungszeit des ersten Joins beträgt 8,8 Sekunden und nimmt für jeden weiteren Join ab. Der zweite Join hat einen Mittelwert von 4,5 Sekunden, der dritte von 3,8 Sekunden, der vierte von 3,5 Sekunden und der fünfte von 3,4 Sekunden. Dies entspricht in etwa dem Verlauf der möglichen Resultate der Joins (zu sehen in Tabelle 3.1). Dies

bedeutet, dass das System mehr Aufwand hat, je mehr Resultate ein Join liefert (Zwischenspeicherung der Resultate, Ausgabe im Standard-Output im Testfall etc.). Die Hypothese, dass eine steigende Anzahl sequentieller Joins ein ISV-System stressen kann, wurde in diesem Experiment also nicht bestätigt. Mehr sequentielle Joins mit sinkenden Resultaten ergeben keinen überproportionalen Aufwand für das System. Mit einem sequentiellen Join, bei dem die Anzahl Resultate pro Join konstant bleibt oder sogar grösser wird, könnte diese Hypothese möglicherweise aber gestützt werden. In einem verteilten System stellt sich weiter die Frage, ob die einzelnen Joins unabhängig voneinander auf verschiedenen Maschinen ausgeführt werden können. Ist dies der Fall, macht es wenig Sinn einen Stresstest mit steigender Anzahl aufeinanderfolgender sequentieller Joins durchzuführen.

Hypothese: Variable Zeitfenster in sequentiellen Joins können ein ISV-System stressen. Die unterschiedlich grossen Zeitfenster von 10, 100, 1'000 und 10'000 Sekunden hatten in diesem Experiment den grössten Einfluss. Jobs mit mehrfach vorhandenen Zeitfenstern von 10 Sekunden hatten tendenziell einen hohen Durchsatz, lieferten jedoch wenige Resultate. Umgekehrt wiesen Jobs mit Zeitfenstern von 10'000 Sekunden einen tiefen Durchsatz mit vielen Resultaten auf. Dies bedeutet, dass Streams-Esper grundsätzlich mehr Aufwand betreiben muss, wenn viele Resultate berechnet werden müssen. Die grossen Unterschiede zeigen, dass die Zeitfenster viel Einfluss auf die Ausführungszeit, den Durchsatz und die Trefferquote haben und somit ein ISV-System durchaus stressen können. Je grösser das Zeitfenster, desto mehr Zwischenresultate müssen gespeichert und desto mehr Resultate berechnet werden. Bei einem Stresstest mit einem sequentiellen Join ist es wichtig darauf zu achten, dass die Zeitfenster der ersten Joins genügend gross gewählt werden, damit viele Resultate geliefert werden. Ist das Zeitfenster am Anfang klein, müssen im weiteren Verlauf der Abfrage weniger Daten gespeichert und berechnet werden, was einen höheren Durchsatz bedeutet, aber auch, dass insgesamt weniger Resultate ausgegeben werden. Dies stellt einen Zielkonflikt dar. Wichtig dabei ist, den verwendeten Datensatz zu kennen und basierend auf diesem Wissen die Zeitfenster zu wählen.

4.4 Stand der Forschung

Dieses Kapitel stellt drei existierende Benchmarks für ISV-Systeme vor. Zuerst wird Linear Road, ein Benchmark für DSMSs, beschrieben. Anschliessend folgen die Erläuterungen zu zwei Benchmarks für SFP-Systeme. Zum Schluss erfolgt eine kurze Analyse der Benchmarks basierend auf den Erkenntnissen von Scharrenbach *et al.* (2013).

Linear Road Linear Road (Arasu *et al.*, 2004) ist eine Benchmark für Performanzanalysen von DSMSs und anderen Systemen (z.B. traditionelle DBMSs). Es simuliert ein Maut-System für Motorfahrzeuge auf der Autobahn einer grossen, fiktiven Stadt (Arasu *et al.*, 2004). Linear Road berechnet dynamisch die Gebühren basierend auf aktuellen Verkehrsstaus, Unfällen und historischen Daten. Das System simuliert ein Verkehrsüberwachungssystem, bemerkt Unfälle und Staus und benachrichtigt die Fahrer über solche Ereignisse. Zusätzlich zu den dynamischen Daten, werden historische Informationen über jede Meile der Autobahn gesammelt (z.B. wie oft ein Unfall vorkommt etc.). Da die Daten nicht semantisch im Sinne des RDF-Datenmodells sind, können keine Inferenzmechanismen und Graph-Muster-Abfragen getestet werden. Linear Road enthält einen historischen Datengenerator, einen Verkehrs-Simulator, der die Datenströme erstellt, einen Daten-Treiber, der die Daten in das zu testende System einspeist sowie einem Validierer, der den Output auf Korrektheit und die Anforderungen an die Antwortzeiten prüft (Arasu *et al.*, 2004).

SRBench SRBench (Zhang *et al.*, 2012) ist eine Benchmark für SFP-Systeme. Es besteht aus einem realen Datensatz aus der Linked Open Data Coud[1] mit einer grossen Anzahl an Anwendungsfällen von realen Szenarien (Zhang *et al.*, 2012). Die Benchmark testet SFP-Systeme auf verschiedene Funktionen, u.a. auf die Unterstützung von Daten mit Semantik.

SRBench verwendet drei verschiedene Datensätze: LinkedSensorData[2], GeoNames[3] und DBpedia[4]. Die LinkedSensorData enthalten Wetter-Daten aus der USA, beobachtet seit 2002. Um die Verbindung verschiedener Datensätze testen zu können, werden GeoNames, eine geografische Datenbank welche alle Länder und über acht Millionen Orte enthält, sowie DBpedia verwendet, welche mit GeoNames verbunden werden kann.

SRBench enthält 17 Abfragen, die unter anderem nur Datenströme, Datenströme und Hintergrunddaten und Datenströme aus unterschiedlichen Datensätzen testet. Zusätzlich werden auch Inferenzmechanismen und Schlussfolgerungen in den Abfragen verwendet. Beispiele für Abfragen sind einmaliger Regen während einer Stunde, das Auftreten eines Wirbelsturms, minimale und maximale Tagestemperatur, Joins mit statischen Daten (Orte mit starkem Schneefall), stündliche Durchschnittstemperatur und Feuchtigkeit von grossen Städten und Orte, bei welchen die Windgeschwindigkeit in der letzten Stunde höher war als bei einem bekannten Wirbelsturm (Zhang *et al.*, 2012).

SLD-Bench Der SLD-Bench (Le-phuoc *et al.*, 2012) enthält Linked Stream Data, Daten die das RDF-Datenmodell erweitern für Datenströme von Sensoren und sozialen Netzwerken. Da soziale Netzwerke sehr stark verbundene Graphen von Benutzern darstellen, werden sie für diesen Benchmark benutzt. Die Daten bestehen aus Datenströmen und statischen Daten. Die Datenströme enthalten GPS-Ströme (geben an, wo Benutzer aktuell sind), Ströme aus Beiträgen und Kommentaren sowie Strömen von Fotos. Zusätzlich werden statische Daten über die Benutzer, wie Benutzername und andere Informationen aus dem Benutzerprofil verwendet. Die Benchmark testet die Systeme auf drei verschiedene Arten: Funktionalitäten (Abfragen nur auf Datenströmen und Datenströme mit Hintergrunddaten) , Korrektheit und Performanz (maximaler Durchsatz, Skalierbarkeit mit mehr gleichzeitigen Abfragen und grösseren statischen Datensätzen) (Le-phuoc *et al.*, 2012). Der soziale Graph wird synthetisch erstellt. In den Abfragen kommen keine Inferenzmechanismen vor.

Analyse Alle drei Benchmarks beziehen ihre Daten von einem Sensorsystem (Scharrenbach *et al.*, 2013). Deshalb ist ein Stresstest bezüglich Lastenausgleich grundsätzlich möglich. Beim SRBench sind die Sensordaten der Wetterstationen jedoch mehr oder weniger stabil. SLD-Bench und Linear Road hätten das Potenzial auf unregelmässige Datenraten, jedoch ist nicht beschrieben, wie dies getestet wird bzw. durch welche Parameter man hohe Datenraten erreichen könnte (Scharrenbach *et al.*, 2013).

Die beiden Benchmarks Linear Road und SLD-Bench haben Anwendungsfälle von sequentiellen Joins, jedoch werden keine temporalen Joins getestet obwohl dies bei beiden möglich wäre (Scharrenbach *et al.*, 2013). Die sequentiellen Joins werden zudem nicht in unterschiedlich komplexen Abfragen getestet.

Joins von Datenströmen mit Hintergrunddaten werden in allen drei Benchmarks durchgeführt. SRBench und SLD-Bench verfügen allerdings über wenige Abfragen, die fix sind. Deshalb kann ein System damit nicht systematisch gestresst werden (Scharrenbach *et al.*, 2013). Joins mit Hintergrunddaten, welche auf der Festplatte gespeichert sind, können nur mit dem SRBench und dem SLD-Bench getestet werden. Die statistischen Daten beim Linear Road passten grundsätzlich

[1] http://linkeddata.org/
[2] http://wiki.knoesis.org/index.php/LinkedSensorData
[3] http://www.geonames.org/
[4] http://dbpedia.org/

in den Hauptspeicher. Keiner der Benchmarks berücksichtigt allenfalls wechselnde Hintergrunddaten (Scharrenbach *et al.*, 2013).

Keiner dieser Benchmarks erfüllt alle Anforderungen an eine systematische Benchmark für ein ISV-System. Dies ist jedoch auch sehr schwierig zu erreichen. Einerseits durch die vielen Herausforderungen, andererseits wegen den heterogenen Systemen und den unterschiedlichen Einsatzzwecken.

4.5 Limitationen der Arbeit

Diese Arbeit ist dem Versuch gewidmet, systematisch sequentielle Joins in ISV-Systemen zu testen. Limitationen stellen der verwendete Datensatz in Verbindung mit dem Experiment, das einmalig durchgeführte Experiment und das Fehlen von Vergleichsmöglichkeiten dar. Dadurch, dass Joins in ISV-Systemen kaum erforscht sind, basiert vieles dieser Arbeit auf eigener Forschung und eigenen Annahmen.

Der verwendete Datensatz aus dem ViSTA-TV-Projekt beinhaltet reale Daten. Diese verfügen, wie in Kapitel 3.3.2 dargelegt, über eine schwankende Datenrate, sind jedoch nicht für einen Stresstest eines ISV-Systems optimiert. Dank der schwankenden Datenrate konnte jedoch deren Auswirkung auf eine Abfrage in einem CEP-System ansatzweise gezeigt werden. Einen synthetischen Datensatz zu erstellen, erwies sich im Rahmen einer solchen Arbeit gerade für sequentielle Joins als schwierig und zu zeitaufwendig. Kleine synthetische Datensätze wurden aber verwendet, um zu testen, ob die Abfragen korrekt erstellt wurden.

Die 1'024 Jobs im Experiment in Kapitel 3.3.3 wurden aus Zeitgründen nur einmal durchgeführt. Somit können die Ausführungszeiten der Jobs durch Hardwareschwankungen des Karken-Clusters verzerrt sein. Dies ist insofern problematisch, als dass vor allem die Ausreisser für diese Arbeit von hoher Bedeutung sind. Zusätzlich war das Cluster während der Ausführung nicht nur für die Experimente dieser Arbeit in Betrieb. Mehrfaches Ausführen der Jobs und die Berechnung der Mittelwerte hätten diese Limitation vermeiden können. Weiter ist nicht klar, wie stark das Ausgeben der Antworten in den Standard-Output für die spätere Evaluation die Performanz der Jobs beeinflusst hat. Ein Job, der mehr Resultate lieferte, musste auch mehr in den Standard-Output schreiben.

Die Verifikation der Hypothesen ist schwierig, da nur ein Experiment mit einer Abfrage (jedoch mit verschiedenen Zeitfenstern) durchgeführt wurde. Die Auswirkungen der verschiedenen Zeitfenster konnten gut aufgezeigt werden. Für die anderen Hypothesen ist es insofern problematisch, als dass Vergleiche fehlen. Wie würde das Resultat des Experiments mit einem Datensatz aussehen, der eine sehr hohe Datenrate besitzt? Was passiert, wenn die Datenmenge fortlaufend erhöht wird? Welchen Einfluss haben die verschiedenen Join-Konditionen auf den Rechenaufwand? Was passiert, wenn aus fünf sequentiellen Joins zehn gemacht werden? All diese Fragen können durch die vorliegende Arbeit nicht vollständig beantwortet werden. Da die Ausführung und vor allem die Auswertung eines Experiments sehr aufwendig ist, musste in dieser Arbeit darauf verzichtet werden, mehrere Experimente durchzuführen.

Kapitel 5

Zusammenfassung und Ausblick

Diese Arbeit beschreibt ein methodisches Vorgehen, um sequentielle Joins in ISV-Systemen systematisch zu testen. Um die unendlichen Informationsströme verarbeiten zu können, existieren Fenster-Operatoren in Abfragesprachen für ISV-Systeme, welche es erlauben, die Daten in Teilen zu verarbeiten. Wie in Kapitel 4.4 beschrieben wurde, gibt es bereits Benchmarks für solche Systeme. Diese berücksichtigen jedoch wichtige Aspekte nicht. Diese Arbeit nimmt die generischen Vorschläge von Scharrenbach *et al.* (2013) auf und versucht, diese auf die speziellen Eigenschaften und Herausforderungen von Joins in ISV-Systemen anzuwenden. Dazu wurden die Arten, Eigenschaften und Herausforderungen von einfachen, temporalen und sequentiellen Joins analysiert und beschrieben. Es wurde ein Experiment mit einem fünffach sequentiellen Join mit unterschiedlichen Zeitfenstern und einem realen Datensatz von IPTV-Logs durchgeführt. Der Datensatz wurde ausgiebig statistisch analysiert und die Zeitfenster der Abfragen basierend auf diesen Erkenntnissen gewählt. Die Ergebnisse zeigen, dass unterschiedliche Grössen der Zeitfenster einen beachtlichen Einfluss auf den Durchsatz, die Trefferquote und die Antwortzeit haben. Die schwankende Datenrate im verwendeten Datensatz konnte das System insofern stressen, als dass es durch eine höhere Datenrate mehr Resultate zu berechnen gab und somit die Antwortzeit und der Durchsatz leiden mussten.

Das Kapitel 4 enthält bereits einige Vorschläge für zukünftige Forschungen. So könnten beispielsweise die Datenraten fortlaufend erhöht werden, um ein System an seine Grenzen zu bringen. Sequentielle Joins mit überproportionaler Anzahl an Resultaten pro Join könnten getestet werden. Interessant wären sicherlich auch Joins mit sehr grossen Zeitfenstern und weit auseinanderliegenden Join-Partnern. Weiter ist noch offen, wie und ob sequentielle Joins in verteilten Systemen getestet werden können.

In Kapitel 1.2 wird beschrieben, wie ein systematischer Test für sequentielle Joins in einem ISV-System aufzubauen ist. Was in dieser Arbeit nicht gezeigt werden konnte und sich deshalb für zukünftige Forschungen eignet, sind Stresstests, bei welchen die Abfragen methodisch komplexer gemacht werden. Durch Vergleiche zwischen verschiedenen Abfragen könnten letztlich Rückschlüsse auf die aufgestellten Hypothesen gezogen werden. Spannend wäre auch das Erforschen der Zusammenhänge und Prioritäten zwischen den definierten Parametern (Datenrate, Datenmenge, Rechenaufwand eines Joins, Zeitfenster, Anzahl sequentieller Joins). Mit welchem Parameter bringt man ein ISV-System am schnellsten an seine Grenzen? Welche Kombinationen verstärken sich? Es gibt noch viele offene Fragen und viel zu tun. Diese Arbeit ist ein erster Ansatz die Vorschläge von Scharrenbach *et al.* (2013) zu nehmen und einen Teil der Forschungslücke zu schliessen.

Kapitel 6

Anhang

Anhang A

Output von Streams-Esper

Dieses Kapitel zeigt die Stellen in Streams-Esper auf, welche für die Evaluation verändert wurden.

Listing A.1: Der Konstruktor der EsperProcessor.java Klasse. Die Startzeit wird ausgegeben.

```
public EsperProcessor() {
   //Output the starting time of the processor in milliseconds
   System.out.println("Starting_Time," + System.currentTimeMillis());

   _configuration = new Configuration();
   _epStatements = new ArrayList<String>();
   _statementSinksMap = new HashMap<String, String>();
   _startTimestampMap = new HashMap<String, String>();
   _endTimestampMap = new HashMap<String, String>();
   _currentTime = Long.MIN_VALUE;
}
```

Listing A.2: Die update Methode der EsperStatementSubscriber.java Klasse. Diese wird immer Aufgerufen, wenn Esper ein Event an den nächsten Prozessor weitergibt. Alle Attribute sowie die aktuelle Systemzeit werden ausgegeben.

```
/**
 *
 * @param values
 * the values of an Esper event.
 * @throws Exception
 */
public void update(Object... values) throws Exception {
   synchronized (_sink) {
      Data item = DataFactory.create();

      //Create a String with the current system time
      String s = System.currentTimeMillis() + ",";

      for (int i = 0; i < values.length; ++i) {

         item.put(_keys[i], (Serializable) values[i]);

         //Append the current value of the event to the output string
         s = s + (Serializable) values[i] + ",";
      }

      System.out.println(s);

      _sink.write(item);
   }
}
```

Listing A.3: Die Methode finish in der EsperProcessor.java Klasse. Die Endzeit wird ausgegeben.

```
public void finish() throws Exception {
   _log.debug("Finishing {} ...", this.getClass());
   if (_epService != null) {
      _epService.destroy();
   }
   _log.debug("Finished finishing {}", this.getClass());

   //Output the ending time of the processor in millisecs
   System.out.println("Ending_Time," + System.currentTimeMillis());
}
```

Anhang B

Datenaufbereitung und Statistiken

Dieses Kapitel gibt weitere Einblicke in die Datenaufbereitung sowie zusätzliche Statistiken über den verwendeten Datensatz.

B.1 Datenaufbereitung

Listing B.1: SQL Abfrage-Datei query_20120801_epg.sql. Die Programminformationen des 01.08.2012 werden vom ViSTA-TV-Datenbankserver geholt.

```
SET CLIENT_ENCODING TO 'UTF8';
COPY
(SELECT t_start, t_end, sid, title
FROM
epg_zattoo
WHERE
t_start BETWEEN '2012-08-01 00:00:00'
AND '2012-08-01 23:59:59'
ORDER BY t_start)
TO stdout
WITH CSV
```

Listing B.2: SQL Abfrage-Datei query_20120801_users.sql. Die Benutzerdaten des 01.08.2012 werden vom ViSTA-TV-Datenbankserver geholt.

```
SET CLIENT_ENCODING TO 'UTF8';
COPY
(SELECT t_start, t_end, uid, channel
FROM
viewership_zattoo
WHERE t_start BETWEEN '2012-08-01 00:00:00'
AND '2012-08-01 23:59:59'
ORDER BY t_start)
TO stdout
WITH CSV
```

Listing B.3: Die beiden Befehle, um die Abfragen auszuführen.

```
psql -h pg.ifi.uzh.ch -f query_20120801_epg.sql -o epg.csv vistatv vistatv_read

psql -h pg.ifi.uzh.ch -f query_20120801_users.sql -o users.csv vistatv
    vistatv_read
```

Listing B.4: Ausschnitt aus 01_TimeStamp_2_long.py. Die Zeitstempel der Inputdateien werden in Millisekunden seit der Epoche umgewandelt.

```
# create the input format for timestamps one and two
     sourceFormat1 = '%Y-%m-%d %H:%M:%S'
     sourceFormat2 = '%Y-%m-%d %H:%M:%S'

     # create csv writer
     writer = csv.writer(open(outputpath + outputfilename, 'w', encoding='utf8
         '), delimiter=',', lineterminator='\n')

     for row in reader:
        # read the time in sourceformat1 and convert it to milliseconds since
            the epoch
        startTime = row[0]
        sourceStartTime = datetime.datetime.strptime(startTime, sourceFormat1)
        start = time.mktime(sourceStartTime.timetuple())
        start = start * 1000

        # read the time in sourceformat2 and convert it to milliseconds since
            the epoch
        endTime = row[1]
        sourceEndTime = datetime.datetime.strptime(endTime, sourceFormat2)
        end = time.mktime(sourceEndTime.timetuple())
        end = end * 1000

        # write the tuples with the timestamp in milliseconds to the output
            file
        # making the integer identifier of the input tuple to a string (adding
             "ID")
        # (Esper has a better support for string identifiers than for integers
            )
        writer.writerow([str(round(start)), str(round(end)), row[2] + "ID",
            row[3]])
```

Listing B.5: Ausschnitt aus 02_Sort_By_TimeStamp.py. Die Tupel der Inputdateien werden aufsteigend nach dem Startzeitpunkt sortiert. Tupel bei welchen der Startzeitpunkt (erstes Attribut im Tupel) und der Benutzeridentifikator (drittes Attribut im Tupel) gleich sind, werden ignoriert.

```
with open(inputpath + infile, 'r', encoding="ISO-8859-1") as csvfile:
      outputfilename = infile

      # read the first n bytes for checking delimiter and quotes
      dialect = csv.Sniffer().sniff(csvfile.read(1024000))

      # reset to first position fo file after sniffing the dialect
      csvfile.seek(0)

      # create the csv reader
      reader = csv.reader(csvfile, dialect)

      # adds the input tuples to the list, duplicates are ignored
      present = 0
      for row in reader:
         for row2 in rows:
            if row[0] == row2[0] and row[2] == row2[2] :
               present = 1
               break
         if present == 0:
            rows.append(row)
            i=i+1
         present =0

# sort the list in ascending order
rows.sort()
```

Listing B.6: Ausschnitt aus Arriving_Events_Per_Minute.py. Dieses Skript summiert die Anzahl Events pro Minute in Datenzeit auf. Wird die grösse des Arrays angepasst sowie anstatt auf Minuten auf Sekunden aufsummiert kann die Anzahl ausgegebener Resultate eines Jobs pro Sekunde berechnet werden.

```
# create an array of size 1440, the size is equal to the minutes of a day
    (60 minutes * 24 hours)
array = [0]*1440

# add up the number of events for every minute in the input file
for row in reader:
   # get the hours of the input tuple
   hour= datetime.datetime.fromtimestamp(int(row[0])/1000).strftime('%H')
   # get the minutes of the input tuple
   min= datetime.datetime.fromtimestamp(int(row[0])/1000).strftime('%M')
   # adding up the nuber of events at the given position
   array[int(hour)*60+int(min)]+=int(row[1])


# output the content of the array together with the minutes
minute = 0
for x in array:

   writer.writerow([minute, x])

   minute = minute +1
```

B.2 Weitere Statistiken zum verwendeten Datensatz

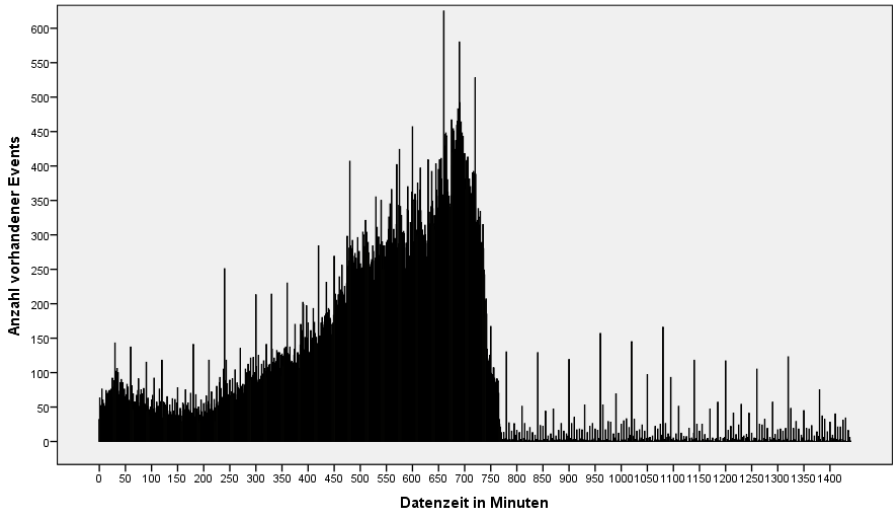


Abbildung B.1: Anzahl ankommender Events über die Datenzeit der ViSTA-TV-Daten des 01.08.2012 in Minuten. Ab 12:50 Uhr sind keine Benutzerdaten mehr vorhanden, weshalb die Datenrate einbricht.

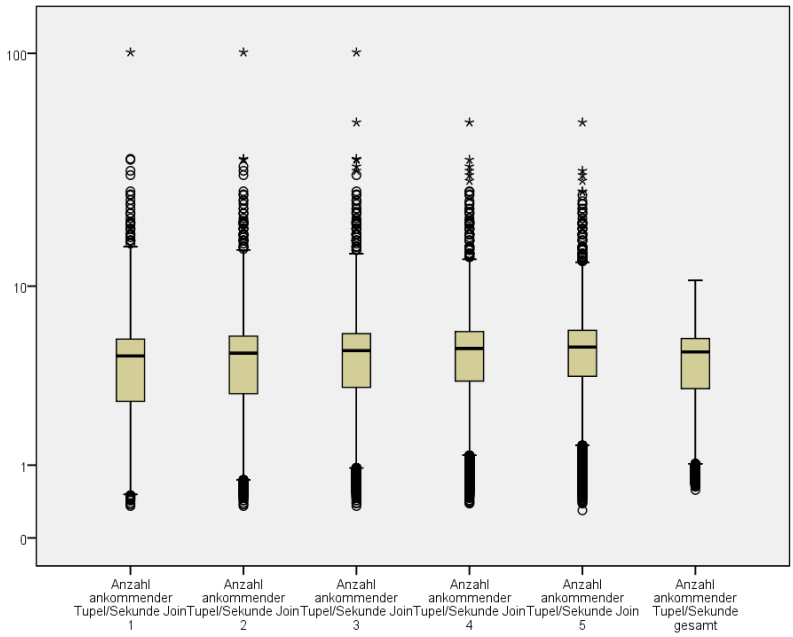


Abbildung B.2: Boxplots für die Anzahl ankommender Tupel/Sekunde der Joins 1-5 und insgesamt.

Anhang C

Fünffacher sequentieller Join

Dieses Kapitel enthält den Anhang zum Experiment des fünffachen sequentiellen Joins. Zuerst wird der Aufbau des Experiments präsentiert. Danach folgen zusätzliche Informationen über die Auswertung sowie weitere Statistiken zu den Resultaten.

C.1 Aufbau des Experiments

Listing C.1: Abgekürztes Beispiel einer XML-Konfigurationsdatei mit Zeitfenstern von 100 Sekunden (100'000 Millisekunden). Drei vertikale Punkte bedeuten, dass etwas weggelassen wurde.

```
<container>
   <!-- Definition of the input data -->
   <Stream id="merged" class="stream.io.multi.SequentialMultiStream">
      <Stream id="FirstStream"
            class="stream.io.CsvStream" url="classpath:/data/input.csv"
            keys="start_time,end_time,ID,data" />
            .
            .
            .
      <Stream id="FifthStream"
            class="stream.io.CsvStream" url="classpath:/data/input.csv"
            keys="start_time,end_time,ID,data" />
   </Stream>

   <!-- Definition of the output queues, where the results are stored -->
   <Queue id="queue1" />
         .
         .
         .
   <Queue id="queue5" />
   <!-- Definition of the main process with the defined input streams of "
      merged" -->
   <process id="esper" input="merged">
      <!-- Possible to set delays in the input streams -->
      <stream.flow.Delay time="0ms" />
```

```
<!-- The first EsperProcessor with output queue "queue1" -->
<ch.uzh.ifi.ddis.ifp.streams.EsperProcessor
   output="queue1">
   <!-- The configuration settings -->
   <config>
      <esper-configuration
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://www.espertech.com/schema/esper"
         xsi:schemaLocation="http://www.espertech.com/schema/esper
         http://www.espertech.com/schema/esper/esper-configuration-2.0.
             xsd">

         <engine-settings>
            <defaults>
               <threading>
                  <!-- internal timer is disabled since the data has own
                      timestamps -->
                  <internal-timer msec-resolution="1" enabled="false" />
               </threading>
            </defaults>
         </engine-settings>
         <!-- Setting the property names and classes of the input streams
              -->
         <event-type name="FirstStream">
            <java-util-map start-timestamp-property-name="start_time">
               <map-property name="start_time" class="long" />
               <map-property name="end_time" class="long" />
               <map-property name="ID" class="String" />
               <map-property name="data" class="String" />
            </java-util-map>
         </event-type>
                  .
                  .
                  .
         <event-type name="FifthStream">
            <java-util-map start-timestamp-property-name="start_time">
               <map-property name="start_time" class="long" />
               <map-property name="end_time" class="long" />
               <map-property name="ID" class="String" />
               <map-property name="data" class="String" />
            </java-util-map>
         </event-type>
      </esper-configuration>
      <!-- The first Esper statement with a selfjoin on FirstStream
          searching for two sequential events of a user within 100000
          milliseconds in data time -->
      <statement output="queue1">insert into p1 select
         e2.ID as ID, e2.start_time
```

```
                as st1, e1.start_time as st2
                from pattern
                [every e2 = FirstStream -> e1 = FirstStream
                (ID = e2.ID,e2.start_time &lt; start_time,
                start_time-e2.start_time &lt; 100000)]
             </statement>
          </config>
       </ch.uzh.ifi.ddis.ifp.streams.EsperProcessor>
       <!-- The second EsperProcessor with output queue "queue2". The
           configuration settings do not have to be defined again. -->
       <ch.uzh.ifi.ddis.ifp.streams.EsperProcessor
          output="queue2">
          <config>
             <statement output="queue2">insert into p2 select
                e2.ID as ID, e2.st1
                as st1, e2.st2 as st2, e1.start_time as st3
                from pattern
                [every e2 = p1 -> e1 = SecondStream
                (ID = e2.ID,e2.st2 &lt; start_time,
                 start_time-e2.st2 &lt; 100000)]
             </statement>
          </config>
       </ch.uzh.ifi.ddis.ifp.streams.EsperProcessor>
                   .
                   .
                   .
       <!-- The fifth EsperProcessor with output queue "queue5" -->
       <ch.uzh.ifi.ddis.ifp.streams.EsperProcessor
          output="queue5">
          <config>
             <statement output="queue5">insert into p5 select
                e2.ID as ID, e2.st1
                as st1, e2.st2 as st2, e2.st3 as st3, e2.st4 as
                st4,e2.st5 as st5, e1.start_time as st6
                from pattern
                [every e2 = p4 -> e1 = FifthStream
                (ID = e2.ID,e2.st5 &lt; start_time,
                start_time-e2.st5 &lt; 100000)]
             </statement>
          </config>
       </ch.uzh.ifi.ddis.ifp.streams.EsperProcessor>
    </process>

</container>
```

Listing C.2: Python-Skript, welches die XML-Konfigurationsdateien mit unterschiedlichen Zeitfenstern erstellt. In der Inputdatei sind die Zeitfenstervariablen als Strings vorhanden (VariableWindowValue1, VariableWindowValue2 etc.)

```
#declaration of the input and output files
inputpath = '../Input/'
inputfile = 'inputfile.xml'
outputpath = '../Output/'

#arrays of the five different variables for the time windows
var1 = {10000,100000,1000000, 10000000}
var2 = {10000,100000,1000000, 10000000}
var3 = {10000,100000,1000000, 10000000}
var4 = {10000,100000,1000000, 10000000}
var5 = {10000,100000,1000000, 10000000}

#fivefold for-loop which replaces the "VariableWindowValue1-5" values with the
    time windows
#output filename is set to the sizes of the windows
for v1 in var1:
   for v2 in var2:
      for v3 in var3:
         for v4 in var4:
            for v5 in var5:
               with open (outputpath + str(v1) + str(v2) + str(v3) + str(v4) +
                   str(v5) + '.xml','w') as newFile:
                  with open(inputpath + inputfile) as file:
                     for line in file:
                        line = str(line).replace("VariableWindowValue1", str(v1
                            ))
                        line = str(line).replace("VariableWindowValue2", str(v2
                            ))
                        line = str(line).replace("VariableWindowValue3", str(v3
                            ))
                        line = str(line).replace("VariableWindowValue4", str(v4
                            ))
                        line = str(line).replace("VariableWindowValue5", str(v5
                            ))
                        newFile.write(line)
```

Listing C.3: Skript, um die generierten XML-Konfigurationsdateien auf dem Kraken-Cluster auszuführen.

```
#!/bin/bash
#create a new directory with the current date as name in the output folder
time_stamp=$(date +%Y%m%d)
cd ../output/
mkdir -p $time_stamp

#definition of the input folder with the XML configuration files
input= ../input/*

#for every file run a job on the Kraken cluster
for i in $input
do
torque_script="#!/bin/bash
#PBS -N experiment1
#PBS -l nodes=1:ppn=4
#PBS -l walltime=00:30:59,mem=2000mb
#PBS -j oe
#PBS -o ../log/`basename $i`.out
#PBS -e ../log/`basename $i`.err
#PBS -V

cd ../program/
#java command to start the job, it writes the standard output to a file in the
    output directory
java -Xmx2048m -cp "cglib-nodep-2.2.3.jar:antlr-runtime-3.2.jar:org.apache.
    commons.logging.jar:esper-4.9.0.jar:streams-esper.jar:log4j-1.2.17.jar:
    stream-runtime-0.9.10.jar:stream-api-0.9.10.jar:stream-core-0.9.10.jar:
    slf4j-log4j12-1.7.5.jar:slf4j-api-1.7.5.jar" stream.run $i > ../output/
    $time_stamp/`basename $i`
"
echo "$torque_script" | qsub


done
```

C.2 Auswertungsskript

Das Python-Skript, welches den Output von Streams-Esper auswertet ist auf der angefügten CD zu finden (evaluation.py). Es berechnet folgende Werte:

- Dauer der Ausführung (Antwortzeit)
- Dauer, die Streams-Esper braucht, um die Prozessoren zu starten
- Dauer, die Streams-Esper braucht, um die Prozessoren zu beenden
- Dauer zwischen dem Start des letzten Prozessors und dem ersten Resultat
- Dauer zwischen dem letzten Resultat und dem ersten beenden eines Prozessors
- Durchsatz: verarbeitete Tupel pro Sekunden
- Ausgegebene Resultate pro Sekunde
- Anzahl korrekte Antworten des fünften Joins
- Anzahl inkorrekter und fehlender Antworten des fünften Joins
- Trefferquote: Prozentsatz korrekter Antworten
- Anzahl ausgegebener Antworten jedes Joins
- Ausführungszeit jedes Joins
- Ausführungszeit für jeweils 10% der Antworten.

C.3 Weitere Statistiken zu den Resultaten

C.3.1 Ausführungszeiten

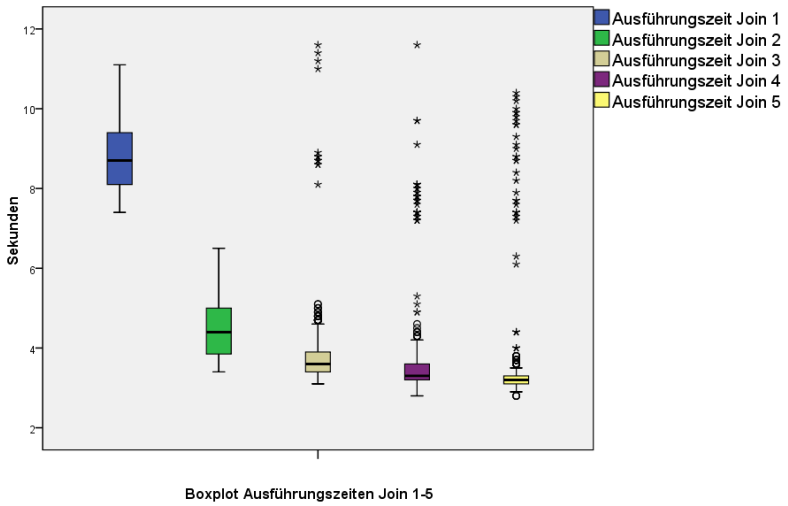


Abbildung C.1: Boxplots der Ausführungszeiten für die Joins 1-5.

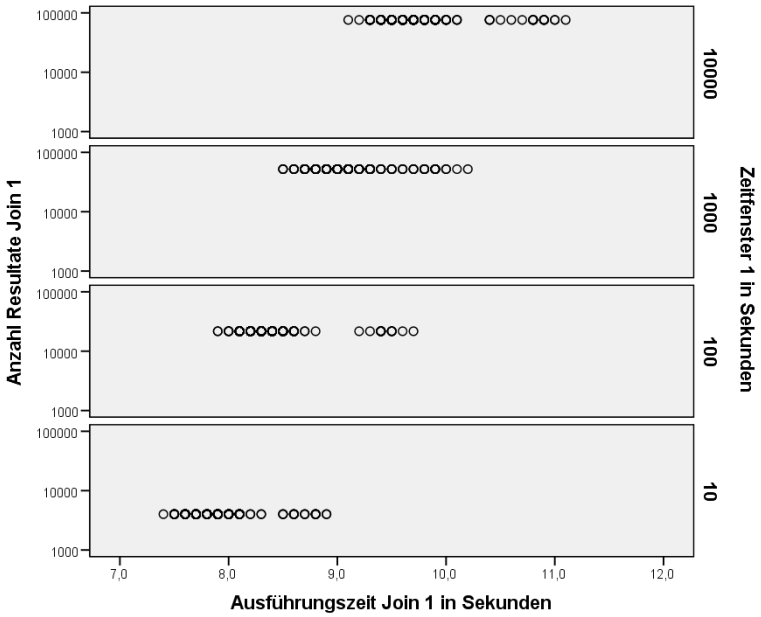


Abbildung C.2: Streudiagramm: Die Ausführungszeit des ersten Joins im Vergleich zu den Anzahl Resultaten und das Zeitfenster des ersten Joins.

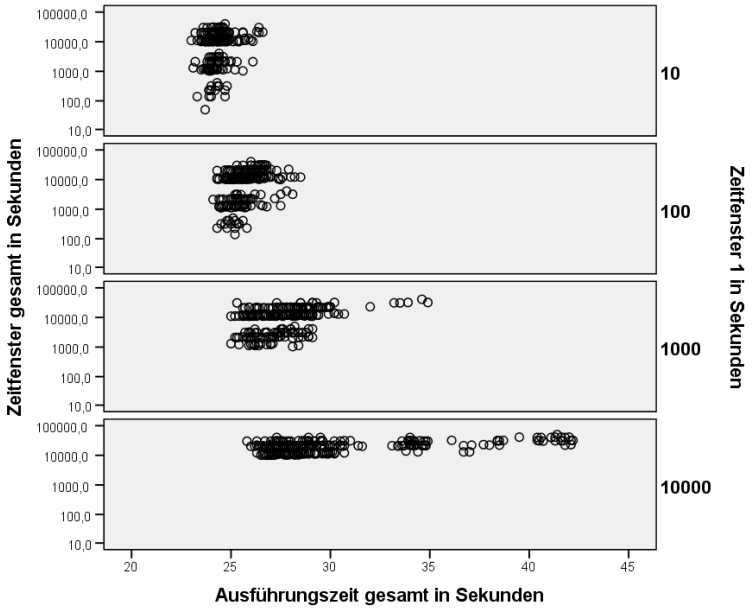


Abbildung C.3: Ausführungszeiten in Systemzeit im Verhältnis zum Zeitfenster insgesamt und zum Zeitfenster des ersten Joins.

C.3.2 Durchsatz

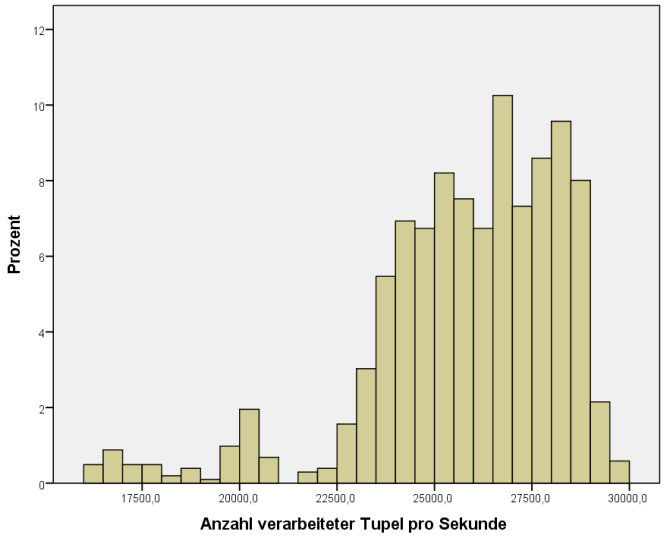


Abbildung C.4: Verteilung der verarbeiteten Tupel/Sekunde (Durchsatz).

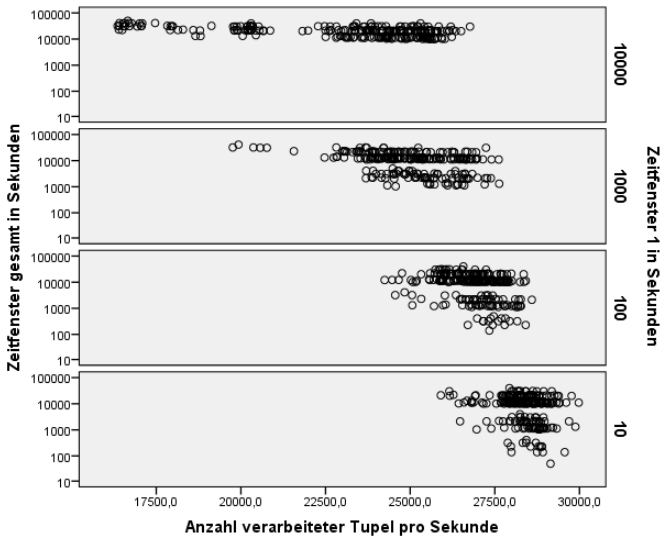


Abbildung C.5: Verarbeitete Tupel/Sekunde im Vergleich zum gesamten Zeitfenster und zum Zeitfenster des ersten Joins. Je höher die Zeitfenster insgesamt und das des ersten Joins, desto weniger Tupel pro Sekunde kann das System tendenziell verarbeiten.

C.3.3 Ausgegebene Resultate und Trefferquote

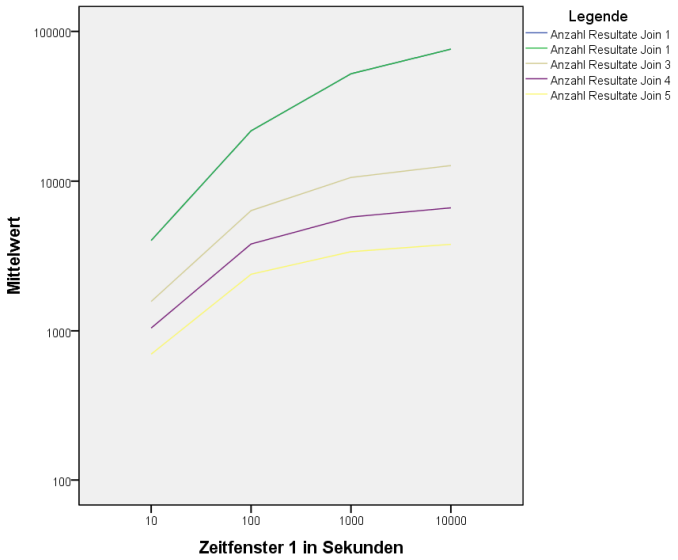


Abbildung C.6: Mittelwerte der Anzahl ausgegebener Resultate im Vergleich zu den Zeitfenstern des ersten Joins.

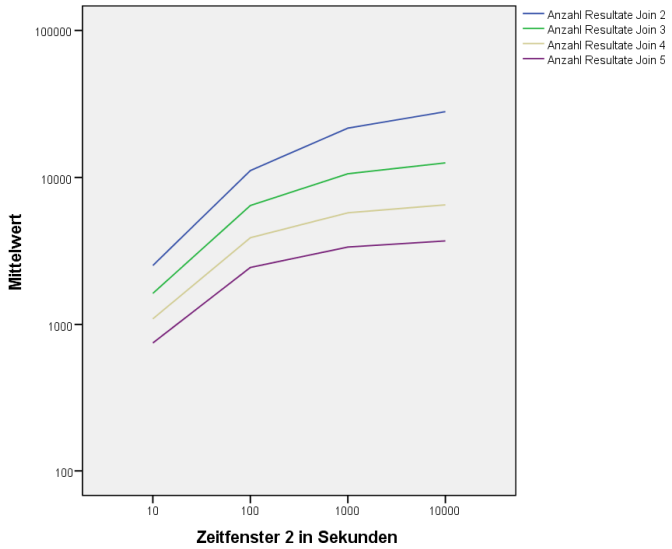


Abbildung C.7: Mittelwerte der Anzahl ausgegebener Resultate im Vergleich zu den Zeitfenstern des zweiten Joins.

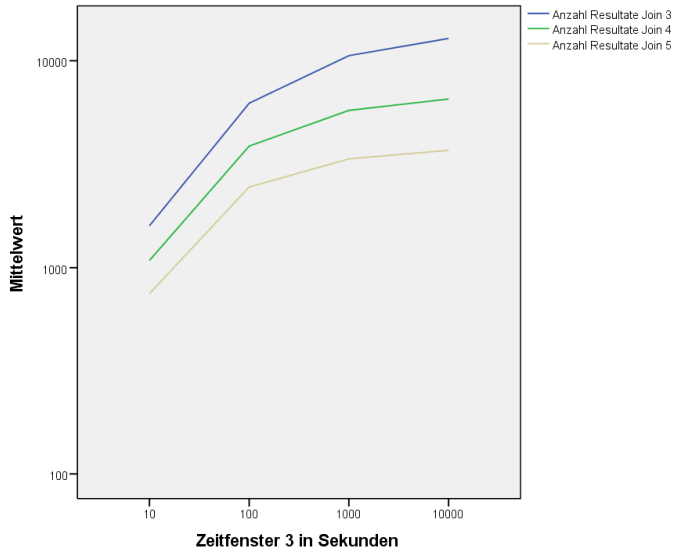


Abbildung C.8: Mittelwerte der Anzahl ausgegebener Resultate im Vergleich zu den Zeitfenstern des dritten Joins.

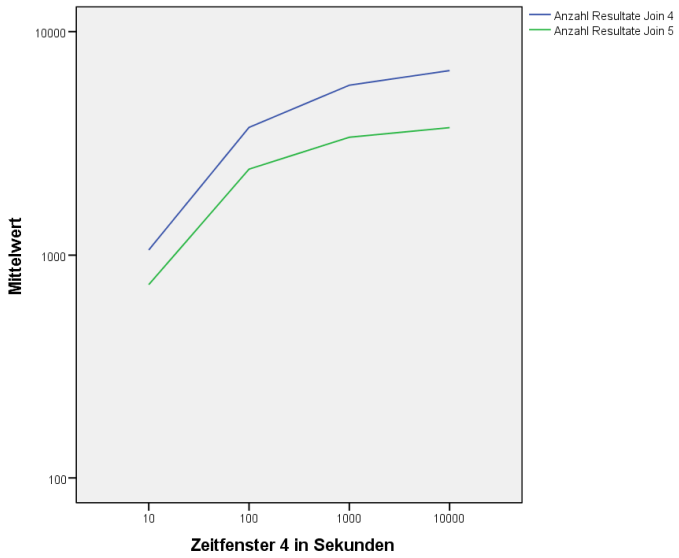


Abbildung C.9: Mittelwerte der Anzahl ausgegebener Resultate im Vergleich zu den Zeitfenstern des vierten Joins.

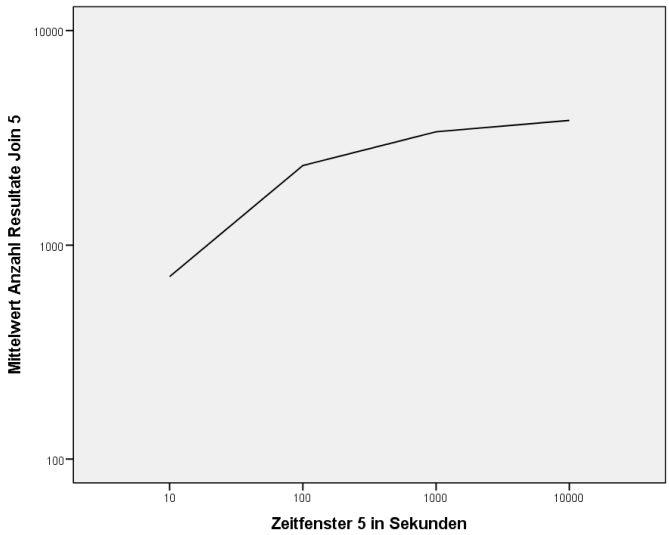


Abbildung C.10: Mittelwerte der Anzahl ausgegebener Resultate im Vergleich zu den Zeitfenstern des fünften Joins.

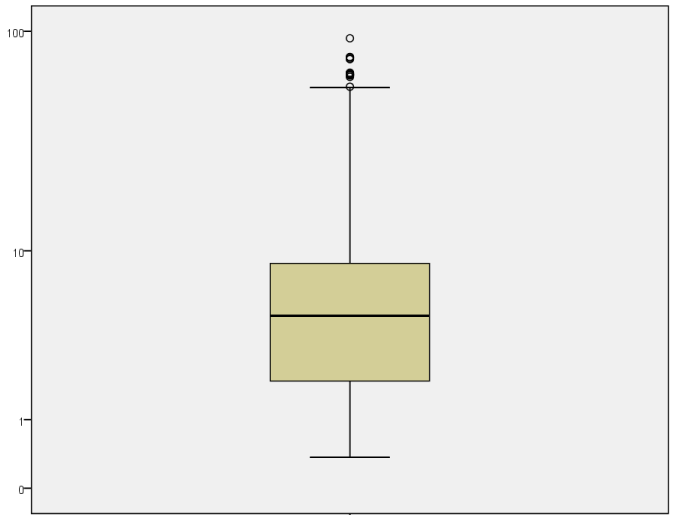


Abbildung C.11: Boxplot: Prozentsatz aller Resultate des fünften Joins (Trefferquote).

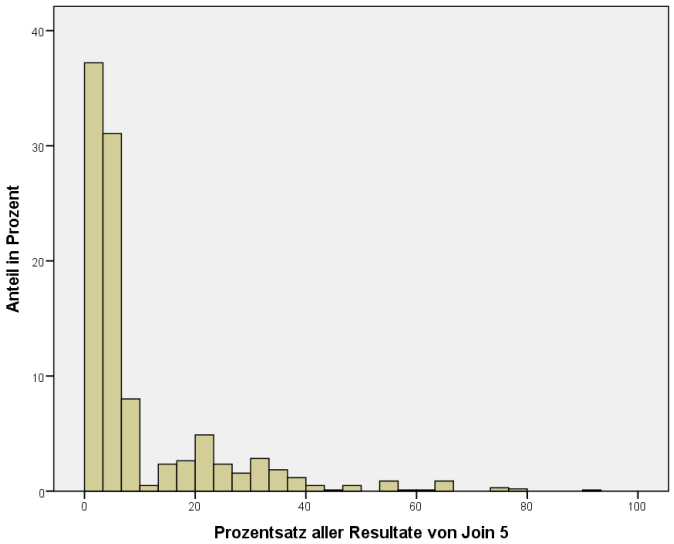


Abbildung C.12: Verteilung: Prozentsatz aller Resultate des fünften Join (Trefferquote).

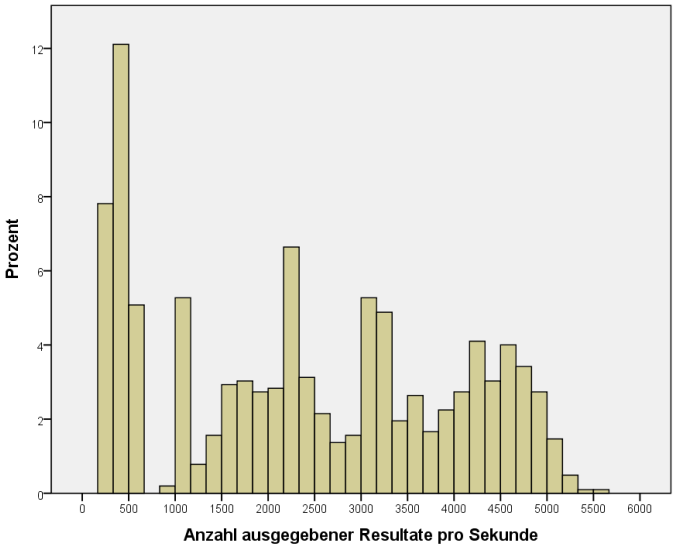


Abbildung C.13: Verteilung der ausgegebenen Resultate/Sekunde.

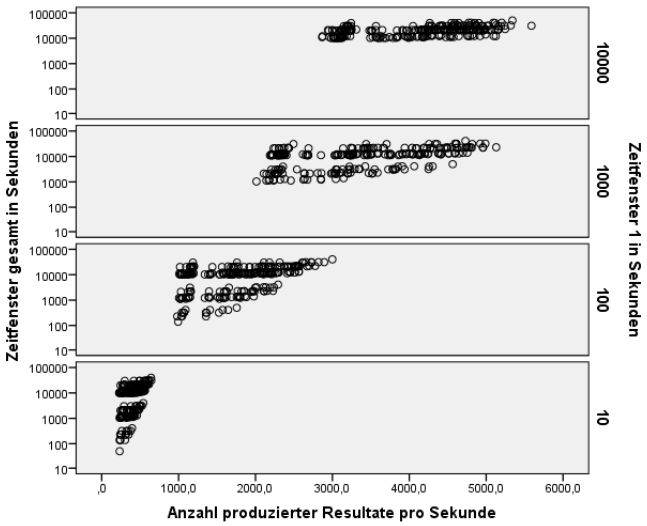


Abbildung C.14: Ausgegebene Resultate/Sekunde im Vergleich zum gesamten Zeitfenster und zum Zeitfenster des ersten Joins. Die Anzahl produzierter Resultate pro Sekunde ist mehrheitlich höher, je grösser das totale Zeitfenster und das des ersten Joins ist.

Literaturverzeichnis

James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker and Richard Tibbetts. Linear Road : A Stream Data Management Benchmark. In *VLDB*, 2004.

Gianpaolo Cugola and Alessandro Margara. Processing flows of information. *ACM Computing Surveys*, 44(3):1–62, 2012.

Danh Le-phuoc, Minh Dao-tran, and Minh-duc Pham. Linked Stream Data Processing Engines : Facts and Figures. In *The Semantic Web - ISWC 2012: 11th International Semantic Web Conference, ISWC 2012, Boston, MA, USA, Nov 12-14, 2012, Proceedings*, volume 1380, pages 1–12, 2012.

Ling Liu and M. Tamer Özsu. *Encyclopedia of database systems*. 2009.

Thomas Scharrenbach, Jacopo Urbani, Alessandro Margara, Emanuele Della Valle and Abraham Bernstein. Seven Commandments for Benchmarking Semantic Flow Processing Systems. In *Proc.ESWC 2013 (to appear)*, number 296126, pages 1–15, 2013.

Ying Zhang, Pham Minh Duc, Oscar Corcho and Jean-paul Calbimonte. SRBench : A Streaming RDF / SPARQL Benchmark. In *The Semantic Web - ISWC 2012: 11th International Semantic Web Conference, ISWC 2012, Boston, MA, USA, Nov 12-14, 2012, Proceedings*, 2012.