Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Birger Kamp

Evaluation von Open Source Data Integration Tools im Kontext von Big Data

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Birger Kamp

Evaluation von Open Source Data Integration Tools im Kontext von Big Data

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 25.04.2018

Birger Kamp

Thema der Arbeit

Evaluation von Open Source Data Integration Tools im Kontext von Big Data

Stichworte

Big Data, ETL, Benchmarking, Talend, Apache Gobblin, Apache Spark, Datenflussverarbeitung

Kurzzusammenfassung

Der Bereich der Datenintegration ist auf Prozesse und Tools angewiesen, die Daten effektiv und effizient verarbeiten. Viele Tools und Frameworks, die für Big Data-Problemstellungen entwickelt werden, eignen sich auch für die Nutzung im Datenintegrationsbereich. In dieser Arbeit werden ein bekanntes ETL-Tool und eine Kombination aus zwei Big Data-Frameworks miteinander verglichen.

Die TPC, ein Non-Profit-Konsortium aus Datenintegrationsherstellern, hat den ETL-Benchmark *TPC-DI* spezifiziert, durch welchen ETL-Tools und -Frameworks miteinander verglichen werden können. Die komplexe Spezifikation wird mit dem Tool *Talend Open Studio for Big Data* und einer Kombination aus den Big Data-Frameworks *Apache Gobblin* und *Apache Spark* implementiert. Einer der Importjobs wird außerdem so modifiziert, dass er einen Event-Stream statt eines Dateisystems als Datenquelle verwendet. Zur Zeit des Schreibens ist diese Arbeit die einzige Veröffentlichung, die den *TPC-DI*-Benchmark mit diesen beiden Messkandidaten und der Streaming-Erweiterung durchführt.

Die Messungen werden in mehr als 50 unterschiedlichen Experimenten durchgeführt. Dadurch werden viele Unterschiede der Implementationen, sowohl in den Importlaufzeiten, als auch in der CPU- und RAM-Nutzung, sichtbar. Die Ergebnisse zeigen, dass die Talend-Implementation mehr CPU-Last auf dem Datenbankserver erzeugt als die Gobblin-Spark-Implementation. Während die Gobblin-Spark-Implementation mehr CPU-Last auf den ausführenden Nodes ausführt als die Talend-Implementation. In den Laufzeiten ist zu sehen, dass bei den zwei kleinsten Importdatensets, die Gobblin-Spark-Implementation die kürzeste Laufzeit erreicht. Im größten gemessenen Datenset hingegen erzielt Talend die geringere Laufzeit.

Birger Kamp

Title of the paper

Evaluation of Open Source Data Integration Tools in Context of Big Data

Keywords

Big Data, ETL, Benchmarking, Talend, Apache Gobblin, Apache Spark, Streamprocessing

Abstract

The field of data integration relies on processes and tools that process data effectively and efficiently. Many tools and frameworks that are developed for big data problems are also suitable for use in data integration. This paper compares a well-known ETL tool and a combination of two big data frameworks.

The TPC, a non-profit consortium of data integration vendors, has specified the ETL benchmark *TPC-DI*, which compares ETL tools and frameworks. The extensive specification is implemented with the tool *Talend Open Studio for Big Data* and a combination of the Big Data frameworks *Apache Gobblin* and *Apache Spark*. One of the import jobs is also modified to use an event stream instead of a file system as the data source. At the time of writing, this work is the only publication that performs the *TPC-DI* benchmark with these two candidate candidates and a streaming extension.

The measurements are carried out in more than 50 different experiments. As a result, many differences in the implementation, both in the import runtime, as well as in the CPU and RAM usage, will be visible. The results show that the Talend implementation generates more CPU load on the database server than the Gobblin Spark implementation. While the Gobblin Spark implementation runs more CPU on the executing nodes than the Talend implementation. In terms of runtime, the Gobblin Spark implementation achieves the shortest runtime for the two smallest import data sets. By contrast, Talend achieves the lower running time in the largest measured data set.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einführung

Es werden heutzutage bei fast jeder Gelegenheit Daten erfasst. Bei jedem Besuch eines Webshops wird das Nutzerverhalten getrackt, jede Nutzung des Smartphones wird beim Hersteller registriert und Umweltdaten werden in *Smart Cities* gesammelt. Laut IBM (2013) wurden schon 2013 jeden Tag mehr als 2,5 Exabyte Daten pro Tag generiert, wovon 80% unstrukturiert sind. So große Datenmengen sind ohne Big Data-Technologien nicht in der nötigen Zeit analysierbar.

Im Bereich der Datenintegration ist man auf Prozesse und Tools angewiesen, die Daten effektiv und effizient verarbeiten. Große Hersteller wie *IBM*, *Oracle*, *Microsoft*, *SAP*, *Informatica* und *Cisco* konkurrieren auf dem Markt der Datenintegrationslösungen. Die Produkte sollen sich gegenseitig im Feature-Umfang und in der Performance überbieten.

Es gibt OpenSource-Alternativen zu den kommerziellen Datenintegrations-Tools, die noch zu wenig untersucht wurden. Entweder, weil sie ungeeignet scheinen oder, weil sie für den Einsatz im Datenintegrationsbereich nicht ausreichend bekannt sind. Dabei gibt es OpenSource-Tools und -Frameworks aus dem BigData-Bereich, die dieselben Probleme lösen, wie ein kommerzielles Datenintegrationsprodukt. Apache Toplevel-Projekte werden durch ihre Community zu performanten und stabilen Systemen entwickelt, die mehr und mehr Einzug in Unternehmen halten.

1.1. ETL und ELT

Nach Lenzerini (2002) ist Datenintegration *das Problem, Daten von unterschiedlichen Quellen zu konsolidieren und dem Nutzer eine einheitliche Sicht auf diese Daten zu bieten.* Im Bereich der *Business Intelligence* werden nach Dayal u. a. (2009) *Data Warehouses* verwendet, um große relationale Datenmengen in einer *einheitlichen Sicht für den Nutzer* sowie für Analysezwecke bereitzustellen. Ein Data Warehouse konsolidiert Daten aus mehreren operationalen Datenbanken. Mithilfe von Datenintegrationsprozessen (auch Extract-Transform-Load (ETL)-Prozesse genannt) wird ein Data Warehouse mit Daten befüllt. ETL-Prozesse umfassen das *Extrahieren*, das *Transformieren* und das *Laden* von Daten.

Während der *Extraktion* werden Daten, die anschließend in das Data Warehouse importiert werden sollen, aus dem Quellsystem exportiert. Nach Vassiliadis und Simitsis (2009) ist dieser

Schritt der Einfachste des ETL-Prozesses. Es gilt dennoch zu beachten, dass während des Exports eine erhöhte Last auf dem Quellsystem vorliegen kann, was dessen Nutzung eventuell beeinträchtigt. Es gibt außerdem mehrere Exportvarianten. Je nach dem, welche verwendet wird, werden unterschiedliche Daten aus dem Quellsystem exportiert. Die einfachste Variante ist, dass der vollständige Datenstand exportiert wird. Dann muss im *Transformationsschritt* entschieden werden, welche Daten bereits vorhanden waren, neu, verändert oder gelöscht wurden. Ein komplexerer und deutlich aufwändigerer Ansatz ist, dass der ETL-Prozess das Log-File des Quellsystems parst und so herausfindet, welche Datensätze verändert wurden.

Im Schritt der *Transformation* geht man davon aus, dass die extrahierten Daten in einer *Data Staging Area* abgelegt wurden, von wo aus die Daten geladen werden können. Nach Vassiliadis und Simitsis (2009) enthält der Transformationsschritt die Kernfunktionalität eines ETL-Prozesses. Die Implementation und Performance dieses Schritts hängt daher stark vom Zweck der Applikation ab. Vassiliadis und Simitsis (2009) haben die Verarbeitungsprobleme, die in diesem Schritt auftreten können, wie folgt kategorisiert:

***Schema-Level* Probleme:** Benennungs- und Strukturkonflikte

***Record-Level* Probleme:** Duplizierte oder sich widersprechende Einträge und Konsistenzprobleme

***Value-Level* Probleme:** Technische Probleme, wie unterschiedliche Datentypen oder unterschiedliche Interpretationen der Daten

Um diese Probleme zu bearbeiten, werden im Transformationsschritt häufig folgende Funktionalitäten genutzt:

- Normalisierung und Denormalisierung der Daten
- Zusammenführen von Daten aus mehreren Quellen
- Verändern der Datenstruktur
- Filtern von Daten

Im *Loading*-Schritt werden nach Vassiliadis und Simitsis (2009) die transformierten Daten in die Datensenke geschrieben.

Eine Variante eines ETL-Prozesses ist der Extract-Load-Transform (ELT)-Prozess. Die Schritte sind dieselben, wie bei einem ETL-Prozess, aber sie finden in einer anderen Reihenfolge statt.

Die Daten werden direkt nach der Extraktion in eine Datenbank geschrieben. Die Transformationsschritte werden anschließend in SQL formuliert und durch die Datenbank ausgeführt. Ein ELT-Prozess bietet sich nach Vassiliadis und Simitsis (2009) beispielsweise dann an, wenn der Datenbankserver hardwareseitig entsprechend rechenstark ausgestattet ist.

1.2. Datenintegration mit BigData-Technologien

Es gibt den Trend in der Informatik, dass immer mehr Daten erzeugt, gespeichert und analysiert werden. Dieser Trend wird als *BigData* bezeichnet und durch die VVV-Definition (Sagiroglu und Sinanc (2013)) beschrieben. Die V's stehen für *Variety*, *Velocity* und *Volume*. BigData setzt sich mit Daten auseinander, die unterschiedliche Strukturen aufweisen (*Variety*), die in hoher Rate erzeugt oder verarbeitet werden müssen (*Velocity*) und Daten, die in großen Mengen auftreten (*Volume*). Zur Bewältigung von BigData-Themen wurden Frameworks und Tools entwickelt, die auf diese Probleme eingehen. Für unstrukturierte Daten haben sich NoSQL-Datenbanken etabliert, die schemalos sein können, anders als relationale Datenbanken. Für Daten, deren Ergebnis mit möglichst wenig Latenz (*Fast Echtzeit*) benötigt wird, wurden Streaming-Technologien entwickelt, wie zum Beispiel *Apache Storm*, *Apache Flink* und *Apache Kafka*. Für große Datenmengen, die nicht auf einem einzelnen Computer verarbeitet werden können, wurden Techniken zur verteilten Verarbeitung ausgebaut. Viele der BigData-Techniken existierten schon vor dem BigData-Trend. Das Besondere an dieser Entwicklung ist allerdings, dass BigData-Technologien auf *Commodity*-Hardware genutzt werden können und nicht zwangsläufig teure und spezialisierte Hardware benötigt wird.

BigData-Technologien befassen sich nach Dong und Srivastava (2013) und Chen u. a. (2013) mit denselben Problemen, wie auch Datenintegrationsapplikationen. Beide Problembereiche befassen sich mit unterschiedlich strukturierten Daten, die zusammengeführt werden müssen und die (abhängig von der Anwendung) in großen Mengen auftreten können.

Das *Velocity*-Problem beschäftigt beispielsweise auch die Entwickler von Datenintegrationsprozessen. Vassiliadis und Simitsis (2009) stellen die Probleme, Hürden und Vorteile von *Fast Echtzeit* ETL-Prozessen vor. Das Problem der *Fast Echtzeit*-Verarbeitung beschäftigt auch die BigData-Entwickler. Datenintegrations- und BigData-Applikationen scheinen also inhaltlich dicht beieinander zu liegen. Der Hersteller für Datenintegrationslösungen *Talend* bezeichnet die Kombination aus den beiden Bereichen BigData und Datenintegration als *Big Data Integration* (Talend (2018c)), die Erstellung von ETL-Prozessen mithilfe von BigData-Technologien.

1.3. ETL-Benchmarking

Es gibt ETL-Lösungen, die gemeinsam mit großen Datenbanksystemen angeboten werden. Daneben gibt es Hersteller, die sich auf solche ETL-Tools konzentrieren, die ein breites Spektrum von Systemen unterstützen. Der Markt der ETL-Tools ist laut Simitsis u. a. (2009) umkämpft und mehrere Millionen Dollar wert. ETL-Tools werden häufig anstelle von selbstentwickelten Anwendungen, die auf den jeweiligen Anwendungszweck spezialisiert sind, eingesetzt. Kunden versprechend sich mit den Tools eine höhere Entwicklungseffizienz und bessere Wartbarkeit.

Die ETL-Tools unterscheiden sich kaum durch Features und versuchen sich daher in der Effizienz voneinander abzuheben. Die Datenintegrationshersteller überbieten sich gegenseitig mit erreichten Performance-Weltrekorden. Wyatt u. a. (2009) haben eine Auflistung erreichter Weltrekorde zwischen 2001 und 2008 aufgeführt. Auszugsweise hier die jüngsten beiden Einträge der Liste:

- Microsoft hat laut Wyatt (2008) 1TB Daten des TPC-H Benchmarks in unter 30 Minuten verarbeitet. Sie beschreiben, welche Hardware und Implementation sie dafür verwendet haben.
- Das Original des Pressberichts ist leider nicht mehr aufrufbar, aber Ohlden (2008) schreibt über den Weltrekord der Hersteller *Syncsort*, *Vertica* und *HP*. Sie haben gemeinsam einen Benchmark entwickelt und durchgeführt, der 5,4TB Daten in circa 57 Minuten in eine Datenbank geladen hat.

Wyatt u. a. (2009) zählen außerdem noch weitere Weltrekorde auf, die zeigen, dass ETL-Hersteller versuchen, den Rekord des Anderen zu überbieten. Dazu wurde in vielen Weltrekorden ein eigener Benchmark entwickelt und verwendet, den andere Hersteller nicht verwenden. Dadurch sind die Ergebnisse der Benchmarks nicht oder nur schwierig miteinander zu vergleichen. Die beiden oben genannten Weltrekorde sind beispielhaft für die Unvergleichbarkeit. Die Ergebnisse eines Benchmarks hängen von den Quelldaten, der Transformationen, der Datensenke und nicht zuletzt von der genutzten Hardware ab. Gäbe es bei ETL-Tools, ähnlich wie bei relationalen Datenbanken, ein standardisiertes Beschreibungsmodell, dann wären unterschiedliche Benchmarkspezifikationen laut Wyatt u. a. (2009) einfacher miteinander zu vergleichen.

Wyatt u. a. (2009) und Simitsis u. a. (2009) haben sich damit auseinandergesetzt, wie ein ETL-Benchmark gestaltet sein muss, damit er ETL-Tools miteinander vergleichbar macht. Die ETL-Tools der einzelnen Hersteller unterscheiden sich im Design-Ansatz, wie der Prozess entwickelt wird, in der Programmiersprache und auch in der Anzahl der möglichen Transformationen.

Daher sollte ein ETL-Benchmark Basisfunktionalitäten verwenden, die vermutlich jedes ETL-Tool unterstützt oder die mit wenig Aufwand nachgebildet werden können. Es ist weiterhin nicht die Aufgabe eines Performancebenchmarks, *jede* Funktionalität eines ETL-Tools zu verwenden und einen Leistungstest dafür durchzuführen.

Nach Simitsis u. a. (2009) ist es nicht wichtig, ob der Datenintegrationsprozess ein ETL- oder ein ELT-Prozess ist. In einem abschließenden Benchmarkbericht wird die jeweilige Implementation erläutert, sodass für den Leser nachvollziehbar ist, inwieweit die ETL- oder ELT-Implementation auf den eigenen Anwendungsfall übertragbar ist. Es muss außerdem im Benchmark spezifiziert werden, ob es sich um eine Batch- oder eine Streamverarbeitung handelt, davon abhängig verändert sich der Verarbeitungsfluss. Ein Bechmark wird zudem für den Leser und Entwickler verständlicher, wenn er sich innerhalb einer fachlichen Domäne in einem definierten Szenario befindet. Zum Beispiel kann ein Benchmark die Datenverarbeitung eines HR-Systems spezifizieren, das Daten aus anderen Systemen wie einem Bewerbermanagement oder staatlichen Stellen bezieht.

Die Transaction Processing Performance Council (TPC) ist eine Non-Profit Vereinigung aus vielen Herstellern aus dem Datenintegrations- und Datenbankmarkt. Sie definieren laut Transaction Processing Performance Council (2018a) Benchmarkspezifikationen für Transaktionsverarbeitung und Datenbanken. Es gibt bereits die Benchmark-Spezifikationen *TPC-H* und *TPC-E*, mit denen man Benchmarks für OLTP-Datenbanken durchführen kann. In diesen Spezifikationen werden ETL-Prozesse definiert, die Daten in ein aufwändiges Data Warehouse laden. Laut Simitsis u. a. (2009) sind die *TPC-H* und *TPC-E* aber nicht für ETL-Benchmarks geeignet, denn sie legen zu wenig Fokus auf die ETL-Prozesse und zu viel Fokus auf die Datenbank als Datensenke. Die Benchmarks haben sich beispielsweise damit beschäftigt, wie konkurrierende Updates von Datensätzen in einem Data Warehouse gehandhabt werden. Das ist für einen Datenkbank-Benchmark sicherlich sinnvoll, es steht aber nicht zwangsläufig im Fokus eines ETL-Benchmark.

Es fehlte ein standardisierter ETL-Benchmark. Daher wurde laut Wyatt u. a. (2009) im Jahr 2008 ein weiteres TPC-Komitee gebildet, das einen entsprechenden ETL-Benchmark erstellen sollte. Poess u. a. (2014) haben über einen ersten Entwurf des neuen TPC Data Integration (TPC-DI) publiziert. Sie beschreiben welche Datenquellen, Transformationen und Datensenken die Spezifikation verwendet und wie diese Komponenten zusammenhängen. Inzwischen ist die TPC-DI Spezifikation in Version 1.1.0 vorhanden und wurde von wenigen wissenschaftlichen Arbeiten implementiert, die in Abschnitt 1.5 aufgelistet werden. Die TPC-DI ist zum Zeitpunkt dieser Arbeit die einzige vorhandene Benchmarkspezifikation für ETL-Prozesse und daher die Spezifikationsgrundlage dieser Arbeit.

1.4. Motivation

Der Markt der Datenintegrationshersteller wird laut Beyer u. a. (2017) von großen und kommerziellen Lösungen dominiert. In Abbildung 1.1 ist von Beyer u. a. (2017) dargestellt, wie die jeweiligen Datenintegrationsprodukte beurteilt werden. Es sind viele Hersteller aufgeführt, aber es ist kein Apache-Projekt oder ein anderes ETL-Tool abgebildet, das nicht kommerziell verfügbar ist.

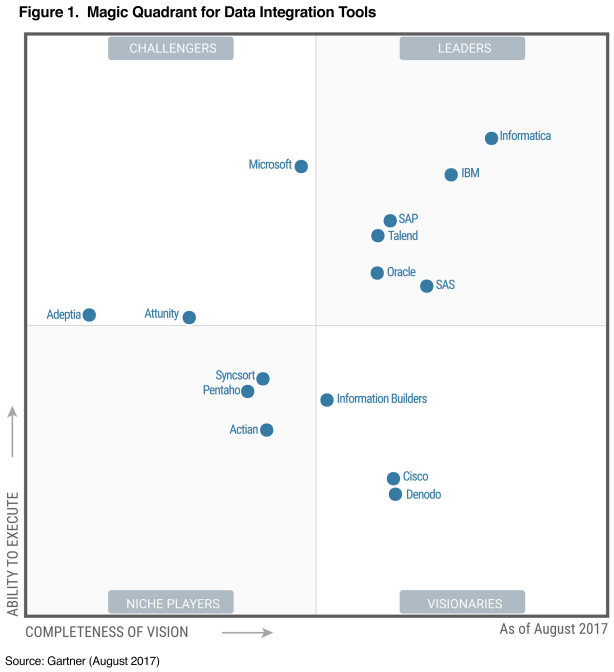

Abbildung 1.1.: Magic Quadrant für Datenintegration-Tools 2017, vgl. Informatica (2017)

Geerdink (2016) hatte auf der Spark Summit EU dazu animiert, weniger traditionelle ETL-Entwicklung zu betreiben. Er zeigte in seiner Präsentation, dass sich das Map-Reduce-Framework *Apache Spark* dazu eignet, um ETL-Prozesse zu implementieren. Es löse viele Probleme, mit denen man im ETL-Bereich zu kämpfen habe. Zu den Vorteilen gehöre zum Beispiel, dass die parallelisierte Datenverarbeitung mit wenig Entwicklungsaufwand möglich sei, der Code sei mit üblichen Unit Tests testbar und sich übliche Monitoring-Frameworks nutzen ließen.

Diese Arbeit wird mit zwei OpenSource-Tools und -Frameworks die TPC-DI Spezifikation implementieren und anschließend eine Performancemessung damit durchführen. Die Messkandidaten werden anhand der Metriken bewertet, die die TPC-DI vorgibt. Es wird außerddem die Nutzung der Hardwareressourcen erfasst, um sie anschließend auszuwerten.

Zusätzlich wird in einem experimentellen Ansatz die TPC-DI Spezifikation mit BigData-Technologien modifiziert. Die modifizierte Implementation wird ebenfalls gemessen und analysiert. Anschließend werden die Spezifikation-konformen und die modifizierten Implementationen miteinander verglichen.

1.5. Verwandte Arbeiten

Zum Zeitpunkt des Schreibens dieser Arbeit gibt es keine offiziellen Messergebnisse, die mit der TPC-DI Spezifikation erzielt wurden. Es gibt zurzeit wenige wissenschaftliche Arbeiten, die ETL-Tools diesem Benchmark unterzogen und mit anderen Benchmarks verglichen haben.

Majchrzak u. a. (2011) haben die ETL-Tools *Apatar OS Data Integration*, *CloverETL Engine*, *Enhydra Octopus*, *Jitterbit Integration Environment*, Kinetic ETL (KETL), *Pentaho Data Integration*, *Scriptella* und *Talend Open Studio* anhand ihrer Features miteinander verglichen. Aus der Liste erfüllten lediglich die Tools *Pentaho* und *Talend* alle der aufgestellten Kriterien. Die Autoren referenzieren auf die Arbeit von Wyatt u. a. (2009), die darauf hinweist, dass es zu der Zeit keinen standardisierten Benchmark gab. Die Autoren hatten daraufhin einen eigenen Benchmark entwickelt, der sich fachlich mit dem Buchhandel beschäftigte. Es wurden Metriken über die Ausführungszeit, die CPU- und RAM-Nutzung erhoben. Das Ergebnis war, dass die Ausführungszeit der Talend-Implementation deutlich geringer ausfiel als die der Pentaho-Implementation. Die CPU-Nutzung der Talend-Implementation war ebenfalls deutlich geringer. Die Autoren vermuteten, dass Talend mehr Teile der Verarbeitung auf den Datenbankserver überträgt als es Pentaho tut.

Bleuel (2016) implementierte in seiner Masterthesis für das Tool *Pentaho* die TPC-DI Spezifikation. Er führte die Messungen auf einem Computer aus, der über einen AMD A10 Quad core APU, 8GB RAM und eine 500GB SSHD verfügte. Auf diesem Computer lief sowohl der Messkandidat als auch die Zieldatenbank. In der Messung verwendete der Autor die Metriken der Spezifikation und erreichte folgendes Ergebnis: Das Benchmarksystem erzielte einen Durchsatz von 13 Zeilen pro Sekunde und Kosten von 61,54 € pro Zeile pro Sekunde. Dieses erreichte Ergebnis wird später mit der Messung aus der vorliegenden Arbeit verglichen.

Qualitz (2013) hat in seiner Masterthesis zwei ETL-Tools miteinander verglichen. Die Thesis ist im Volltext nicht zu finden, daher gibt es keine konkreten Informationen über die Messkandidaten und die Spezifikation des Benchmarks. In einem verfügbaren Exposee kündigt Qualitz an, dass die Messkandidaten „für den akademischen Betrieb kostenfrei verfügbar sein müssen“. Seine Arbeit soll unterschiedliche ETL-Tools miteinander vergleichen und anschließend zwei Messkandidaten bestimmen. Die beiden Messkandidaten sollen dann anhand eines Benchmarks

miteinander verglichen werden. In dem Exposee beschreibt Qualitz nicht, welchen Benchmark er verwendet oder ob er eine eigene Spezifikation verwendet. Die Veröffentlichung des TPC-DI in Version 1.0.0 war laut Transaction Processing Performance Council (2014) im Oktober 2013. Die Spezifikation gab es also zu der Zeit als Qualitz an seiner Masterthesis arbeitete. Möglicherweise hat er diese für seine Benchmarks verwendet.

1.6. Gliederung der Arbeit

In Kapitel 2 wird dem Leser ein technisches Grundverständnis der verwendeten Tools und Frameworks vermittelt. Die markanten Punkte der TPC-DI Spezifikation werden in Kapitel 3 erläutert. Kapitel 4 dient der Nachvollziehbarkeit des Versuchsaufbaus und dessen Durchführung. Dort ist außerdem beschrieben, welche Hypothesen mit der Messung beantwortet werden soll und in welchen Punkten der Messaufbau dieser Arbeit von der TPC-DI Spezifikation abweicht.

Die Ergebnisse des Messungen sind in Kapitel 5 festgehalten und mit Abbildungen und Tabellen veranschaulicht. In Kapitel 6 werden die Messergebnisse analysiert, um die Hypothesen zu beantworten. Abschließend ist in Kapitel 7 eine Zusammenfassung dieser Arbeit und ihrer Ergebnisse mit einem Ausblick auf weitere Forschungsfragen.

2. Technische Grundlagen

Diese Arbeit verwendet unterschiedliche Tools und Frameworks, mit denen das Messsystem dieser Arbeit entwickelt wurde. Im Folgenden werden die verwendeten Komponenten grundlegend in ihrer Funktionsweise erläutert, um ein Verständnis für den Einsatz und das Verhalten der Komponenten zu schaffen.

Wie die einzelnen Tools und Frameworks konfiguriert und eingesetzt wurden, ist in Kapitel 4 beschrieben.

2.1. Talend

Talend ist ein Hersteller von Datenintegrationslösungen, der unter anderem Tools im Portfolio hat, mit denen sich ETL-Prozesse entwickeln lassen. Wenn von Talend gesprochen wird, wird im Rahmen dieser Arbeit das Datenintegrationstool gemeint, das der Hersteller anbietet. Es handelt sich dabei um ein GUI-Tool, in dem man mit vorimplementierten Bausteinen ETL-Prozesse zusammenstellen kann. Möchte man den erstellten Prozess ausführen, generiert Talend ohne Zutun des Nutzers Java-Code, der den erstellten Prozess abbildet.

In Abbildung 2.1 ist ein einfacher Talend Beispielprozess nach Talend (2018a) dargestellt. In diesem Prozess werden zwei Dateien zusammengeführt und anschließend in eine MySQL-Datensenke geschrieben. Mithilfe der Komponente *tMap* können die Datenströme aus den beiden Datenquellen zu einem neuen Datenstrom zusammengeführt werden. In Abbildung 2.2 ist die Konfiguration des tMaps aus dem Beispielprozess dargestellt. Es ist zu sehen, dass die beiden Quelldateien anhand der Bedingung $row1.City = row2.City$ zusammengeführt werden. Die gelben Pfeile zwischen den zwei Eingangsströmen und dem Ausgangsstrom zeigen, welche Felder der Quelldateien in die Datensenke geschrieben werden.

Talend bietet mehrere Datenintegrationslösungen an, die sich im Kern alle ähnlich sind. Viele der Produkte sind in einer kostenlosen Open Source Version verfügbar, die weniger Features hat, als die kostenpflichtige Enterprise Variante. So gibt es beispielsweise das *OpenStudio for Data Integration*, mit dem man den Beispielprozess aus Abbildung 2.1 erstellen kann.

Möchte man Talend mit BigData-Technologien kombinieren, dann verwendet man das *OpenStudio for Big Data*. Es erlaubt ETL-Prozesse zu erstellen, die gängige BigData-Technologien

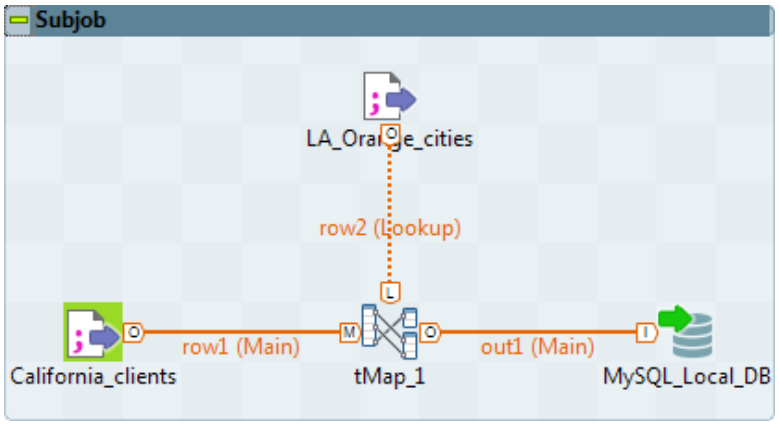

Abbildung 2.1.: Talend Beispielprozess, vgl. Talend (2018a)

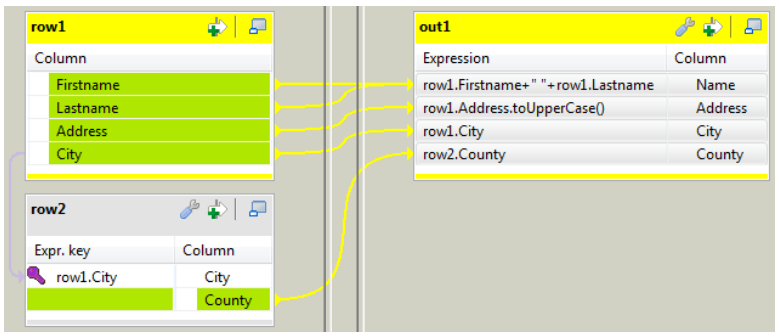

Abbildung 2.2.: Details zum Mapping im Talend Beispielprozess, vgl. Talend (2018a)

wie *HDFS*, *Apache Kafka* oder *MongoDB* unterstützen. In den kostenpflichtigen Varianten dieses Studios können die erstellten Prozesse in verteilten Verarbeitungsframeworks wie *Apache Hadoop* oder *Apache Spark* betrieben werden.

Die kostenlosen Versionen der Datenintegrationsprodukte unterstützen nur den Standalone-Modus, in dem Talend Daten nach dem Paradigma *One-at-a-time* verarbeitet. Laut Marz und Warren (2015) wird in diesem Modus das nächste Objekt einer Quelle erst dann verarbeitet, wenn das vorherige Objekt die gesamte Verarbeitungskette durchlaufen hat.

2.2. Apache Gobblin

Gobblin ist ein Open Source *Data Ingestion*-Framework, das von Qiao u. a. (2015) veröffentlicht wurde. Es wurde anfangs von LinkedIn entwickelt und ist laut Apache-Foundation (2018f) seit Februar 2017 ein Apache Incubator Projekt. Seitdem wird es als *Apache Gobblin* von denselben LinkedIn-Entwicklern und vielen Anderen aus der Open Source Gemeinschaft weiterentwickelt.

Gobblin wirbt damit, dass es sich von anderen *Data Ingestion*-Frameworks unterscheidet, indem es einen generischen Ansatz verfolgt. Es kann mit beliebigen Datenquellen und -senken verwendet werden und die gelesenen Daten beliebig formatiert in die Datensenke schreiben. Unterstützt wird dieser generische Ansatz von Gobblins Architektur, die in Abbildung 2.3 dargestellt ist.

Den Job, den Gobblin ausführen soll, beschreibt man als Entwickler mit den Komponenten *Source*, *Extractor*, *Converter*, *Quality Checker*, *Fork Operator*, *Writer* und *Data Publisher*. Die *Source* ist dafür zuständig, die Daten aus der Datenquelle zu laden. Wenn die Datenquelle beispielsweise ein HDFS-Cluster ist, dann muss die *Source*-Komponente die Verbindung zum Cluster herstellen, die gewünschten Quelldateien auslesen und sie für die weiteren Verarbeitungsschritte bereitstellen. Die Source ist außerdem dafür zuständig, die Quelldaten auf *WorkUnits* aufzuteilen, durch die die parallele Verarbeitung der Daten realisiert wird. Würde die Source-Komponenten beispielsweise aus dem HDFS-Cluster eine CSV-Quelldatei mit 15.000 Zeilen einlesen, könnte sie drei WorkUnits mit je 5.000 Zeilen erstellen.

Der *Extractor* parst die Quelldateien zu Objekten, die weiterverarbeitet werden können. Handelt es sich bei den Quelldateien beispielsweise um CSV-Dateien, dann ist es die Aufgabe des Extractors die Felder einer Zeile auszulesen und sie in verarbeitbare Objekte umzuwandeln. Der *Converter* konvertiert die extrahierten Objekte anschließend in das Ziel-Format. Möchte man beispielsweise CSV-Objekte in Avro-Dateien schreiben, ist es die Aufgabe des Converters jede CSV-Zeile in eine Avro-Zeile mit dem passenden Schema umzuformatieren.

Der *Quality Checker* ist eine optionale Komponente. Sie kann dafür eingesetzt werden, Datensätze herauszufiltern, die gegen vordefinierte Regeln verstoßen. Der *Fork Operator* ist ebenfalls eine optionale Komponente, mit der Datensätze in mehrere Datensenken mit unterschiedlichen Zielformaten geschrieben werden können. Der *Writer* schreibt die Daten anschließend in die Datensenke. Ist die Datensenke beispielsweise ein HDFS-Cluster, dann legt der Writer die Dateien in einem temporären Ordner an und schreibt die Inhalte dort hinein. Sobald alle *WorkUnits* ihre Aufgaben erledigt haben, führt der *Data Publisher* einen Commit durch. Im Beispiel des temporären HDFS-Ordners bedeutet der Commit, dass die Inhalte aus dem temporären Ordner in den Zielordner geschrieben werden und der temporäre Ordner gelöscht wird. In anderen Datensenken, wie zum Beispiel einem *Apache Kafka* Broker, ist der Commit eines Data Publishers technisch nicht möglich. Bei solchen Systemen ist das Ausführen des Writers gleichbedeutend mit dem Commit des Data Publishers.

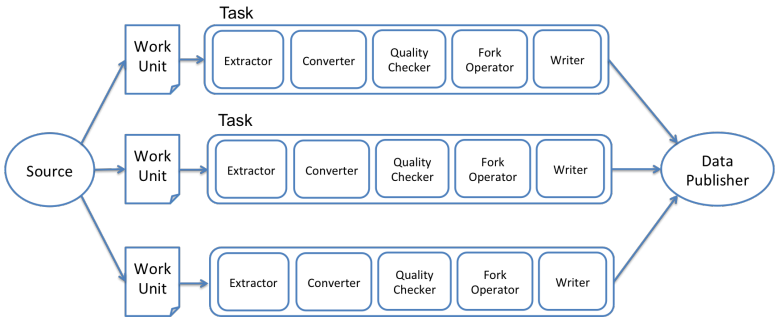

Abbildung 2.3.: Grundarchitektur von Apache Gobblin, vgl. Apache-Foundation (2018b)

Jede der Architekturkomponenten kann durch den Entwickler bereitgestellt werden. Dadurch können beliebige Datenquellen, Datenkonvertierungen und Datensenken in einem *Data Ingestion*-Job implementiert werden.

Die *Work Units* sind unabhängig voneinander und erlauben damit, dass ein Job verteilt ausgeführt werden kann. Laut Qiao u. a. (2015) kann ein Gobblin-Job auf einem einzelnen Node parallel ausgeführt werden oder die *Work Units* werden in einem Hadoop- oder YARN-Cluster deployt. Mit dieser Aufteilung der Verarbeitung kann ein Gobblin-Job - mit etwas Implementationsaufwand - auch in anderen verteilten Systemen wie *Apache Spark* oder *Apache Flink* ausgeführt werden.

Gobblin-Jobs werden durch Properties-Dateien konfiguriert. Die Gobblin-Entwickler haben einen Beispieljob zum Import von Wikipedia-Artikeln in einen HDFS-Cluster bereitgestellt, der in Listing 1 abgebildet ist. Für jede der Job-Komponenten (*Source,Extractor,...*) wird eine Java-Klasse und einige Konfigurationsparameter angegeben. Für die *Source* wird beispielsweise

```
job.name=PullFromWikipedia
job.description=A getting started example for Gobblin

source.class=org.apache.gobblin.example.wikipedia.Wikipedi
 ↪ aSource
source.page.titles=LinkedIn,Wikipedia:Sandbox
source.revisions.cnt=5

wikipedia.api.rooturl=https://en.wikipedia.org/w/api.php
wikipedia.avro.schema={"namespace":
 ↪ "example.wikipedia.avro","type": "record","name":
 ↪ "WikipediaArticle","fields": [{"name": "revid", "type":
 ↪ ["double", "null"]},{"name": "pageid", "type":
 ↪ ["double", "null"]},{"name": "title", "type":
 ↪ ["string", "null"]},{"name": "user", "type": ["string",
 ↪ "null"]},...,{"name": "size",  "type": ["double",
 ↪ "null"]},{"name": "contentformat",  "type": ["string",
 ↪ "null"]},{"name": "contentmodel",  "type": ["string",
 ↪ "null"]},{"name": "content", "type": ["string",
 ↪ "null"]}]}
gobblin.wikipediaSource.maxRevisionsPerPage=10
converter.classes=org.apache.gobblin.example.wikipedia.Wik
 ↪ ipediaConverter
extract.namespace=org.apache.gobblin.example.wikipedia

writer.destination.type=HDFS
writer.output.format=AVRO
writer.partitioner.class=org.apache.gobblin.example.wikipe
 ↪ dia.WikipediaPartitioner

data.publisher.type=org.apache.gobblin.publisher.BaseDataP
 ↪ ublisher
```

Listing 1: Gobblin Beispielkonfiguration

die Klasse *WikipediaSource* angegeben, die die Inhalte der Wikipedia-Seiten parst und zur weiteren Verarbeitung bereitstellt (siehe Apache-Foundation (2018c)). Die Source wird mit den Parametern *source.page.titles* und *source.revisions.cnt* konfiguriert. Das Listing zeigt auch die Writer-Konfiguration, die dafür sorgt, dass die ausgelesenen Wikipedia-Artikel als Avro-Dateien in einen HDFS-Cluster geschrieben werden.

2.3. Apache Spark

Spark ist ein OpenSource-Framework zum verteilten Verarbeiten von Map-Reduce-ähnlichen Anwendungen. Es wurde von Zaharia u. a. (2010) veröffentlicht und an der *Berkeley University* entwickelt. Damals bekam es nicht zuletzt rasch Aufmerksamkeit, weil es bewiesen hat, dass es im Vergleich zum damaligen Map-Reduce-Vorreiter *Apache Hadoop* bis zu zehn mal schneller sein kann. 2014 wurde Spark zu einem Top-Level-Projekt der Apache-Foundation (Apache-Foundation (2018h)).

Spark besteht aus einer erweiterten Master-Slave-Architektur, die in Abbildung 2.4 dargestellt ist. Apache Spark kann in mehreren Varianten deployt werden: Standalone, *Apache Mesos*, *Hadoop YARN* oder *Kubernetes*. Abhängig von dem gewählten Deployment, wird der Spark-Master anders in den Cluster implementiert. Die Slave-Nodes werden auch *Worker*-Nodes genannt und kommunizieren mit dem Spark-Master. Zusätzlich gibt es einen *Driver*-Node, der über die Spark-Applikation verfügt. Dieser Driver-Node kommuniziert mit dem Master-Node, um Hardware-Ressourcen im Cluster zu reservieren. Die reservierten Ressourcen werden als *Executor* bezeichnet. Abhängig von der Hardware-Ressourcen eines Worker-Nodes, können mehrere Executors auf ihm vorhanden sein. Der Driver kommuniziert direkt mit den Executors, um die Map-Reduce-Tasks an diese zu verteilen und ausführen zu lassen.

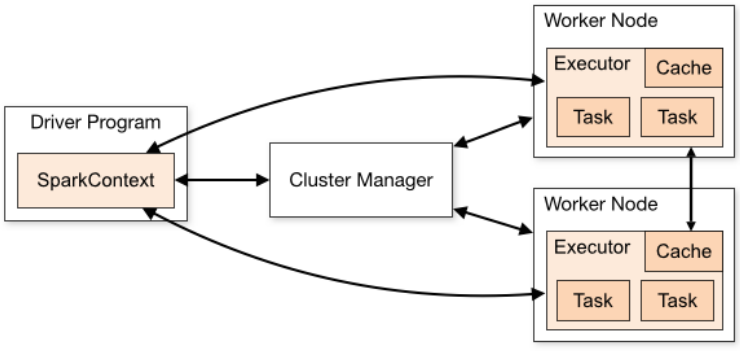

Abbildung 2.4.: Apache Spark Architektur, vgl. Apache-Foundation (2018i)

Wie auch *Apache Hadoop*, verarbeitet Spark Daten im *Batch-Processing*. Batch-Processing bedeutet, dass eine Datenmenge verarbeitet wird. Im Gegensatz dazu steht das Verarbeitungsparadigma *One-at-a-time*, das strikt einen einzelnen Datensatz vollständig verarbeitet, bevor die Verarbeitung des nächsten Datensatzes begonnen wird.

Aufbauend auf Apache Spark hat sich ein Ökosystem entwickelt, das in Abbildung 2.5 dargestellt ist. Es wurden Bibliotheken auf Spark entwickelt, die die Möglichkeiten von Spark erweitern und abstrahieren. Auf den Nutzen dieser Erweiterungen wird im Folgenden kurz eingegangen.

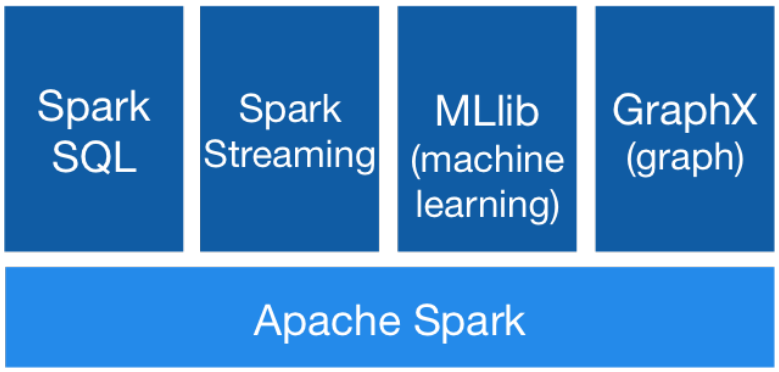

Abbildung 2.5.: Apache Spark Ökosystem, vgl. Apache-Foundation (2018j)

Spark Streaming nutzt Spark um Stream-Verarbeitung zu realisieren, die die Streamdaten in *Micro Batches* verarbeitet. *MLlib* hat Machine Learning Algorithmen für Spark implementiert, sodass aufwändige ML-Algorithmen und große Datenmengen von der verteilten Verarbeitung durch Spark profitieren. *GraphX* verwendet Spark für die Anwendung von Graphenalgorithmen. *Spark SQL* erlaubt es, Spark-Jobs als SQL-Anfragen zu formulieren, die anschließend optimiert und als Spark-Job in einem Cluster ausgeführt werden. Spark SQL wurde von Armbrust u. a. (2015) entwickelt, um bei der Analyse von großen Datenmengen kürzere Analyseintervalle und mehr Interaktion während der Datenanalyse zu bieten. Die Erweiterung bietet eine deklarative API zum Entwickeln von aufwändigen Jobs, ohne, dass der Entwickler spezielle Kenntnisse über der Entwicklung von Spark-Jobs benötigt.

2.4. HDFS

Das HDFS ist ein Teil des OpenSource Projekts *Apache Hadoop* der *Apache Foundation*. Dieses ist laut Apache-Foundation (2018h) seit 2008 ein Apache Top Level Projekt. Mit dem Upgrade von Version 1 zu Version 2 wurde Hadoop in mehrere Komponenten aufgeteilt, die unabhängig

voneinander eingesetzt werden können. Die Komponentenaufteilung ist in Abbildung 2.6 dargestellt. Dort ist zu sehen, dass vor allem der Block *MapReduce* aus Version 1 für Version 2 in die Komponenten *YARN*, *Tez* und *MR* aufgeteilt wurde. Bei den separierten Komponenten handelt es sich um die Teile von Hadoop, die für das Batch-Processing zuständig sind, also die eigentliche *Hadoop* Engine.

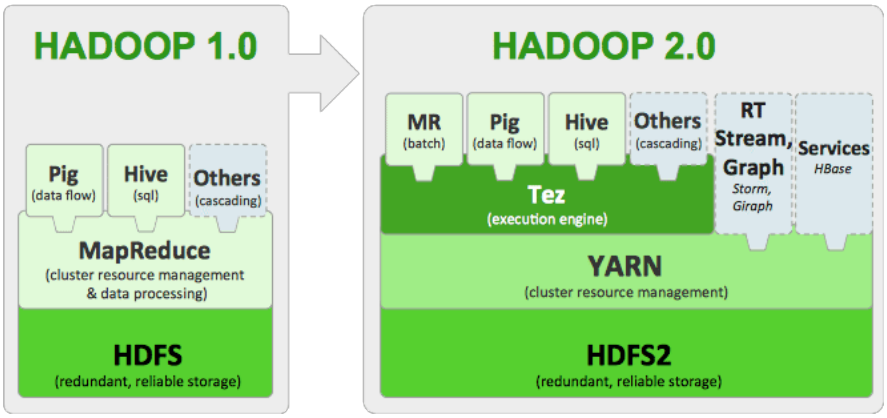

Abbildung 2.6.: Hadoop Technologie-Stack , vgl. HortonWorks (2013)

Im Rahmen dieser Arbeit wird nur das HDFS aus dem Hadoop-Stack verwendet, daher wird im Folgenden nur auf diese Komponente eingegangen. Das HDFS ist ein verteiltes Dateisystem, das die Blöcke der gespeicherten Dateien auf mehrere Nodes verteilt und Duplikate der Blöcke erstellt. Das HDFS wird häufig in Kombination mit Verarbeitungs-Engines wie *MapReduce* aus dem Hadoop-Stack, *Apache Spark* oder *Apache Flink* verwendet.

Ein HDFS-Cluster besteht aus einer Master-Slave-Architektur, die in Abbildung 2.7 dargestellt ist. Jeder Cluster verfügt über einen *NameNode*, der Metadaten über die gespeicherten Dateien bereitstellt. Bei den Metadaten handelt es sich um Informationen wie zum Beispiel: - wie ist die Ordnerstruktur im HDFS, - welche Berechtigungen liegen auf einer Datei oder einem Ordner und - welche Blöcke einer Datei sind auf welchen DataNodes abgelegt. Die *DataNodes* sind die Slaves, die lediglich Dateiblöcke speichern und anfragenden Clients bereitstellen.

Möchte ein Client beispielsweise eine Datei aus dem HDFS lesen, fragt er die Datei zunächst bei dem NameNode an. Der NameNode antwortet mit der Information, welche Dateiblöcke der angefragten Datei auf welchen DataNodes zu finden sind. Anschließend fragt der Client die Dateiblöcke direkt bei den DataNodes an und kann sie von dort laden. Analog verhält es sich beim Schreiben in den HDFS-Cluster. Möchte ein Client eine Datei in den HDFS-Cluster schreiben, spricht er zunächst den NameNode an. Der NameNode erzeugt in seinen Metadaten die Information, dass es eine neue Datei gibt und teilt dem Client mit, welche Dateiblöcke

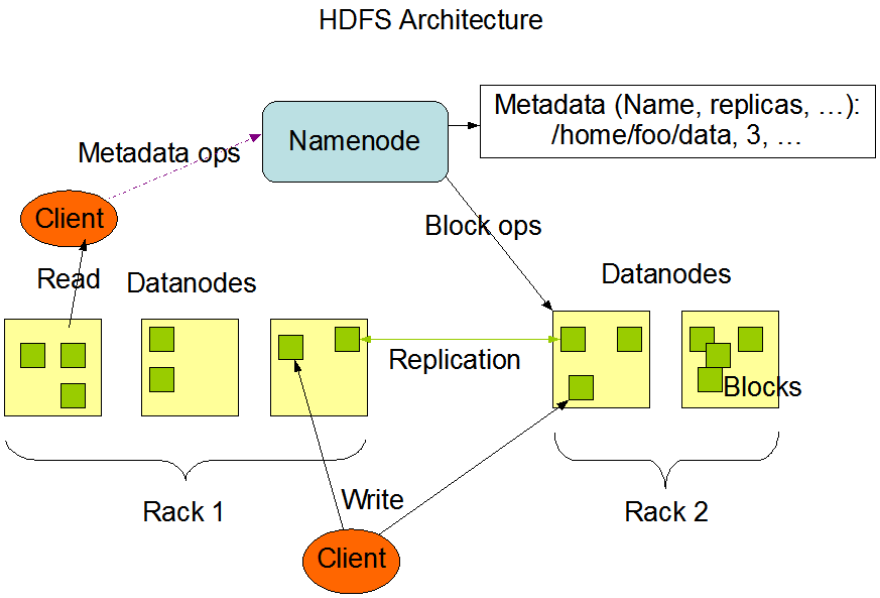

Abbildung 2.7.: HDFS Architektur , vgl. Apache-Foundation (2018e)

dieser auf welche DataNodes zu schreiben hat. Anschließend schickt der Client die Dateiblöcke zu den entsprechenden DataNodes.

Das HDFS bietet außerdem eine gewisse Ausfallsicherheit. Die Standardeinstellung eines HDFS-Clusters ist, dass jeder Dateiblock drei mal vorhanden ist. Sobald ein Client also einen Dateiblock auf einen DataNode geschrieben hat, erteilt der NameNode den Auftrag diesen Block an einen anderen DataNode zu replizieren. Sollte ein DataNode ausfallen, dann ist ein Dateiblock noch auf zwei anderen DataNodes vorhanden und die Integrität der Datei ist weiterhin gewährleistet.

Im Bereich von BigData-Processing-Engines vermeidet man es, große Datenmengen über ein Netzwerk zu schicken, da eine Netzwerkverbindung langsam ist verglichen mit der Datenübertragungsrate innerhalb eines Computers. Hat man beispielsweise einen Hadoop-MapReduce-Cluster, der mehrere verarbeitende Nodes hat und Dateien aus einem HDFS-Cluster als Datenquelle verwendet, dann stellt man auf jedem verarbeitenden Node auf einen DataNode bereit. Damit hat jeder Node einen Teil der Datenquelle Node-Lokal liegen und es müssen deutlich weniger Daten über das Netzwerk von einer Datenquelle zum verarbeitenden Node geschickt werden.

3. TPC-DI: Benchmark Spezifikation

Der TPC-DI Benchmark des Transaction Processing Performance Council (2014) ist eine umfangreiche Spezifikation, die entwickelt wurde, um Datenintegrations-tools und -frameworks unterschiedlicher Hersteller miteinander vergleichen zu können. Sie beschreibt wie die Quelldateien formatiert sind, wie sie zu transformieren sind und welches Schema die Ziel-SQL-Tabellen haben. Sie definiert außerdem, welche Messmetriken an welchen Zeitpunkten zu erheben sind, wie die Ergebnisse eines Benchmarks zu veröffentlichen sind und unterstützt damit beim herstellerübergreifenden Benchmarking von Datenintegrationstools.

Die Spezifikation wird bereitgestellt für einen *Testsponsor*, der die Hard- und Software zur Verfügung stellt. Der Testsponsor ist verantwortlich für die Implementierung und Durchführung der Messung gemäß der TPC-DI Spezifikation. Die Messergebnisse und die Details der Messung stellt der Testsponsor als Full Disclosure Report (FDR) bereit, um sie nach einer Überprüfung durch die TPC veröffentlichen zu können. Die Spezifikation bezeichnet die Hard- und Softwarekomponenten, die an dem Benchmark beteiligt sind als System Under Test (SUT). Im Folgenden wird das SUT auch als *Messsystem* bezeichnet.

Die Spezifikation unterteilt die Datentransformationen in historische und inkrementelle Imports. Als historischer Import versteht der TPC-DI das initiale Laden von Daten in die Datensenke. Dabei kann es sich um einen großen Datensatz handeln, dessen Import undefiniert lange dauern kann. Ein inkrementeller Import beinhaltet das regelmäßige Laden von Datenänderungen, was laut Spezifikation täglich stattfindet. Die Datenmenge eines inkrementellen Imports ist entsprechend kleiner im Vergleich zu einem historischen Import und kann daher deutlich schneller verarbeitet werden. Die Spezifikation definiert für den historischen und die inkrementellen Imports unterschiedliche Transformationsprozesse, die implementiert werden müssen. Die inkrementellen Imports müssen so implementiert sein, dass sie wiederholbar sind, daher werden zwei inkrementelle Imports von der Spezifikation gefordert.

Im Folgenden werden die Inhalte der TPC-DI Spezifikation auf die markanten Punkte reduziert, die für das weitere Verständnis der Arbeit wichtig sind. Die Details zu den einzelnen Themen sind direkt in der Spezifikation von Transaction Processing Performance Council (2014) zu finden.

3.1. Fachlichkeit der Benchmarkdaten

Die Spezifikation geht von einem konkreten Anwendungsfall aus, der in Abbildung 3.1 konzeptionell dargestellt ist. Es handelt sich fachlich um ein Datenmodell aus dem Aktienhandel. Wie in der Abbildung gezeigt, gibt es mehrere Systeme, die unterschiedliche fachliche Datenhoheiten haben. Die Daten dieser Systeme sollen zusammengeführt und in einem *Data Warehouse* abgelegt werden. Dazu stellt jedes der fachlichen Quellsysteme seine Dateien den Transformationprozessen in der *Staging Area* bereit. Die Transformationen nehmen diese Daten auf, führen die definierten Verarbeitungsschritte durch und schreiben sie anschließend in das Data Warehouse.

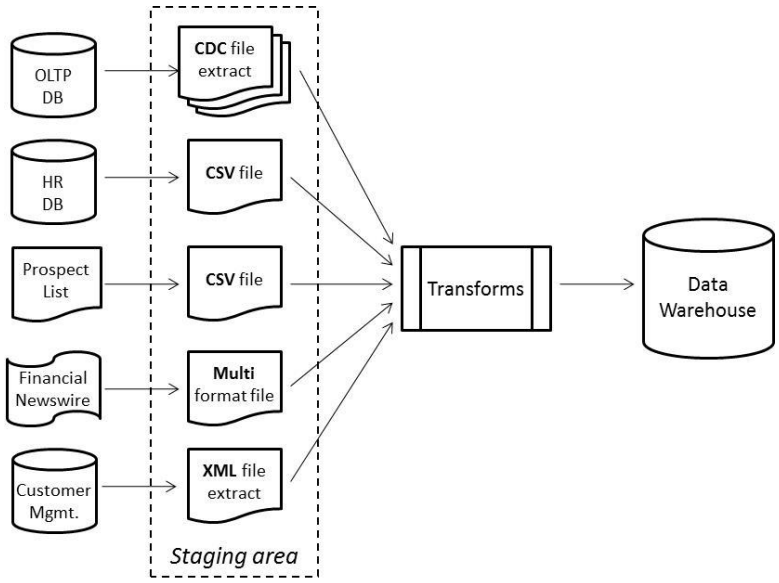

Abbildung 3.1.: Fachliche Konzeption des Ausgangssystems (vgl. Transaction Processing Performance Council, 2014, S. 12)

Die Spezifikation lässt offen, wie die Staging Area, die Transformationen und das Data Warehouse technisch realisiert werden. Es werden allerdings Bedingungen an die beteiligten Hardware- und Software-Komponenten gestellt, auf die in Abschnitt 3.8 eingegangen wird.

Laut der Spezifikation decken die Quelldaten, Transformationen und das Data Warehouse nicht alle Anforderungen und Fälle ab, die an ein Datenintegrationssystem gestellt werden können. Da die Daten, Transformationen und Geschäftsregeln auf ein Datenmodell aus dem Aktienhandel abgestimmt sind, lassen sich die Messergebnisse auch nur auf diese Fachlichkeit beziehen. Dennoch lassen sich Teile des Benchmarks auch auf andere Anwendungsgebiete übertragen.

3.2. Datengenerierung

Die TPC bietet den Datengenerator *DIGen*[1] an, der die Quelldaten generiert, so wie sie in der Spezifikation beschrieben werden. DIGen ist mit dem Parameter *scale-factor* konfigurierbar, womit die Größe des zu generierenden Datensets bestimmt werden kann. Dabei entspricht der Skalierungsfaktor 10 etwa einer Datenmenge von 1GB. DIGen generiert Daten für drei Imports: ein historischer Import und zwei inkrementelle Imports. Die Größe der Importdaten muss laut Transaction Processing Performance Council (2014, S. 15) so gewählt sein, dass ein inkrementeller Import 30min - 60min dauert.

DIGen erstellt zu jeder Importdatei eine Auditdatei, deren Inhalt für die Audit-Phase benötigt wird. Auf die Audit-Phase wird in Abschnitt 3.6 eingegangen. Die Auditdateien enthalten Informationen über die zugehörige Importdatei. In der Auditdatei für beispielsweise die Importdatei *Prospect.csv* steht, für wieviele Prospect-Zeilen es einen passenden Datensatz aus der *Kunden*-Tabelle gibt und wieviele der Prospect-Zeilen neu in der Datensenke eingefügt werden müssen. Diese Informationen stehen ausschließlich für die Audit-Phase bereit und dürfen nicht in den Transformationsprozessen verwendet werden.

Zusätzlich erzeugt DIGen einen Bericht, der Aufschluss darüber gibt, wieviele Datenzeilen in den einzelnen Batches generiert wurden und durch die Transformationsprozesse zu verarbeiten sind.

Der Datengenerator DIGen basiert auf dem Tool *PDGF* von Rabl u. a. (2011), das die TPC auch in anderen Benchmarks verwendet. Damit die generierten Daten konsistent und fachlich korrekt sind, gibt es laut Poess u. a. (2014) in PDGF die Abstraktionsschicht der *Update Black Box*. Durch diese Schicht wird geregelt, wie die Lebenszyklen der einzelnen Entitäten erfolgen und welche Interaktionen in welchem Status passieren dürfen. Für die Entitäten *Kunde* und *Konto* beispielsweise gibt es folgendes Modell: Zur Erstellungszeit eines Kunden, hat der Kunde den Status *aktiv*. In diesem Status kann der Kunde Veränderungen erhalten, es kann zum Beispiel seine Adresse oder Telefonnummer verändert werden. Jeder Kunde hat außerdem mindestens ein Konto und während der Kunde aktiv ist, können beliebig viele Konten dazukommen. Zum Lebenszyklus eines Kunden gehört auch, dass er irgendwann in den Status *inaktiv* wechselt. Ab diesem Zeitpunkt dürfen keine Veränderungen mehr stattfinden und auch keine weiteren Konten zu ihm hinzugefügt werden. Zeitgleich mit der Deaktivierung des Kunden werden auch die Konten des Kunden deaktiviert. Auf seinen Konten dürfen daher ab diesem Zeitpunkt auch keine Veränderungen mehr stattfinden.

Dieser und weitere komplexe Lebenszyklen werden durch DIGen generiert und beachtet.

[1] Herunterladbar unter http://www.tpc.org/tpcdi/

3.3. Format der Quelldateien

Das Schema der Quelldateien ist in Transaction Processing Performance Council (2014, S. 20-35) definiert. Dort ist beschrieben, welche Felder es mit welchen Einschränkungen gibt.

Tabelle 3.1.: Eigenschaften der Importdateien, vgl. Poess u. a. (2014)

Dateiname	Format	H	I
Account.txt	DEL/CDC		✓
CashTransaction.txt	DEL/CDC	✓	✓
Customer.txt	DEL/CDC		✓
CustomerMgmt.xml	XML	✓	
DailyMarket.txt	DEL/CDC	✓	✓
Date.txt	DEL	✓	
Time.txt	DEL	✓	
FINWIRE*	Multi-Record	✓	
HoldingHistory.txt	DEL/CDC	✓	✓
HR.csv	CSV	✓	✓
Industry.txt	DEL	✓	
Prospect.csv	CSV	✓	✓
StatusType.txt	DEL	✓	
TaxRate.txt	DEL	✓	
TradeHistory.txt	DEL	✓	
Trade.txt	DEL/CDC	✓	✓
TradeType.txt	DEL	✓	
WatchHistory.txt	DEL/CDC	✓	✓

Ein Überblick über die Dateien und einige der relevanten Eigenschaften sind in Tabelle 3.1 aufgeführt. Die Spalte *Format* der Tabelle beschreibt die Formatierung der Datei:

DEL Die Felder einer Zeile sind mit dem „|“-Zeichen separiert.

CSV Die Felder einer Zeile sind mit einem Komma separiert.

XML Der Inhalt der Datei ist im XML-Format angegeben.

CDC In einem inkrementellen Import werden dem Schema dieser Datei zusätzlich die Felder *CDC_FLAG* und *CDC_DSN* vorangestellt.

Multi-Record Die Datei enthält Zeilen von drei Entitätstypen, die unterschiedliche Schemata haben. Die Felder einer Zeile werden anhand der festen Länge eines Feldes ausgelesen. Die ersten drei Felder der möglichen Entitätstypen sind gleich lang und anhand eines der Felder kann erkannt werden, von welchem Entitätstyp die Zeile ist.

Die Spalten *H* und *I* haben jeweils einen Haken, falls die Datei im historischen oder in einem inkrementellen Import vorkommt und zu verarbeiten ist. Alle Dateien, außer den FINWIRE-Dateien, treten in einem Batch einzeln auf. Für jedes Quartal eines Jahres wird jeweils eine FINWIRE-Datei erstellt, die alle während des historischen Imports in chronologischer Reihenfolge importiert werden müssen.

Einige der in Tabelle 3.1 aufgeführten Quelldateien haben in einem inkrementellen Import zusätzliche Felder im Vergleich zum historischen Import. Dies stellt die Anforderung an die Datenintegrationsapplikation, dass sie unterschiedliche Schemata einer Quelldatei akzeptiert und verarbeiten kann.

3.4. Schemadefinition der Datensenke

Das Schema der Datenbanktabellen ist in Transaction Processing Performance Council (2014, S. 37-46) definiert. Dort ist beschrieben, welche Felder es mit welchen Datentypen und Beschränkungen gibt. Die Datenbanktabellen dienen als Datensenke für die Transformationsprozesse.

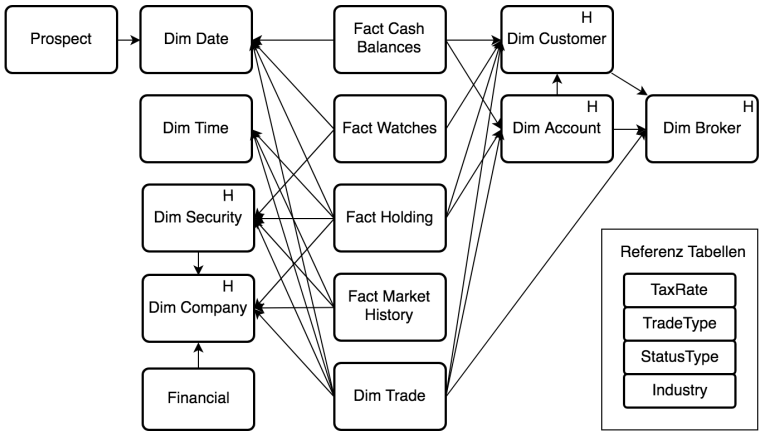

Abbildung 3.2.: Snowflake-Schema der Datenbanktabellen, vgl. Poess u. a. (2014)

In Abbildung 3.2 ist das *Snowflake*-Schema der Datenbanktabellen dargestellt, das nach Kimball und Ross (2011) aus Fakt-, Dimensions- und Referenztabellen besteht. Die Dimensionstabellen beschreiben die Geschäftsentitäten und die Fakttabellen enthalten messbare Informationen, die beschreiben, was mit den Geschäftsentitäten passiert ist. Eine Fakttabelle enthält beispielsweise zu welchem Zeitpunkt eine Transaktion mit welchem Preis erfolgt ist oder wieviele Aktien jemand an einem Datum besessen hat. Es gibt Tabellen, die beide Rollen gleichzeitig besitzen, wie zum Beispiel die Tabelle *DimTrade*. Je nachdem wie sie verwendet wird, ist es eine Fakt- oder Dimensionstabelle. Das Schema enthält außerdem einige Referenz-

tabellen, die statischen Inhalt besitzen. Diese Tabellen werden zur Normalisierung der Daten verwendet.

Einige der Entitäten in Abbildung 3.2 sind mit einem *H* gekennzeichnet, diese Tabellen verfügen über ein Historie. Wenn eine Zeile aus diesen Tabellen aktualisiert werden soll, wird nicht direkt die Zeile aktualisiert, sondern es wird eine neue Zeile mit den neuen Werten erstellt. Diese Tabellen verfügen über zwei Schlüssel, der Entitätenschlüssel, zum Beispiel *CustomerID*, und ein technischer Schlüssel, der in der Spezifikation als *Surrogate Key* bezeichnet wird. Es kann daher mehrere Zeilen mit dem selben Entitätenschlüssel geben, die anhand des *Surrogate Keys* eindeutig identifiziert werden können. Der *Surrogate Key* wird von anderen Tabellen als Fremdschlüssel referenziert. Jede der Historie-Tabellen verfügt außerdem über eine Boolean-Spalte *IsCurrent*, die beschreibt, ob die jeweilige Zeile die aktuellste eines Entitätenschlüssels ist. Sucht man beispielsweise die aktuellste Zeile einer *CustomerID*, lässt sich diese aus der Kombination von *CustomerID* und $IsCurrent = true$ identifizieren.

Das Schema besteht aus vier Fakttabellen, zehn Dimensions- und vier Referenztabellen. Alle diese Tabellen werden während des historischen Imports durch die Transformationsprozesse befüllt. Die Abbildung 3.2 zeigt, welche Tabellen einander referenzieren. Ein Pfeil zwischen zwei Tabellen bedeutet, dass die Pfeilquelle eine Referenz auf das Pfeilende hält. In der Spezifikation der Datensenke darf keine Fremdreferenz leer sein. Dadurch ergibt sich eine Reihenfolge, in der die Transformationsprozesse ausgeführt werden müssen. Die Tabelle *Dim Account* ist beispielsweise abhängig von der Tabelle *Dim Customer*. Daher kann *Dim Account* erst befüllt werden, wenn der Import von *Dim Customer* beendet ist. In Transaction Processing Performance Council (2014, S. 53-54) ist eine tabellarische Auflistung enthalten, die zeigt welche Imports voneinander abhängig sind.

Zusätzlich zu den in Abbildung 3.2 beschriebenen Tabellen gibt es die Tabellen *Audit* und *DIMessage*. Einige Transformationsprozesse schreiben in die Tabelle DIMessage Nachrichten hinein, wenn sie Daten verarbeiten, die gegen definierte Geschäftsregeln verstoßen oder geben Information, wieviele neue und aktualisierte Datensätze geschrieben wurden. Die Tabelle *DIMessage* ist daher keine fachliche Entität, sondern stellt ein einfaches Logging-Verfahren der Prozesse dar.

In der Tabelle *Audit* werden die von DIGen generierten Auditdateien während der Auditphase importiert. Die Audit-Tabelle unterstützt daher bei der Überprüfung der Transformationsprozesse und ist wie die DIMessage-Tabelle keine fachliche Entität.

3.5. Spezifikation der Transformationsprozesse

Die TPC-DI Spezifikation beschreibt detailliert wie die Quelldateien zu verarbeiten und in die Datenbanktabellen zu schreiben sind. Dazu werden viele unterschiedliche Transformationen angewandt, die in einem realistischen Anwendungsszenario auftreten können. Im Folgenden werden die Transformationsprozesse anhand einiger Eigenschaften beschrieben und welche Quelldateien und Datenbanktabellen sie verwenden. Jeder der spezifizierten Transformationsprozesse importiert eine oder mehrere Quelldateien in eine Datenbanktabelle. Jeder Transformationsprozess ist daher ein *Importprozess*.

Die einfachsten Prozesse sind die, die die Referenztabellen befüllen. Sie schreiben die Importdaten unverändert in die Datenbank. Es handelt sich dabei um folgende Quelldateien und Datenbanktabellen:

- *Date.txt* wird in die Tabelle *DimDate* geschrieben
- *Time.txt* wird in die Tabelle *DimTime* geschrieben
- *Industry.txt* wird in die Tabelle *Industry* geschrieben
- *StatusType.txt* wird in die Tabelle *StatusType* geschrieben
- *TaxRate.txt* wird in die Tabelle *TaxRate* geschrieben
- *TradeType.txt* wird in die Tabelle *TradeType* geschrieben

Die restlichen Prozesse müssen die Importdaten aufwändiger verarbeiten. Poess u. a. (2014) beschreiben die Prozesse anhand einiger Charakteristika, die in Tabelle 3.2 aufgeführt sind. Jede Eigenschaft erhält einen Bezeichner und eine Beschreibung. In Tabelle 3.3 sind die einzelnen Transformationsprozesse und die Charakteristikabezeichner aufgeführt. Wenn in einer Zelle ein Haken gesetzt ist, bedeutet das, dass in diesem Prozess diese Eigenschaft genutzt wird. Da die Transformationsprozesse in der Spezifikation keine eindeutigen Namen erhalten, haben Poess u. a. (2014) ein Namensschema entwickelt. Der Name eines Prozesses besteht aus einer Kombination der Phase, in der er statt findet und aus dem Tabellennamen, den er befüllt. Der Importprozess, der im historischen Import die Tabelle *DimAccount* befüllt, heißt beispielsweise $T_{H,DimAccount}$. Der inkrementelle Import für eine Tabelle wird mit einem I gekennzeichnet: $T_{I,DimAccount}$

Nach diesem Schema wurden die Prozesse in Tabelle 3.3 betitelt und mit den entsprechenenden Transformations-Charakteristika versehen. Dort ist zu abzulesen, dass die oben genannten

einfachen und statischen Importprozesse nur die Eigenschaft C_4 aufweisen, die für das Einlesen von CSV-Daten und Umwandeln zu Relationalen Daten steht. Die anderen Prozesse weisen mehr Eigenschaften auf und erzeugen damit potenziell mehr Implementations- und Verarbeitungsaufwand.

In Abbildung 3.3 ist dargestellt, aus welchen Quellen die Datenbanktabellen befüllt werden. Die Kästen mit unterbrochenen Linien sind Quelldateien, die von *DIGen* für den jeweiligen Batch generiert wurden. Die Kästen mit einer durchgehenden Linie sind Datenbanktabellen.

Die Quelldateien, die mit einem *H* gekennzeichnet sind, kommen nur im historischen Import vor und sind in den inkrementellen Imports nicht mehr zu beachten. Die Quelldateien, die mit einem *I* gekennzeichnet sind, kommen nur in inkrementellen Imports vor und sind in dem historischen Import nicht zu beachten. Die Quelldateien, die weder mit einem *H*, noch einem *I* gekennzeichnet sind, müssen in historischen und inkrementellen Imports verarbeitet werden.

Es ist zu sehen, dass Datenbanktabellen als Datensenken und als Datenquellen in den Importprozessen auftreten. In der Abbildung ist außerdem ersichtlich, welche Quellen durch den Transformationsprozess zusammengeführt werden. Damit lassen sich die Charakteristika C_1, C_4, C_8, C_{10}, C_{11} und C_{14} nachvollziehen.

Tabelle 3.2.: Transformations-Charakteristika, vgl. Poess u. a. (2014)

Bezeichner	Beschreibung
C_1	Transformiert XML zu relationalen Daten
C_2	Erkennt Änderungen in Dimensionsdaten und wendet Historie-Mechanismus an
C_3	Schreibt Zeilen in Tabelle *DIMessage*
C_4	Transformiert CSV zu relationalen Daten
C_5	Filtert Eingabedaten anhand von Bedingungen
C_6	Erkennt neue, gelöschte und aktualisierte Einträge in Eingabedaten
C_7	Konvertiert fehlende Werte zu *NULL*
C_8	Führt Daten aus einer Quelldatei mit Daten einer anderen Quelldatei zusammen, die unterschiedliche Schemata haben
C_9	Standardisiert Einträge einer Quelldatei
C_{10}	Führt Daten von mehreren Quelldateien mit Dimensionstabelle zusammen
C_{11}	Führt Daten von mehreren Quellen zusammen, die unterschiedliche Formate haben
C_{12}	Konsolidiert mehrere Änderungen eines Tages und erkenne die letzte
C_{13}	Führt umfangreiche arithmetische Berechnungen durch
C_{14}	Liest Dateien, deren Zeilen unterschiedliche Typen haben
C_{15}	Prüft Daten auf Fehler oder das Einhalten von definierten Regeln

Tabelle 3.3.: Charakteristika der Transformationsprozesse, vgl. Poess u. a. (2014)

	Transformations-Charakteristika														
Importjob	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}
$T_{H,DimAccount}$	✓	✓	✓		✓		✓				✓	✓			
$T_{I,DimAccount}$		✓	✓	✓	✓		✓				✓	✓			
$T_{H,DimBroker}$				✓	✓										
$T_{H,DimCompany}$		✓	✓		✓	✓	✓			✓				✓	
$T_{H,DimCustomer}$	✓	✓	✓		✓	✓	✓		✓	✓		✓			
$T_{I,DimCustomer}$		✓	✓	✓	✓	✓	✓		✓	✓		✓			
$T_{H,DimDate}$				✓											
$T_{H,DimSecurity}$		✓			✓	✓				✓	✓	✓		✓	✓
$T_{H,DimTime}$				✓											
$T_{H,DimTrade}$			✓	✓				✓		✓	✓				
$T_{I,DimTrade}$			✓	✓						✓	✓				
$T_{H,FactCashBalances}$				✓						✓	✓				
$T_{I,FactCashBalances}$				✓						✓	✓				
$T_{H,FactHolding}$				✓						✓	✓		✓		
$T_{I,FactHolding}$				✓						✓	✓		✓		
$T_{H,FactMarketHistory}$			✓	✓					✓	✓	✓	✓			
$T_{I,FactMarketHistory}$			✓	✓					✓	✓	✓	✓			
$T_{H,FactWatches}$				✓		✓	✓			✓	✓		✓		
$T_{I,FactWatches}$				✓		✓	✓			✓	✓		✓		
$T_{H,Industry}$				✓											
$T_{H,Financial}$						✓							✓	✓	✓
$T_{H,Prospect}$		✓	✓	✓						✓					
$T_{I,Prospect}$		✓	✓	✓						✓					
$T_{H,StatusType}$				✓											
$T_{H,TaxRate}$				✓											
$T_{H,TradeType}$				✓											

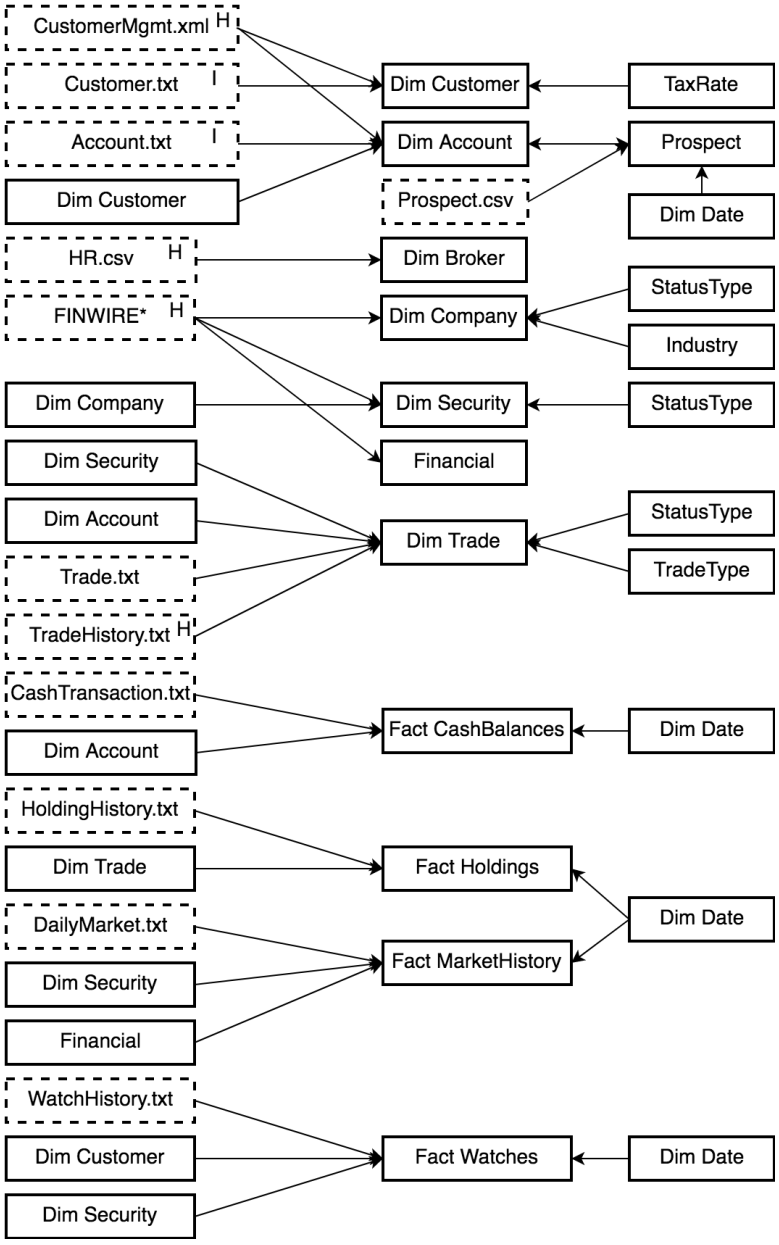

Abbildung 3.3.: Quellen und Ziele der Transformationsprozesse

3.6. Ausführungsphasen

Transaction Processing Performance Council (2014, S. 84-88) beschreibt einige Ausführungsphasen, die definieren, zu welchem Zeitpunkt des Benchmarks was passieren muss und in welchen Phasen Messdaten erhoben werden.

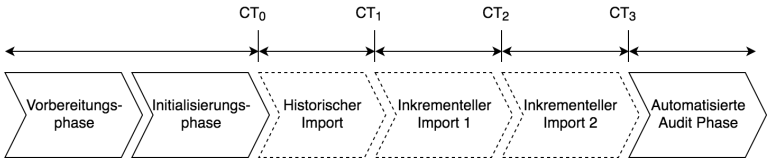

Abbildung 3.4.: Ausführungsphasen gemäß der TPC-DI, vgl. Poess u. a. (2014)

Die einzelnen Phasen sind in Abbildung 3.4 gezeigt. Jeder der Kästen stellt eine Messphase dar. In den Messphasen, deren Kästen eine gestrichelten Rahmen haben, werden Messdaten erhoben. Die jeweils nächste Phase wird erst begonnen, sobald die vorherige Phase beendet wurde.

In der *Vorbereitungsphase* werden einmalige Aufgaben durchgeführt, die zur Lauffähigkeit der Messung beitragen. Dazu gehören laut der Spezifikation zum Beispiel Aufgaben wie das Generieren der Importdaten mit *DIGen*, das Einrichten der Messsysteme mit einem Betriebssystem oder das Installieren des Datenbanksystems. Anschließend wird in der *Initialisierungsphase* ein *sauberer* Stand hergestellt. Dazu wird die Datenkbank mit dem Tabellenschema überschrieben und andere Artefakte, die von vorherigen Messungen vorhanden sind, werden entfernt.

Nach der Initialisierungsphase kann die Messung beginnen. Zu den in Abbildung 3.4 gezeigten Zeitpunkten CT_0, CT_1, CT_2 und CT_3 wird eine sogenannte *Batch Validation Query* ausgeführt, die den aktuellen Stand der Datenkbank erfasst und für die spätere *Automatisierte Audit Phase* bereitstellt. Die *Batch Validation Query* ist laut Transaction Processing Performance Council (2014, S. 88) in Anhang B der Spezifikation zu finden. In Anhang B wird auf herunterladbare Zusatzinhalte hingewiesen, die nicht auf der Webseite der TPC[2] zu finden sind. Das Fehlen der Inhalte wurde dem Chairman der TPC-DI Meikel Poess mitgeteilt.

Nachdem die *Batch Validation Query* zum Zeitpunkt CT_0 durchgeführt wurde, beginnt der historische Import, der die BatchID 1 hat und den Ordner *Batch1* der generierten Daten in die Datenbank importiert. Sobald der historische Import erfolgreich beendet ist, wird zum Zeitpunkt CT_1 erneut die *Batch Validation Query* ausgeführt. Darauf folgend werden nacheinander die inkrementellen Imports mit den BatchIDs 2 und 3 ausgeführt, die die Ordner *Batch2*

[2] http://www.tpc.org/, Stand: 05.03.2018

und *Batch3* der generierten Daten in die Datenbank importieren. Zu den Zeitpunkten CT_2 und CT_3 wird wieder die *Batch Validation Query* ausgeführt.

Nachdem der letzte inkrementelle Import beendet ist, beginnt die *Automatisierte Audit Phase*. Während der Datengenerierung, die in Abschnitt 3.2 beschrieben ist, wurden neben den Importdateien auch Auditdateien generiert, die in dieser Phase in die Tabelle *Audit* geladen werden müssen. Anschließend muss die *Data Visibility 1 Query* durchgeführt werden, die laut TPC-DI Spezifikation im Anhang C der Spezifikation zu finden ist. In Anhang C wird, wie auch bei der *Batch Validation Query*, auf weitere nicht auffindbare Zusatzinhalte hingewiesen. Nach der *Data Visibility 1 Query* wird die *Audit Query* durchgeführt, die in Anhang A der Spezifikation zu finden ist. Auch in Anhang A der Spezifikation wird auf die nicht findbaren weiteren herunterladbaren Inhalte verwiesen.

Damit ein Messlauf gültig ist, muss jeder der Tests in der *Automatisierten Audit Phase* erfolgreich durchlaufen.

3.7. Zu erhebene Messmetriken

Die TPC-DI beschreibt Metriken, die im Anschluss an den beschriebenen Messlauf in Abschnitt 3.6 berechnet werden. Alle nachfolgenden Formeln wurden aus der Spezifikation entnommen. Für die Berechnung der Metriken werden die erhobenen Zeitpunkte CT_0, CT_1, CT_2 und CT_3 benötigt. Daraus werden die Laufzeiten der einzelnen Imports gebildet:

Historischer Import $E_H = CT_1 - CT_0$

Inkrementeller Import 1 $E_{I1} = CT_2 - CT_1$

Inkrementeller Import 2 $E_{I2} = CT_3 - CT_2$

Zur Berechnung des Durchsatzes eines Imports, wird die Zeilenanzahl eines Batches aus dem DIGen-Bericht verwendet. Die Zeilenanzahl eines Imports wird im Folgenden als R_x bezeichnet. Mit der Laufzeit und der Anzahl der verarbeiteten Zeilen eines Imports lässt sich der Durchsatz berechnen:

Historischer Import $T_H = R_H / E_H$

Inkrementeller Import 1 $T_{I1} = R_{I1} / Max(E_{I1}, 1800s)$

Inkrementeller Import 2 $T_{I2} = R_{I2} / Max(E_{I2}, 1800s)$

Wie zu sehen, unterscheidet sich die Durchsatz-Formel des historischen Imports von der Formel der inkrementellen Imports. Wie bereits in Abschnitt 3.2 erwähnt, soll der Skalierungsfaktor der generierten Daten so gewählt werden, dass ein inkrementeller Import zwischen 30 und 60 Minuten dauert. Wegen der Mindestdauer von 30 Minuten wird auch der Durchsatz der inkrementellen Imports mit mindestens 30 Minuten berechnet. Die Maximaldauer von 60 Minuten wird von der Spezifikation bei der Durchsatzberechnung nicht beachtet. Falls ein inkrementeller Import kürzer als 30 Minuten dauert, wirkt sich das negativ auf den berechneten Durchsatz aus, was laut der TPC-DI Spezifikation beabsichtigt ist.

Transaction Processing Performance Council (2014, S. 90) nennt die drei Hauptmetriken der TPC-DI Spezifikation, die *Performance Metrik*, die *Preis-Performance-Metrik* und das *Verfügbarkeitsdatum*.

Für die Performance Metrik werden die vorher berechneten Durchsatz-Werte benötigt, aus denen mit folgender Formel gemäß Spezifikation die Performance Metrik *TPC_DI_RPS* berechnet wird:

$$TPC\_DI\_RPS = Trunc(GeoMean(T_H, Min(T_{I1}, T_{I2})))$$

Die Funktion $Trunc(arg)$ gibt die ganze Zahl des Arguments zurück und die Funktion $GeoMean(args)$ berechnet das geometrische Mittel der gegebenen Argumente. Wie die Formel zeigt, bildet der *TPC_DI_RPS* den Mittelwert aus dem Durchsatz des historischen Imports und des schlechteren inkrementellen Imports. Poess u. a. (2014) gehen auf die Formel ein und erklären, dass man abhängig von der Importimplementation Performancegewinne im historischen oder im inkrementellen Import erzielen kann. Die Formel zur Kombination der Durchsätze berücksichtigt Performancegewinne gleichermaßen in beiden Importarten.

Mit der Preis-Performance-Metrik *Price-per-TPC_DI_RPS* wird das Verhältnis der Hardware-Kosten zur Performance angegeben. Sie wird gemäß Spezifikation wie folgt berechnet:

$$Price\text{-}per\text{-}TPC\_DI\_RPS = \$/TPC\_DI\_RPS$$

$ beschreibt den Preis des Messsystems, wie es die *TPC Pricing* Spezifikation nach Transaction Processing Performance Council (2018b) definiert. Die Liste der Komponenten, die für die Preisberechnung zu berücksichtigen sind, sind in Transaction Processing Performance Council (2014, S. 98) beschrieben. Dort werden folgende Komponenten genannt, die summiert den Messsystem-Preis bestimmen:

- Alle Software- und Hardware-Komponenten des Messsystems

- Speicherkapazitäts-Kosten der Datenbank und der Importdaten-Ablage. Was hier genau zu beachten ist, ist in Transaction Processing Performance Council (2014, S. 98-99) beschrieben
- Alle Software- oder Hardware-Komponenten, die zur Administration oder Wartung des Messsystems dienen
- Software, die zur Erstellung, Modifizierung oder Vorbereitung von Importjobs benötigt wird, für ein Minimum von fünf Nutzern

Die Spezifikation exkludiert außerdem folgende Punkte aus der Kostenberechnung:

- Geräte und Kabel des Endnutzers
- Geräte zur Ausführung von *DIGen*
- Hardware zur Entwicklung des Datenintegrations-Framework oder -Tools
- Geräte zur Erstellung des FDR

Als dritte Hauptmetrik nennt die TPC-DI Spezifikation das *Verfügbarkeitsdatum*. Dieses beschreibt den Zeitpunkt, ab dem alle Hardware- und Softwarekomponenten des Messsystems allgemein verfügbar sind. Da die TPC-DI sich auch an Hersteller von Datenintegrations-Hardware und -Software richtet, kann der Benchmmark mit Hard- oder Software durchgeführt werden, die noch nicht auf dem Markt verfügbar ist.

3.8. Qualifikation der Messhardware

In Kapitel 8 der TPC-DI Spezifikation werden Kriterien beschrieben, mit denen sich ein Messsystem für eine *offizielle* TPC-Messung qualifiziert. Es werden Anforderungen an die Ablage der Importdaten, an die Datenbank und an die Implementation der Transformationsprozesse gestellt.

Die Ablage der Importdaten muss beispielsweise während eines permanenten und nicht wiederherstellbaren Ausfalls eines Speichermediums der Importdatenablage weiterhin die Daten bereitstellen. An die Datenbank wird zum Beispiel die Anforderung gestellt, dass falls die Datenbank während eines Imports ausfällt, alle committeten Daten auch persistiert sind. Sobald die Datenbank wieder erreichbar ist, müssen die bis zum Ausfall persistierten Daten verfügbar sein. Für die Qualifikation der Importdatenablage und der Datenbank sind konkrete Szenarien in Kapitel 8 der Spezifikation beschrieben, die das System bestehen muss.

Außerdem werden die implementierten Transformationsprozesse geprüft. Dazu werden mit *DIGen* Daten mit dem Skalierungsfaktor 5 generiert und durch die Transformationsprozesse in die Datenbank importiert. In Transaction Processing Performance Council (2014, S. 92-94) sind SQL-Anfragen formuliert, die nach dem Import an die Datenbank gestellt werden und deren Ergebnis wird in *.txt*-Dateien gespeichert. In Transaction Processing Performance Council (2014, S. 91) wird ein Tool erwähnt, das bei der Auswertung der exportierten Dateien und deren Korrektheit unterstützen soll. Dieses Tool steht laut der Spezifikation auf der Webseite der TPC[3] zum Download bereit. Zum Zeitpunkt des Schreibens dieser Arbeit ist das Tool allerdings nicht auf der Webseite zu finden, was dem Chairman der TPC-DI Meikel Poess mitgeteilt wurde. Mit dem Tool kann man laut der Spezifikation feststellen, ob die exportierten Daten korrekt sind. Wenn diese Überprüfung erfolgreich ist, kann man annehmen, dass auch die Transformationsprozesse korrekt implementiert sind.

Damit eine Messung offiziell über die TPC veröffentlicht werden kann, muss das Messsystem jedes der beschriebenen Szenarien erfolgreich durchlaufen und die Logik der Transformationsprozesse muss erfolgreich geprüft worden sein.

3.9. Vollständiger Bericht des Benchmarks

Um das Ergebnis der Messungen offiziell als TPC Benchmark zu veröffentlichen, muss nicht nur die in Abschnitt 3.8 beschriebene Qualifizierung absolviert werden, sondern es muss ein vollständiger und formaler Bericht erstellt werden. Die TPC-DI Spezifikation beschreibt in Kapitel 10 wie ein solcher FDR auszusehen hat. Ein FDR muss nachvollziehbar beschreiben, wie das Messsystem funktioniert und wie es gestartet wird. Er enthält außerdem die in Abschnitt 3.7 beschriebenen Metriken und listet die Preise der Messkomponenten auf. Die Spezifikation fordert für den FDR ein bestimmtes Layout und zeigt ein Beispiel.

Der letzte Schritt vor der Veröffentlichung ist die Überprüfung des FDRs und der Messung durch einen TPC zertifizierten und unabhängigen Prüfer. Dieser Prüfer wird anschließend ebenfalls im FDR genannt. Zur Veröffentlichung der Messergebnisse werden der FDR, eine Beschreibung der Transformationsprozesse und die Quelldateien an die TPC übergeben. Abschließend muss der Testsponsor die Messergebnisse gegebenenfalls kostenpflichtig der Öffentlichkeit bereitstellen.

[3] http://www.tpc.org

4. Versuchsaufbau und -durchführung

Die Durchführung und Messung des Benchmarks erfolgt nach einem Vorgehen, das im Folgenden beschrieben wird. Dabei wird auf den Ablauf, die Parametrisierung jedes Laufs, die Hardware-Ausstattung der beteiligten Computer, die Verteilung der Systemkomponenten im Cluster, die verwendeten Datensets und die erfassten Metriken eingegangen.

Die Import-Prozesse dieser Arbeit entsprechen nahezu vollständig der Spezifikation des TPC-DI Benchmarks, der in Kapitel 3 beschrieben wird. Es wurden allerdings nicht alle Rahmenbedingungen beachtet, wie zum Beispiel die *Audit*-Phase oder der *FDR*. Damit entspricht die Datenerhebungen dieser Arbeit nicht einem vollständigen TPC-DI Benchmark.

Zusätzlich zu den Importjobs gemäß der TPC-DI Spezifikation gibt es einen Job, der einen Kafka-Broker als Datenquelle verwendet. Die Messwerte des Streaming-Imports können anschließend mit denen des Non-Streaming-Imports verglichen werden.

Im Folgenden wird der Versuchsaufbau beschrieben, mit dem die Messungen durchgeführt werden. Außerdem werden die Unterschiede zwischen der TPC-DI-Spezifikation und dem Vorgehen dieser Arbeit erläutert.

4.1. Hypothesen

Nach Beendigung der Messung werden viele Messwerte vorliegen, die interpretiert werden müssen. Um der Analyse und dem Messaufbau einen Rahmen zu geben, werden im Folgenden einige Hypothesen aufgestellt:

A Einer der Messkandidaten kann alle Importjobs der TPC-DI mit einer kürzeren Laufzeit durchführen als die Vergleichsimplementation.

B Je größer das zu verarbeitende Datenset ist, desto proportional länger dauert der Import.

C Je mehr Nodes an einem Importjob beteiligt sind, desto kürzer ist die Laufzeit des Imports.

D Während eines Imports ist die CPU- und RAM-Last des ausführenden Nodes höher als die der restlichen Nodes.

E Nimmt die Verarbeitung weniger Zeit in Anspruch, wenn ein Event-Stream statt eines Dateisystems als Datenquelle verwendet wird?

Die Hypothese A entsteht aus der TPC-DI Spezifikation, nach der eine kürzere Laufzeit zu einer besseren Bewertung führen kann. Eine kürzere Laufzeit führt zu einer höheren RPS-Metrik und bedeutet auch eine kürzere Belegung von Hardware-Ressourcen. Mit *Laufzeit* ist die gemessene Zeit eines Imports gemeint, die in den gemessenen Phasen gemäß des Abschnitts 3.6 stattfinden.

Mit der Hypothese B wird überprüft, ob die Laufzeit mit der Datensetgröße korreliert. Die erwartete Korrelation wird damit begründet, dass eine größere Menge an Daten die Hardware-Ressourcen länger belegen, was zu einer verlängerten Ausführungszeit führt.

Durch die Hypothese C wird überprüft, welche Auswirkungen das Skalierungsverhalten der Gobblin-Spark-Kombination auf die Laufzeit hat. Da bei mehreren Nodes mehr Hardware-Ressourcen zur Verfügung stehen als bei einem einzelnen Node, kann es zu einer Laufzeitverringerung kommen.

In der Hypothese D wird überprüft, auf welchem Node während eines Imports die meiste CPU- und RAM-Last entsteht. Diese Hypothese entsteht aus der Arbeit von Majchrzak u. a. (2011), die die Tools *Talend* und *Pentaho* verglichen haben. Sie kamen unter anderem zu dem Ergebnis, dass Talend mehr Verarbeitungsschritte an die SQL-Datenbank abgibt als es Pentaho tat. Es wird daher hier betrachtet, wieviel Verarbeitungsaufwand die Implementationen an die SQL-Datenbank abgeben.

Hypothese E stellt ist eine explorative Fragestellung. Sie überprüft, ob ein Event-Stream die Input-Daten schneller liefert als ein Dateisystem.

Der Messaufbau wird so entworfen, dass die genannten Hypothesen beantwortet werden können. Sie werden in der Analyse in Abschnitt 6.4 wieder aufgegriffen und beantwortet.

4.2. Implementation der Importjobs

Wie bereits erwähnt, wird im Rahmen dieser Arbeit die TPC-DI Spezifikation mit zwei Messkandidaten implementiert. Die einzige Anforderung an die Messkandidaten ist, dass ihr Quellcode offengelegt ist (*OpenSource*).

Der erste Messkandidat ist das *Talend Open Studio For Big Data*. Andere Arbeiten (siehe Abschnitt 1.5) argumentierten bereits für Talend-Produkte und führten anschließend einen

Benchmark mit ihnen durch. Die Talend-Produkte, die den Begriff *Open Studio* im Titel tragen, sind OpenSource und frei nutzbar. Sie haben allerdings Einschränkungen im Feature-Umfang, was unter anderem die parallele Datenverarbeitung verbietet. Die OpenSource-Variante des Studios kann außerdem nur im Standalone-Modus betrieben werden. Das bedeutet, dass sie nur auf einem einzelnen Computer laufen kann. Die kostenpflichtigen Varianten des Studios können Prozesse entwickeln, die in einem Hadoop- oder Spark-Cluster verwendbar sind. Auf Nachfrage bei dem Hersteller *Talend* wurde im Rahmen dieser Arbeit keine Enterprise-Lizenz ausgestellt, sodass die Skalierbarkeit nicht von Talend gemessen werden kann. Das Tool Talend wird daher nur im Standalone-Modus genutzt.

Der zweite Messkandidat ist eine Kombination aus zwei *Apache*-Frameworks. Ein Teil des Messkandidaten ist das bereits vorgestellte Map-Reduce-Framework *Apache Spark* und seine Erweiterung *Spark SQL*. Mit ihr soll der ETL-Prozess in der SQL formuliert werden. Es sollen Messungen mit unterschiedlichen Clustergrößen stattfinden. Durch die Implementation der TPC-DI in SQL, ist es für andere Arbeiten leichter die Implementation zu übernehmen, falls sie ebenfalls ihre ETL-Prozesse in SQL formulieren.

Bevor die Importdaten durch *Spark SQL* verarbeitet werden, müssen sie zunächst eingelesen werden. Da die Importdaten unstrukturiert vorliegen, müssten die unterschiedlichen Formate durch *Spark* unterstützt werden. Dies würde den Fokus der *Spark SQL*-Implementation von der reinen Datentransformation verschieben. Stattdessen wurde in den Apache Incubator Projekten das Dataingestion-Framework *Apache Gobblin* entdeckt. Wie in Abschnitt 2.2 beschrieben, wird Gobblin speziell für die Datenkonvertiertung und das Schreiben der konvertierten Daten in Datensenken entwickelt. Diese beiden Apache-Frameworks ergänzen sich in diesen Punkten. Mit *Gobblin* werden die Quelldaten konvertiert, damit die *Spark SQL*-Implementation einheitliche Daten verarbeiten kann. Der zweite Messkandidat ist daher eine Kombination aus *Apache Gobblin* und *Apache Spark*.

Die Vorgehen der beiden gewählten Import-Frameworks Talend und Gobblin-Spark sind unterschiedlich. Daher funktionieren die jeweiligen Importjobs anders mit unterschiedlichen Vor- und Nachteilen. Während die Talend-Importjobs viel mit der Datenbank interagieren, arbeiten die Spark-Imports viel im Hauptspeicher und schreiben lediglich die Ergebnisse in die Datenbank. Im Folgenden werden die Unterschiede anhand einiger Beispieljobs erklärt.

Als einfachstes Beispiel wird zunächst der *Date.txt*-Import in Talend erklärt. Talend liest die Datei *Date.txt* vom lokalen Dateisystem und verarbeitet sie zeilenweise. Die Verarbeitung einer Zeile beginnt mit dem Parsen der Zeile in die einzelnen Felder des definierten Schemas der Datei. Anschließend werden die Felder der Importdatei auf die Felder der Datenbanktabelle *DimDate*

abgebildet und zuletzt wird die Zeile in die Datenbank geschrieben. Diese Verarbeitung findet solange statt, bis alle Zeilen der Importdatei verarbeitet wurden.

Bei den Gobblin-Spark-Imports sind die Arbeitsschritte anders verteilt. Vor den Spark-Imports finden die Gobblin-Imports statt. Die Gobblin-Imports lesen die Importdateien aus einem HDFS-Ordner, parsen die Dateien Zeile für Zeile nach dem definierten TPC-DI Schema und schreiben sie im Avro-Format in einen HDFS-Ordner. Die Spark-Imports befassen sich nicht mehr mit dem Parsen der Importdateien, sondern lesen lediglich die Avro-Dateien vom HDFS-Ordner ein, die das Schema beinhalten. Der Spark-Import für die Importdatei *Date.txt* bildet ebenfalls alle Felder der Datei auf die Felder der Datenbanktabelle ab und schreibt anschließend die Zeilen in die Datenbank. Wie in Abschnitt 2.3 erläutert, verarbeitet Apache Spark Daten im Batch-Processing. Die Zeilen durchlaufen daher blockweise diese Verarbeitung. Erst wenn *alle* Zeilen den Mapping-Schritt durchlaufen haben, werden *alle* Zeilen gemeinsam in die Datenbank geschrieben.

Etwas komplexer ist der Job, der die Datei *CustomerMgmt.xml* importiert. Dort können mehrere Operationen, wie ein Update oder eine Deaktivierung, auf einem Datensatz stattfinden. Der Talend-Importjob parst die Datei zunächst und iteriert dann über die einzelnen Events, die in der Importdatei gegeben sind. Die Events werden sequentiell abgearbeitet. Je nachdem, welche Informationen in dem Event stehen, wird ein neuer Kunde erzeugt, ein Kunde aktualisiert, ein Kunde deaktiviert, ein neues Konto erzeugt, ein Konto aktualisiert oder ein Konto deaktiviert. Für diese Operationen werden die Events mit Daten aus der Datenbank zusammengeführt. Wird beispielsweise ein Kunde aktualisiert, dann wird der Datensatz des Kunden zunächst aus der Datenbank abgefragt, verändert und wieder in die Datenbank geschrieben. So enthält die Datenbank immer den aktuellen Stand, auf dem der Importjob basiert.

Der Spark-Importjob der *CustomerMgmt.xml* erwartet, dass vorher der Gobblin-Importjob lief, der die XML-Datei in eine Avro-Datei umgewandelt hat. Der Spark-Importjob liest die Avro-Dateien ein und lädt alle Datenbanktabellen, die zur Verarbeitung der Importdatei benötigt werden in den Hauptspeicher der Spark-Slaves. Alle folgenden Operationen finden ebenfalls im Hauptspeicher der Spark-Slaves statt. Anschließend werden die Events nach Entität - *Kunde* oder *Konto* - , nach Entitäten-ID - *CustomerID* und *AccountID* - und Eventdatum gruppiert. Die Zieldatenbanktabellen *DimCustomer* und *DimAccount* sind Tabellen mit Historie, jede Änderung eines Datensatzes muss daher in der Tabelle dokumentiert werden. Zu diesem Zweck wird für jede Entität und Entitäten-ID der älteste Eintrag aus der Importmenge gefiltert und in die Datenbank geschrieben. Anschließend werden die veränderten Datenbanktabellen wieder in den Hauptspeicher der Spark-Slaves geladen. Das Verarbeiten des ältesten Ereignisses

einer Entitäten-ID und Laden der Tabellen wird iterativ weitergeführt, bis alle Importdaten in die Datenbank importiert wurden.

Zwischen dem Laden der Datenbanktabellen und dem Schreiben der aktualisierten Zeilen kann, je nach Größe der Importdaten, viel Zeit liegen. Spark blockiert standardmäßig nicht die Zeilen im Datenbankserver auf denen es aktuell arbeitet. Wenn die Datensätze in der Datenbank zwischen diesen Zeitpunkten von einem anderen Programm verändert werden, bekommt der Spark-Importjob diese Änderung nicht mit und es kann zu einem inkonsistenten Datenstand kommen.

Mit dem ersten Kandidaten Talend werden die Importprozesse im ETL-Stil implementiert. Die Gobblin-Spark-Kombination implementiert die Prozesse im ELT-Stil, indem Gobblin die Daten zunächst in einem HDFS-Cluster ablegt. Anschließend liest Spark die Daten aus dem HDFS-Cluster und verarbeitet sie.

4.3. Messablauf

Die Messung besteht im Wesentlichen aus dem Bereitstellen (*Deployment*) von Systemkomponenten, dem Starten von Messjobs, dem Entfernen (*Undeployment*) von Systemkomponenten und der Validierung der Datensenken.

Zur weiteren Erläuterung des Ablaufs werden folgende Begriffsdefinitionen getroffen:

Messjob Als *Messjob* wird eine einzelne Messung bezeichnet, die alle Messschritte enthält außer den Iterationen über die Anzahl der beteiligten Nodes, der Datensetgröße und der Anzahl der Durchläufe. In Abbildung 4.1 ist diese Definition als Gruppe *Messjob* dargestellt.

Messlauf Als *Messlauf* wird eine komplett abgeschlossene Messung bezeichnet, in der Messjobs mit allen Parametervariationen durchlaufen wurden. In Abbildung 4.1 ist diese Definition als Gruppe *Messlauf* dargestellt.

Cluster Als *Cluster* wird der Computerverbund bezeichnet, der im Rahmen dieser Arbeit verwendet wird. Auf den Cluster-Nodes werden die Messungen durchgeführt und die einzelnen System-Komponenten bereitgestellt. Detaillierte Informationen zum Cluster sind in Abschnitt 4.5 beschrieben.

4.3.1. Phasen einer Messung

In Abbildung 4.1 ist der Ablauf eines Messlaufs dargestellt. Zu Beginn wird eine InfluxDB deployt, in der die erfassten Messdaten zur weiteren Verarbeitung gespeichert werden.

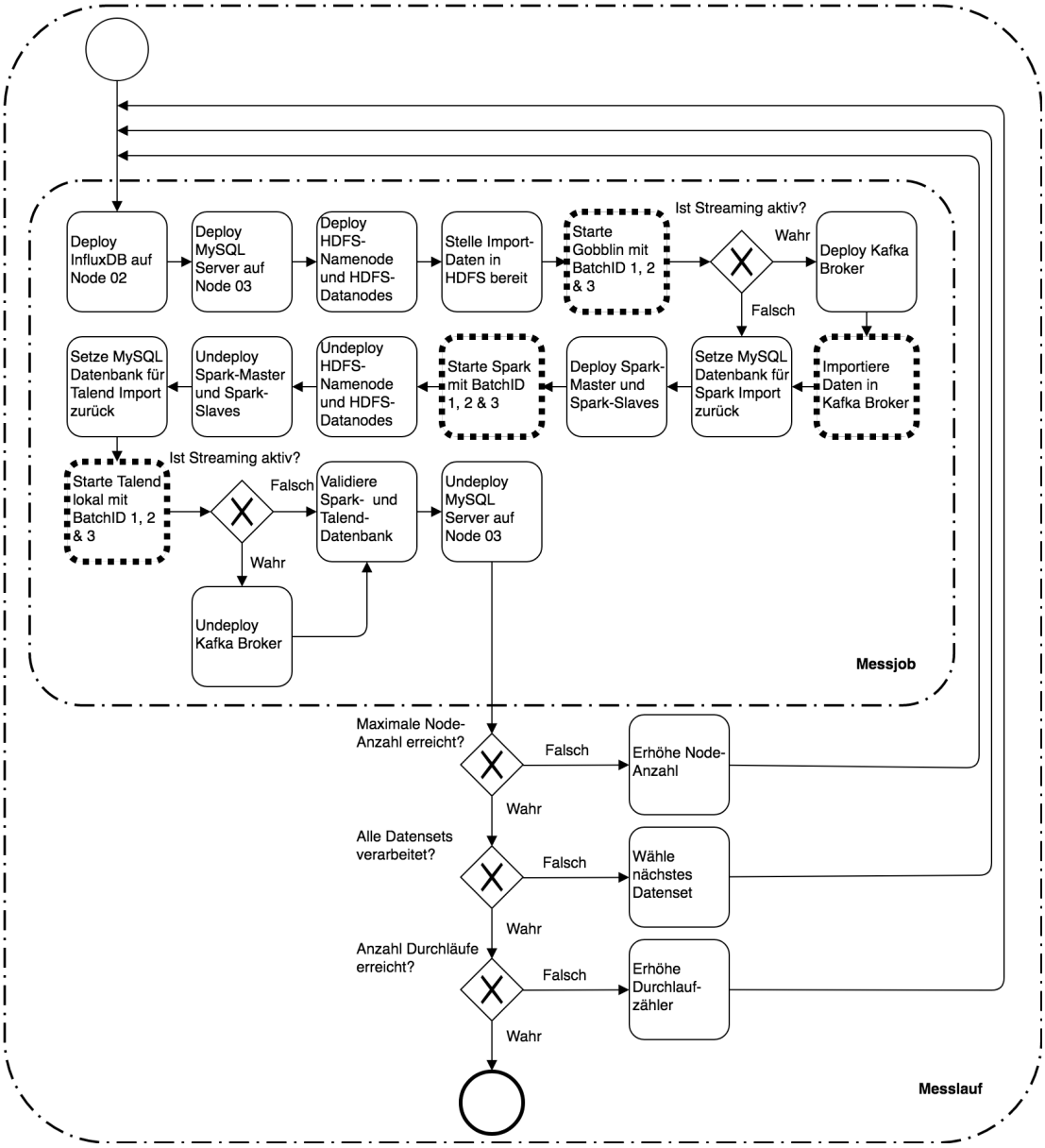

Abbildung 4.1.: Ablauf der Messung

Anschließend wird ein MySQL-Server deployt, der für die Importjobs als Datensenke dient. Der Datenbankserver verfügt über zwei Datenbanken: eine als Datensenke für die Gobblin-Spark-Importjobs und die andere Datenbank als Datensenke für die Talend-Importjobs.

Als erstes wird der Gobblin-Spark-Import ausgeführt. Dazu werden zunächst der HDFS-Cluster deployt und die Quelldaten im HDFS-Cluster bereitgestellt. Anschließend werden die Gobblin-Importjobs mit den BatchIDs 1, 2 und 3 gestartet. Die Gobblin-Jobs verarbeiten die Quelldateien mit ihren unterschiedlichen Dateiformaten zu Dateien im *Apache Avro*-Format[1]. Durch das transformierte homogene Format der Quelldateien, werden die Spark-Importjobs um das Lesen unterschiedlicher Dateiformate entlastet.

Falls im Messjob ein Event-Stream als Quelle verwendet werden soll, wird ein Kafka-Broker im Cluster bereitgestellt. Ein Gobblin-Job importiert die Quelldatei dann in den Kafka-Broker.

Bevor der Spark-Importjob gestartet wird, wird die SQL-Datenbank auf dem MySQL-Server zurückgesetzt und mit dem SQL-Schema versehen. Anschließend wird ein Spark-Master deployt und abhängig von der Anzahl der beteiligten Nodes wird einer oder mehrere Spark-Slaves im Cluster deployt. Sobald der Spark-Cluster bereit ist, werden die Spark-Importjobs für die BatchIDs 1, 2 und 3 gestartet. Wenn die Spark-Importjobs beendet sind, wird der HDFS- und der Spark-Cluster entfernt.

Im Anschluss an die Spark-Messung, wird die Talend-Messung gestartet. Dazu wird die SQL-Datenbank auf dem MySQL-Server zurückgesetzt, mit dem SQL-Schema versehen und es werden die Talend-Importjobs für die BatchIDs 1, 2 und 3 gestartet. Alle Importjobs werden in sequentieller Reihenfolge ausgeführt. Es würden sich einige Importjobs parallel voneinander ausführen lassen, das würde allerdings die Messung der Ressourcennutzung der einzelnen Jobs verfälschen. Das Parallelisieren wäre nur hilfreich, wenn man die Gesamtdauer eines Imports reduzieren möchte, was im Rahmen dieser Messung nicht gewünscht ist.

Falls im Messjob ein Event-Stream als Quelle verwendet wurde, wird der Kafka-Broker zu diesem Zeitpunkt undeployt.

Nachdem die Importjobs beider Frameworks durchgelaufen sind, werden die Datensenken des Spark- und Talend-Imports validiert. Die Validierung erfolgt, in dem die Zeilen-Anzahl der jeweiligen SQL-Tabellen miteinander verglichen wird. Falls die Zeilen-Anzahlen gleich sind, wird angenommen, dass die jeweiligen Importjobs der Framework logisch das gleiche Ergebnis in die Datensenke geschrieben haben. Nur wenn die Importjobs das gleiche Ergebnis berrechnet haben, sind sie und die dazugehörigen Messergebnisse miteinander vergleichbar. Die Messergebnisse eines Messjobs sind daher nur verwertbar, wenn die Validierung erfolgreich ist. Zum Schluss eines Messjobs wird der MySQL-Server undeployt.

[1] Apache-Foundation (2018a)

Der beschriebene Messablauf wird mehrfach durchgeführt. Jede Kombination aus der Anzahl beteiliger Nodes und Datenset-Größen wird drei mal durchlaufen.

Die einzelnen Messschritte werden nacheinander ausgeführt, damit jeder Importjob die vorhandenen Ressourcen konfliktfrei nutzen kann und die Messwerte nicht verfälscht werden.

In den Prozessschritten der Abbildung 4.1, die einen gepunkteten Rahmen haben, werden Messwerte erhoben. Das Deployment und Undeployment von Software-Komponenten wird nicht mitgemessen.

4.3.2. Einschränkung der Streaming-Importjobs

Bei einem Streaming-Job nutzen nicht alle Importjobs der TPC-DI Spezifikation eine Streaming-Quelle. Stattdessen wurde der Importjob, der die Datei *CustomerMgmt.xml* importiert, auf eine Streaming-Quelle angepasst. Diese Datei enthält bereits in XML-Form Events, die das Erstellen, Verändern und Deaktivieren von Kunden und deren Konten beschreiben. Die Datei gehört zu den verhältnismäßig größten Dateien, die für den historischen Import vorliegen (siehe Tabelle 3.1) und hat daher das Potenzial in den gemessenen Laufzeiten deutliche Unterschiede zwischen Streaming- und Non-Streaming-Imports aufzuzeigen. Dadurch, dass es den *CustomerMgmt.xml*-Import als Streaming- und als Non-Streaming-Import gibt, kann man die Messwerte der Imports miteinander vergleichen und es werden möglicherweise Unterschiede in den Importverhalten der jeweiligen Quellen aufgezeigt.

In den Streaming-Importjobs werden ausschließlich die Streaming-Variante des *CustomerMgmt.xml*-Imports und die Non-Streaming-Imports ausgeführt, die für den *CustomerMgmt.xml*-Import benötigt werden. Außerdem wird nur der historische Import durchgeführt. In den inkrementellen Imports sind zu wenig Daten enthalten, als dass man einen Unterschied in den Messwerten zwischen den Streaming- und Non-Streaming-Jobs erkennen kann. In Tabelle 4.1 sind die Imports mit der jeweiligen Quelle aufgelistet, die während eines Streaming-Importjobs laufen.

Tabelle 4.1.: Auflistung der Streaming-Importjobs

Importdateiname	Quelle
Date.txt	Dateisystem (wie Non-Streaming-Import)
TaxRate.txt	Dateisystem (wie Non-Streaming-Import)
HR.csv	Dateisystem (wie Non-Streaming-Import)
CustomerMgmt.xml	Streaming-Quelle

4.4. Parameter einer Messung

Jeder Messwert wird durch folgende Variablen parametrisiert und ist durch diese identifizierbar:

- Größe der Import-Daten
- Genutztes Framework
- Batch ID
- Wird ein Event-Stream als Quelle genutzt? Ja/Nein
- Anzahl beteiligter Nodes

Die Messungen werden mit mehreren Datensetgrößen durchgeführt, die detailliert in Abschnitt 4.6 beschrieben werden. Die Import-Daten wurden mit dem Tool *DIGen* generiert, das von der TPC bereitgestellt wird. Das Tool sorgt dafür, dass die Datensätze konsistent sind. Bei der Generierung der Daten lässt sich nicht die exakte Größe des Datensets konfigurieren, sondern man bestimmt anhand eines Skalierungsfaktors, wieviele Datensätze erstellt werden. Dadurch haben die generierten Input-Daten keine *glatten* Größen, sind aber in sich konsistent.

Bei den genutzten Frameworks wird unterschieden zwischen der Gobblin-Spark-Kombination und Talend. Wie in Abschnitt 4.3 erläutert, wird Talend ohne eine vor- oder nachgeschaltete Technologie in einem Messjob verwendet. Während hingegen Apache Gobblin und Apache Spark immer gemeinsam verwendet werden.

In jedem Messjob wird die Batch ID *1*, *2* oder *3* verwendet. Mit der ID *1* wird der historische Import gekennzeichnet, der bei einer Messung die Basis für alle folgenden inkrementellen Updates bildet. Um die Datenkonsistenz und Vergleichbarkeit mit anderen Messdurchläufen zu erhalten, muss die Reihenfolge der inkrementellen Imports in den Messdurchläufen beibehalten werden.

Wie in Abschnitt 4.3.2 erläutert, gibt es Non-Streaming-Messjobs, die als Datenquelle Dateien von einem Dateisystem lesen, und ein angepasster Streaming-Messjob, der als Datenquelle einen Event-Stream verwendet. Bei jedem Messjob wird daher ebenfalls parametrisiert, ob Streaming- oder Non-Streaming-Messjobs ausgeführt werden. Falls das Streaming aktiviert ist, wird ein Kafka-Broker auf einem der Nodes bereitgestellt und anschließend mit den Import-Daten befüllt.

Es wird außerdem die Anzahl der verarbeitenden Computer variiert. Aufgrund der Clustergröße kann dieser Parameter den Wert 1, 2 oder 3 annehmen. Das Tool Talend kann in der verwendeten OpenSource-Version nur lokal Daten verarbeiten und kann daher nicht von mehreren verarbeitenden Computern profitieren. Talend verfügt über kostenpflichtige

Enterprise-Lizenzen, mit denen man Talend-Jobs in einem Hadoop- oder Spark-Cluster betreiben kann. Auf Nachfrage bei der Firma Talend wurde für diese Arbeit keine Enterprise-Lizenz zur Verfügung gestellt, sodass Talend lediglich in der OpenSource-Version lokal betrieben werden kann. Das Framework Apache Spark profitiert von einer höheren Anzahl an Slave-Nodes, da Spark die Aufgaben auf die beteiligten Nodes verteilen und parallel verarbeiten kann. Wie in Abschnitt 4.3 verwendet Spark als Datenquelle das HDFS, das ebenfalls verteilt in einem Cluster betrieben werden kann. Spark profitiert davon, wenn auf jedem Spark-Slave-Node auch ein HDFS-Datanode verfügbar ist, der Input-Daten liefern kann. Daher skaliert der HDFS-Cluster analog zum Spark-Cluster und wird in den Messjobs auf den beteiligten Nodes bereitgestellt.

4.5. Komponenten im Cluster

Die einzelnen Software-Komponenten müssen auf die Cluster-Nodes aufgeteilt werden, sodass sie sich gegenseitig möglichst wenig einschränken. Es wird im Folgenden erläutert, über welche Hardware-Ausstattung der Cluster verfügt, welche Software-Komponenten es gibt, wie sie verteilt werden und die Arbeitsspeicher-Zuteilung auf den Cluster-Nodes konfiguriert ist.

4.5.1. Hardware-Ausstattung

Der Cluster, mit dem gemessen wird, besteht aus sechs Computern des Modells Fujitsu Celsius W550. Jeder Cluster-Node verfügt über folgende Hardware-Ausstattung:

- Intel Xeon E3-1225, 4 Cores @ 3,3GHz
- 31GB RAM
- 240GB SSD, ist eingehängt im Pfad /
- 1TB HDD, ist eingehängt im Pfad */media/hdd*
- 1 GBit/s Netzwerk-Interface

Jeder Cluster-Node verfügt physikalisch über 32GB RAM, wovon allerdings 1GB der Grafikkarte zugewiesen sind. Es stehen daher 31GB RAM für das Betriebssystem und alle darauf laufenden Systeme zur Verfügung.

4.5.2. Eingesetzte Softwarekomponenten

Die Messung wurde mit bestimmten Software-Versionen durchgeführt. Im Sinne der Nachvollziehbarkeit, werden die verwendeten Versionen mit einem Link zur Quelle im Folgenden aufgeführt:

- Apache Gobblin in Version 0.12.0-rc0, von Apache-Foundation (2018d)
- Apache Spark und Spark SQL in Version 2.2.1, von Apache-Foundation (2018j)
- Talend Open Studio for Big Data in Version 6.5.1, von Talend (2018b)
- Apache Hadoop in Version 2.8.1, von Apache-Foundation (2018e)
- Apache Kafka in Version 0.11.0.1, von Apache-Foundation (2018g)
- MySQL-Server in Version 5.7, von Oracle (2018)
- Telegraf in Version 1.5.2, von InfluxData (2018)

Alle Softwarekomponenten werden, falls nicht anders angegeben, in der Standardkonfiguration verwendet, die der Hersteller angibt. Für die Komponenten *Gobblin*, *Spark* und *Talend* wird der erlaubte Arbeitsspeicher angehoben, die Details dazu sind in Abschnitt 4.5.4 zu finden.

Der Standard-Replikationsfaktor von Speicherblöcken in einem HDFS-Cluster beträgt drei. Es gibt Messjobs mit weniger als drei HDFS-Datanodes, weshalb die Replikation mit dem Standard-Replikationsfaktor fehlschlagen würde. Daher wurde der Replikationsfaktor für Speicherblöcke in allen Messjobs auf eins gesetzt.

Standardmäßig legt der MySQL-Server für jedes Schlüsselfeld einer Tabelle einen Index an, der bei Abfragen zu diesem Feld unterstützt. Zusätzlich werden beim Zurücksetzen der Datenbanken (siehe Abschnitt 4.3.1) Indizes angelegt, die bei *aufwändigen Berechnungen* im Sinne der Charakterisierung C_{13} aus Abschnitt 3.5, unterstützen.

4.5.3. Verteilung der Softwarekomponenten

Abhängig von den gesetzten Messparametern werden die Systemkomponenten unterschiedlich auf den Nodes bereitgestellt.

In Abbildung 4.2 ist dargestellt, welche Systemkomponente auf welchem Cluster-Node bereitgestellt wird. Auf jedem Node läuft das Tool *Telegraf*[2], das die Hardware-Nutzung erfasst und an die InfluxDB auf Node *02* zur Speicherung weiterleitet.

[2] https://github.com/influxdata/telegraf

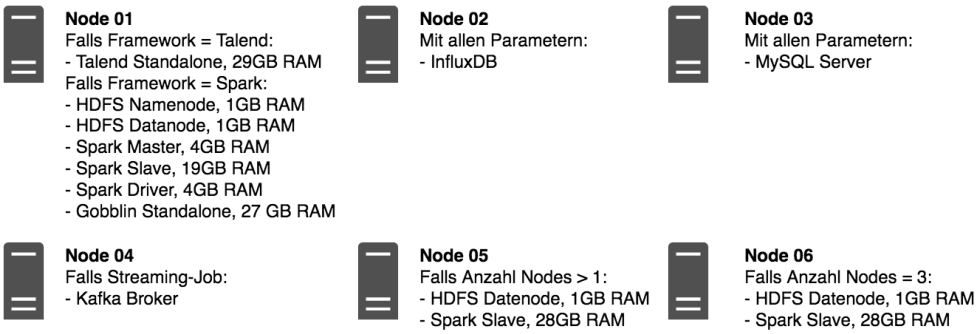

Abbildung 4.2.: Verteilung der System-Komponenten im Cluster

In allen Parametervariationen für das Spark-Framework läuft auf dem Node *01* der HDFS-Namenode, ein HDFS-Datanode, der Spark-Master, der Spark-Driver und ein Spark-Worker. In allen Parametervariationen für das Talend-Framework läuft auf dem Node *01* lokal Talend im Standalone-Modus.

Der Node *03* stellt einen MySQL-Server bereit, der für alle Spark- und Talend-Importjobs als Datensenke verwendet wird. In einem Streaming-Messjob wird auf Node *04* ein Apache Kafka bereitgestellt, der für den Streaming-Messjob als Datenquelle fungiert.

Falls der Messparameter *Anzahl beteiligter Nodes* größer als eins ist, werden die Nodes *05* und *06* verwendet um jeweils weitere Spark-Worker und HDFS-Datanodes bereitzustellen, die an der Verarbeitung beteiligt sind. Für Talend-Importjobs sind die Nodes *05* und *06* nicht relevant, da Talend in der OpenSource-Version nicht verteilt betrieben werden kann.

4.5.4. Zuteilung des Arbeitsspeichers

Jeder Node verfügt über 31GB RAM, von denen 2GB für das Betriebssystem reserviert werden. Es bleiben 29GB RAM, die den Software-Komponenten zugeteilt werden können.

Auf den Nodes *02*, *03* und *04* wird jeweils nur eine Komponente betrieben, der die 29GB RAM zugeteilt werden können.

Auf den Nodes *01*, *05* und *06* werden mehrere Komponenten betrieben, unter denen der Arbeitsspeicher aufgeteilt wird. Die RAM-Aufteilung für diese Nodes ist in Abbildung 4.2 ebenfalls dargestellt. Dort ist zu sehen, bei welchen Messparametern eine Komponente wieviel RAM zugewiesen bekommt. Wenn die Talend-Importjobs ausgeführt werden, ist keine andere Software-Komponente auf dem Node *01* deployt. Daher können die 29GB RAM dem Talend-Import zugeteilt werden.

Wenn die Spark-Importjobs ausgeführt werden, sind im Cluster HDFS- und Spark-Komponenten deployt, die den Arbeitsspeicher teilen. Auf dem Node *01* erhält der HDFS Namenode 1GB RAM, der HDFS Datanode 1GB RAM, der Spark Master 4GB RAM, der Spark Slave 19GB RAM, der Spark Driver 4GB RAM und die Gobblin-Importjobs 27GB RAM. Auf den Nodes *05* und *06* sind dem HDFS Datanode 1GB RAM und dem Spark Slave 28GB RAM zugeteilt.

4.6. Zu importierende Daten

In den Messläufen werden die zu importierenden Datensets variiert. Sie unterscheiden sich in der Anzahl der zu importierenden Zeilen und damit der Datengröße. Die Datensets wurden wie in Abschnitt 3.2 beschrieben mit dem Tool *DIGen* generiert.

Die Größe der einzelnen Datensets, wieviele zu verarbeitende Zeilen jede BatchID enthält und mit welchem Skalierungsfaktor sie von *DIGen* erstellt wurden, ist in Tabelle 4.2 aufgeführt.

Tabelle 4.2.: Größe der Datensets

			Zeilen in BatchID		
Datenset	DIGen Skalierungsfaktor	Größe insgesamt	BatchID 1	BatchID 2	BatchID 3
Pico	5	466MB	7804509	33380	33455
Nano	10	962MB	15980433	67451	67381
Micro	15	1,5GB	24158202	101709	101707
Klein	100	9,6GB	160873381	677582	677508
Mittel	200	20GB	321157481	1354323	1353860
Groß	500	49GB	801946413	3385083	3384856

In Tabelle 4.3 sind alle Import-Dateien aufgeführt mit ihren jeweiligen Größen in den einzelnen Batches und Datensets. Jede der aufgelisteten Dateien existiert einmal, bis auf die Finwire-Dateien. Es gibt pro Datenset 203 Finwire-Dateien, die im historischen Import verarbeitet werden. Die angegebene Größe ist die Summe der einzelnen Finwire-Dateien.

4.7. Metrikerfassung

Während die Importjobs die Importdaten verarbeiten, werden mehrere Metriken erfasst, die im Verlauf dieser Arbeit analysiert werden. Zur Metrikerfassung wird das Tool *Telegraf*[3] verwendet, das auf jedem Cluster-Node läuft und Messwerte erhebt. Telegraf meldet die erfassten Metriken anschließend an die *InfluxDB*, wo sie persistiert werden.

[3] https://github.com/influxdata/telegraf

Tabelle 4.3.: Größe der einzelnen Importdateien

Dateigröße							
Name	Batch ID	Pico	Nano	Micro	Klein	Mittel	Groß
Account.txt	2	3,5KB	7,1KB	11KB	75KB	150KB	388KB
Account.txt	3	3,4KB	7,2KB	11KB	75KB	150KB	389KB
CashTransaction.txt	1	53MB	105MB	157MB	1,1GB	2,1GB	5,2GB
CashTransaction.txt	2	33KB	63KB	104KB	674KB	1,4MB	3,4MB
CashTransaction.txt	3	32KB	64KB	105KB	682KB	1,4MB	3,4MB
CustomerMgmt.xml	1	15MB	30MB	45MB	299MB	597MB	1,5GB
Customer.txt	2	5KB	11KB	16KB	103KB	205KB	517KB
Customer.txt	3	5,2KB	11KB	16KB	103KB	205KB	518KB
DailyMarket.txt	1	136MB	296MB	455MB	3,0GB	6,0GB	15GB
DailyMarket.txt	2	227KB	494KB	762KB	5,1MB	11MB	26MB
DailyMarket.txt	3	227KB	495KB	761KB	5,1MB	11MB	26MB
Date.txt	1	3,4MB	3,4MB	3,4MB	3,4MB	3,4MB	3,4MB
FINWIRE*	1	46MB	103MB	160MB	1,1GB	2,3GB	5,2GB
HoldingHistory.txt	1	13MB	26MB	40MB	278MB	575MB	1,5GB
HoldingHistory.txt	2	9,4KB	22KB	35KB	242KB	491KB	1,3MB
HoldingHistory.txt	3	11KB	22KB	35KB	241KB	487KB	1,3MB
HR.csv	1	2MB	3,9MB	5,9MB	40MB	80MB	202MB
Industry.txt	1	2,7KB	2,7KB	2,7KB	2,7KB	2,7KB	2,7KB
Prospect.csv	1	5MB	10MB	15MB	101MB	201MB	503MB
Prospect.csv	2	5MB	10MB	15MB	101MB	201MB	502MB
Prospect.csv	3	5MB	10MB	15MB	101MB	201MB	502MB
StatusType.txt	1	89B	89B	89B	89B	89B	89B
TaxRate.txt	1	17KB	17KB	17KB	17KB	17KB	17KB
Time.txt	1	4,6MB	4,6MB	4,6MB	4,6MB	4,6MB	4,6MB
TradeHistory.txt	1	52MB	104MB	157MB	1,1GB	2,2GB	5,3GB
Trade.txt	1	62MB	124MB	186MB	1,3GB	2,5GB	6,4GB
Trade.txt	2	80KB	174KB	271KB	1,8MB	3,6MB	9,1MB
Trade.txt	3	85KB	170KB	269KB	1,8MB	3,6MB	9,1MB
TradeType.txt	1	99B	99B	99B	99B	99B	99B
WatchHistory.txt	1	67MB	134MB	202MB	1,4GB	2,7GB	6,8GB
WatchHistory.txt	2	194KB	386KB	582KB	3,9MB	7,9MB	20MB
WatchHistory.txt	3	194KB	386KB	582KB	3,9MB	7,9MB	20MB

Telegraf ist so konfiguriert, dass es seine Standard-Plugins nutzt, sodass folgende Metriken erhoben werden:

- CPU-Auslastung in Prozent
- freie und genutzte Kapazitäten auf der SSD und HDD
- Schreib- und Lesezugriffe auf die SSD und HDD
- freie und genutzte Arbeitsspeicher-Kapazitäten
- freie und genutzte Swap-Kapazitäten
- Nutzung der Netzwerk-Interfaces

Jeder Importjob erhebt außerdem die Start- und Endzeitpunkte, woraus die Laufzeit der Importjobs abgeleitet wird.

Die Messwerte der Gobblin- und Spark-Importjobs müssen gemeinsam betrachtet werden. Gobblin übernimmt in der Pipeline die Aufgabe des Lesens und Formatieren der Quelldateien, was die Talend-Importjobs auch durchführen. Um die Vergleichbarkeit zu gewährleisten, müssen die Gobblin- und Spark-Importjobs logisch als eine Import-Pipeline betrachtet werden.

In dem Abschnitt 4.3 wurde erläutert, in welchen Schritten eines Messjobs Messwerte erhoben werden. Bei der Erhebung der Start- und Endzeitpunkte wird die Start- und Vorbereitungszeit eines Frameworks nicht mitgemessen. Es wird zum Beispiel nicht das Starten der Java Virtual Machine (JVM) mitgemessen, die zum Ausführen der Talend-, Gobblin- und Spark-Jobs benötigt wird.

Bei den Spark-Importjobs wird allerdings das Scheduling der Spark-Tasks vom Spark-Master zum Spark-Slave mitgemessen. Es gibt keine anderen Jobs außer den aktuellen Importjobs, daher erfolgt das Scheduling innerhalb von wenigen Millisekunden, die die Messung vernachlässigbar länger erscheinen lässt.

4.8. Abweichung von der TPC-DI

Die Messung, die im Rahmen dieser Arbeit durchgeführt wird, entspricht keinem vollständigem Benchmark wie es die TPC-DI nach Transaction Processing Performance Council (2014) erwartet. Im Folgenden werden die Punkte aus Kapitel 3 aufgeführt, die diese Arbeit nicht behandelt.

In Abschnitt 3.6 werden die *Batch Validation Query* und die *Data Visibility Query* genannt. Diese Abfragen werden nicht durchgeführt, nicht nur weil, wie dort erwähnt, die Abfragen

nicht bereitgestellt wurden, sondern auch, weil diese Arbeit keine *Automatisierte Audit Phase* gemäß der TPC-DI durchführt. Es wird eine Audit-Phase durchgeführt, allerdings stellt diese sicher, dass die Talend- und Gobblin-Spark-Imports etwas vergleichbares tun. Damit sind die durchgeführten Messungen nur untereinander vergleichbar und nicht zwangsläufig mit offiziellen TPC-DI Messungen.

Laut Transaction Processing Performance Council (2014, S. 84) darf auf dem Messsystem nichts anderes laufen, außer der Datenintegrationsapplikation. Wie in Abschnitt 4.7 erwähnt, läuft auf allen beteiligten Systemen der Dienst *Telegraf*, der die Hardware-Ressourcennutzung erfasst. Die Messung der Ressourcennutzung ist allerdings von der TPC-DI Spezifikation nicht vorgesehen. Die Spezifikation erfasst als Metriken nur die Laufzeiten der Imports, wie in Abschnitt 3.7 erläutert.

Außerdem ist, wie in Kapitel 3.8 erläutert, eine Qualifikation der verwendeten Hard- und Softwarekomponenten erforderlich. Die Qualifikation wird von dieser Arbeit ebenfalls nicht abgedeckt.

Neben den Formalien der TPC-DI entsprechen zwei Punkte in der Datentransformation nicht der Spezifikation. Bei einer Aktualisierung eines Kunden im historischen Import, müssen alle zu ihm gehörenden Konten auf seinen neuen *Surrogate Key* aktualisiert werden. Diese Aktualisierung erfolgt nicht in den Talend-Importjobs, weil sie eine komplexe Umsetzung in Talend benötigt, für die mir der Zugang zu detailliertem Talend-Wissen fehlt. Wie in Abschnitt 3.5 erwähnt, gibt es einige Transformationen, in denen Änderungen eines Tages zusammengefasst werden. Das Aggregieren von Änderungen auf täglicher Basis erfolgt weder in der Talend- noch in der Spark-Implementation.

Zur Veröffentlichung einer TPC-DI-Messung muss, wie in Abschnitt 3.9 erläutert, ein abschließender Bericht angefertigt werden, damit das Messergebnis veröffentlicht werden kann. Da diese Messung den genannten Kriterien der TPC-DI-Spezifikation nicht gerecht wird, würde sie nicht veröffentlicht werden, daher wird kein FDR angefertigt. Dennoch sind viele Informationen, die der FDR beinhalten würde, in dieser Arbeit erklärt.

5. Messergebnisse

Nachdem die Importjobs in allen Parametervariationen beendet waren, konnten die Messergebnisse gesammelt und aufbereitet werden. Jeder Job speicherte seine Start- und Endzeitpunkte in einer separaten Datei. Anhand der Zeitpunkte konnte die Laufzeit jeddes Jobs ermittelt werden und die erfasste Hardware-Nutzung konnte aus der InfluxDB abgefragt werden.

Einige der gesammelten Daten wurden visuell aufbereitet, um das Lesen der Daten zu vereinfachen. Die Daten, die für die Visualisierungen genutzt wurden, sind außerdem im Anhang aufgeführt. Dieser Abschnitt listet die Ergebnisse lediglich auf und analysiert sie nicht. Die Analyse und Diskussion der Messwerte findet im nachfolgenden Kapitel 6 statt.

Wie in Abschnitt 4.6 wurde erläutert, dass die Messungen mit sechs Datensets durchgeführt wurden, die unterschiedliche Größen haben. Ein Messlauf mit dem Datenset *micro* hätte circa drei Wochen gedauert, sodass die Zeit fehlte, um Messungen mit größeren Datensets durchzuführen. Es liegen daher nur Messergebnisse mit den Datensets *pico*, *nano* und *micro* vor. Die Messung mit dem Datenset *micro* wurde aufgrund des Zeitmangels lediglich einmal durchgeführt.

Im Folgenden werden zunächst die Messergebnisse der Laufzeiten und Hardware-Nutzung aufgeführt. Die Ergebnisse werden jeweils nach Non-Streaming- und Streaming-Jobs unterteilt.

5.1. Laufzeiten

Zunächst werden die Laufzeiten der Importsjobs betrachtet. In der Talend-Implementierung wurden die Importjobs der Datenbanktabellen *Company*, *Security* und *Financial* zu dem Job *FinwireImport* zusammengefasst. Dies ist damit begründet, dass diese drei Tabellen ihren Inhalt aus denselben Quelldateien beziehen. Um die Vergleichbarkeit zwischen den Talend- und Spark-Importjobs herzustellen, wurden auch für die Spark-Jobs diese drei Datenbanktabellen ebenfalls zu dem Job *FinwireImport* zusammengefasst.

5.1.1. Non-Streaming-Imports mit einem Node von Talend & Gobblin-Spark

Im Folgenden werden die Messwerte beider Messkandidaten präsentiert, die mit der Clustergröße 1 in den Non-Streaming-Importjobs erreicht wurden.

Die Importjob-Laufzeiten für beide Messkandidaten sind pro Datensetgröße in den Abbildungen 5.1, 5.2 und 5.3 dargestellt. Dort sind jeweils in der oberen Hälfte die Laufzeiten der historischen Importjobs und in der unteren Hälfte die inkrementellen Importjobs gegenübergestellt. Auf der x-Achse sind die Importjobnamen festgehalten, während auf der y-Achse der Laufzeitendurchschnitt mit einer logarithmischen Skala aufgetragen ist. Die dargestellten Daten sind bei der Verifikation der Hypothese A hilfreich.

Der Durchschnitt der Laufzeiten wurde bei den historischen Imports pro Job über alle Messläufe gemittelt. Bei den inkrementellen Imports ist die Laufzeit pro Job über alle Messläufe und die BatchIDs 2&3 gemittelt.

Die Messwerte der Gobblin-Spark-Variante sind in blau, während die Talend-Messungen in grün dargestellt sind. Bei einigen Messungen, wie zum Beispiel dem historischen Talend-IndustryImport, fällt auf, dass die gemessene Zeit scheinbar 0 Sekunden beträgt. Tatsächlich liegt die gemessene Zeit für so einen Job unter 100 Millisekunden. Aufgrund des verwendeten Auswertungsverfahrens gibt es an diesen Stellen ungenaue Messwerte.

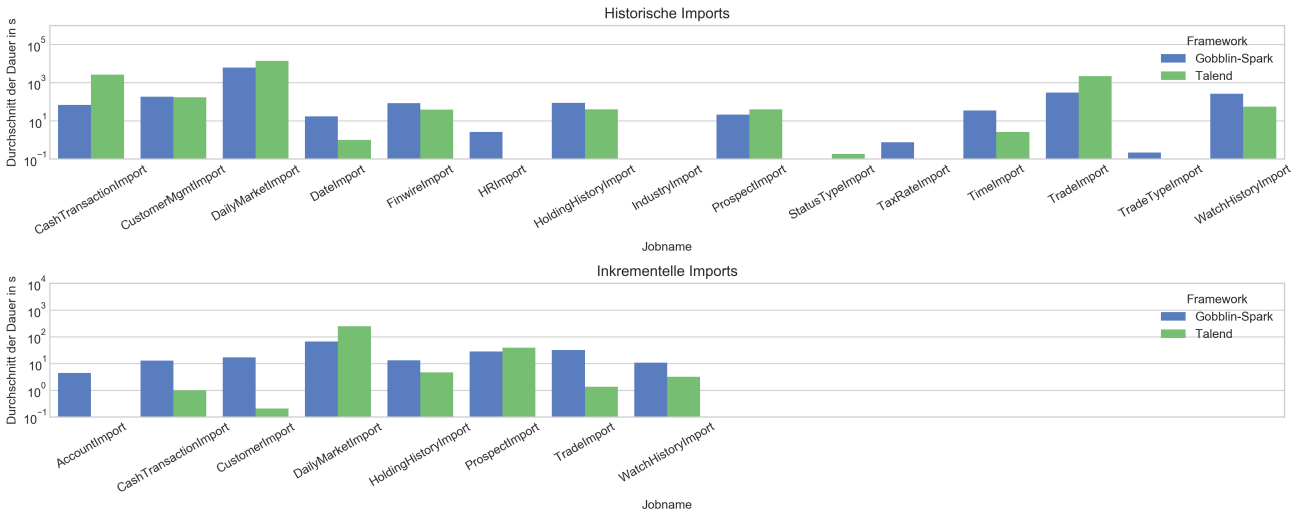

Abbildung 5.1.: Laufzeiten der Non-Streaming-Importjobs im Datenset *pico* mit der Clustergröße 1, gemäß der Tabellen A.1 und A.2

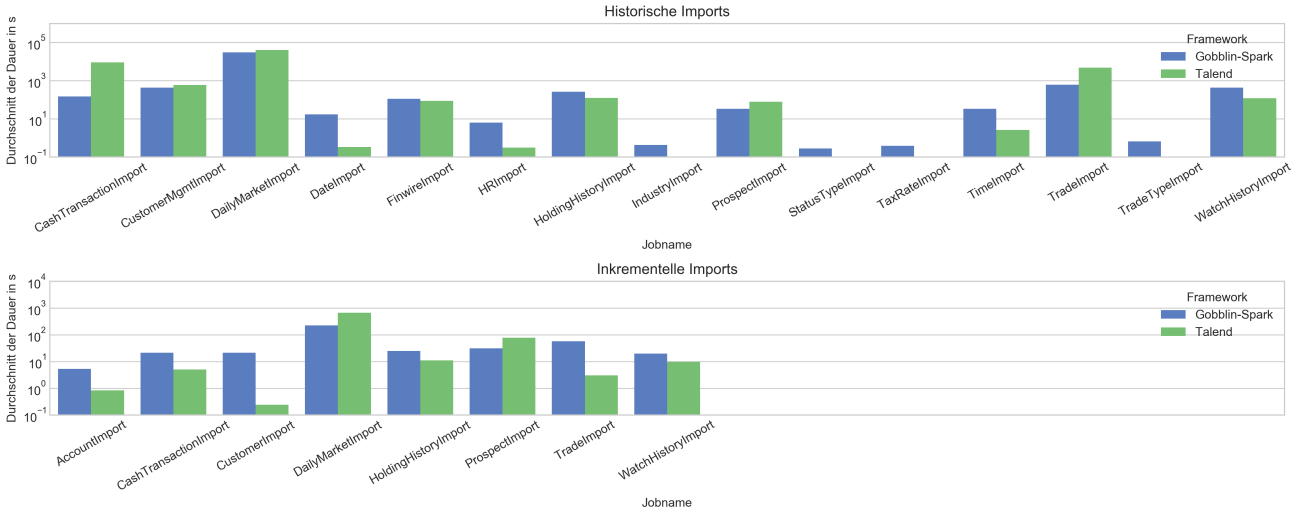

Abbildung 5.2.: Laufzeiten der Non-Streaming-Importjobs im Datenset *nano* mit der Clustergröße 1, gemäß der Tabellen A.3 und A.4

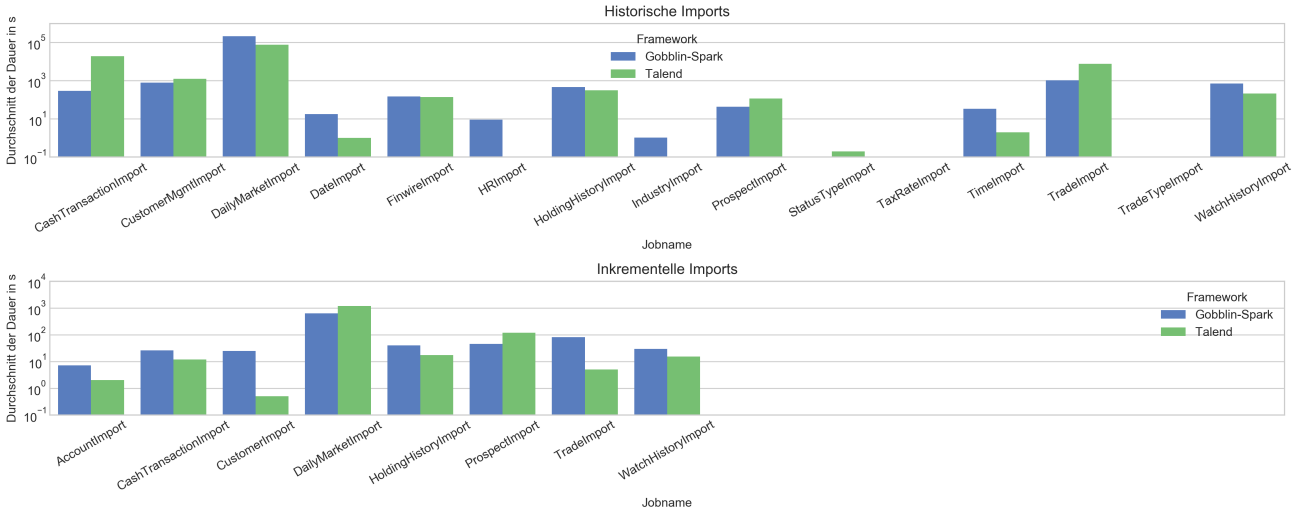

Abbildung 5.3.: Laufzeiten der Non-Streaming-Importjobs im Datenset *micro* mit der Clustergröße 1, gemäß der Tabellen A.5 und A.6

Summiert man die durchschnittlichen Importlaufzeiten pro Datenset, Framework und BatchID, dann erhält man die durchschnittliche Gesamtlaufzeit eines historischen oder inkrementellen Imports. Diese Zahlen sind in Abbildung 5.4 dargestellt. Auf der y-Achse ist die durchschnittliche Gesamtlaufzeit in Sekunden und auf der x-Achse die Datensetgröße aufgetragen. Die blauen Balken stellen die erzielten Werte der Gobblin-Spark-Imports dar, während die grünen Balken die Messwerte der Talend-Imports repräsentieren. Die Daten der Abbildung 5.4 dienen der Überprüfung der Hypothese B.

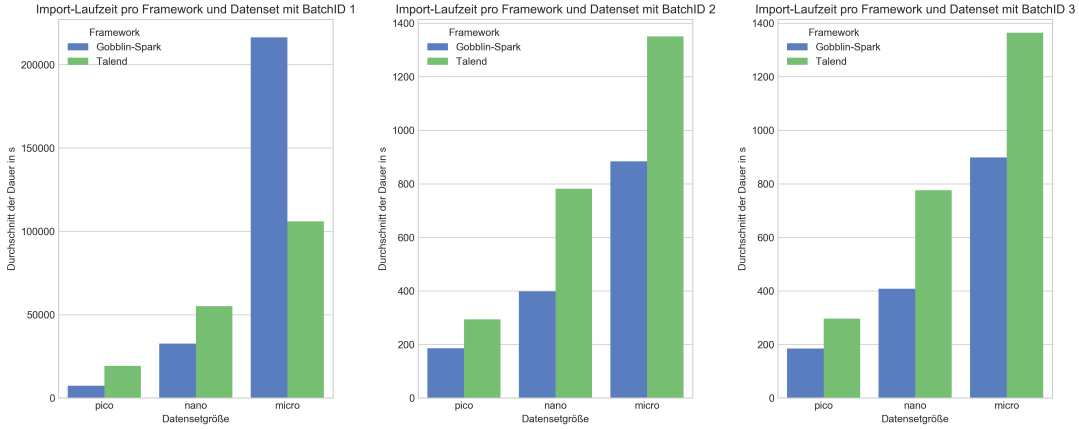

Abbildung 5.4.: Laufzeitsummen der Non-Streaming-Importjobs mit der Clustergröße 1, gemäß der Tabelle A.7

5.1.2. Non-Streaming-Imports von Gobblin-Spark in allen Clustergrößen

Die Gobblin-Spark Importjobs wurden mit drei Clustergrößen durchgeführt. Die gemessenen Laufzeiten sind pro Datenset in den Abbildungen 5.5, 5.6 und 5.7 abgebildet. Sie sind ebenfalls so zu lesen, wie die bisherigen Laufzeitabbildungen. In diesen Abbildungen ist pro Balken eine Clustergröße dargestellt: die blauen Balken bildet die Laufzeitwerte der Clustergröße 1 ab, die grünen Balken die Werte der Clustergröße 2 und die roten Balken die Werte der Clustergröße 3. Die Daten dieses Abschnitts werden zur Verifikation der Hypothese C verwendet.

5.1.3. Streaming-Importjobs

Wie in Abschnitt 4.3.2 beschrieben, unterscheidet sich der Messablauf eines Streaming-Imports von dem eines Non-Streaming-Import darin, dass einer der Importjobs modifiziert wurde. In den nachfolgenden Diagrammen werden daher nur die Laufzeiten des modifizierten CustomerMgmt-

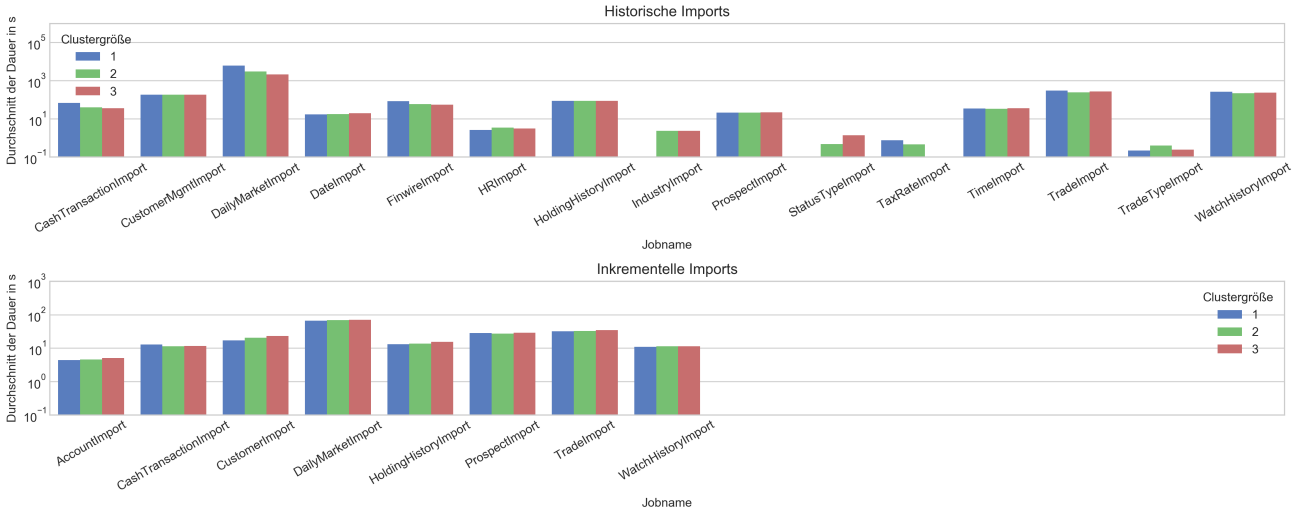

Abbildung 5.5.: Laufzeit aller Non-Streaming-Imports aller Clustergrößen mit Gobblin-Spark-Framework im Datenset *pico*, gemäß der Tabellen A.8 und A.9

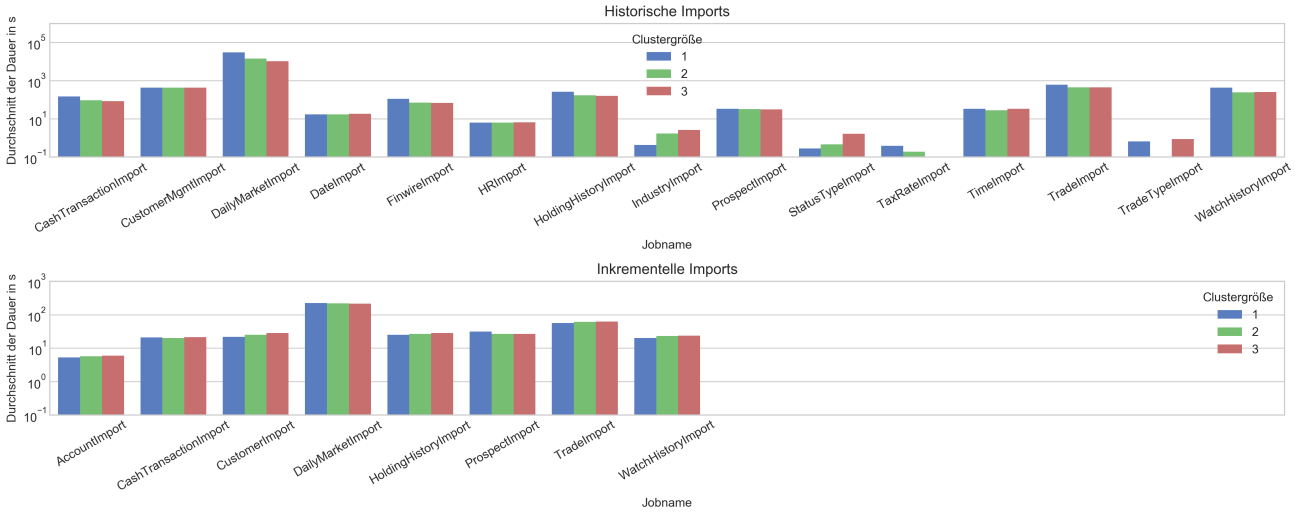

Abbildung 5.6.: Laufzeit aller Non-Streaming-Imports aller Clustergrößen mit Gobblin-Spark-Framework im Datenset *nano*, gemäß der Tabellen A.10 und A.11

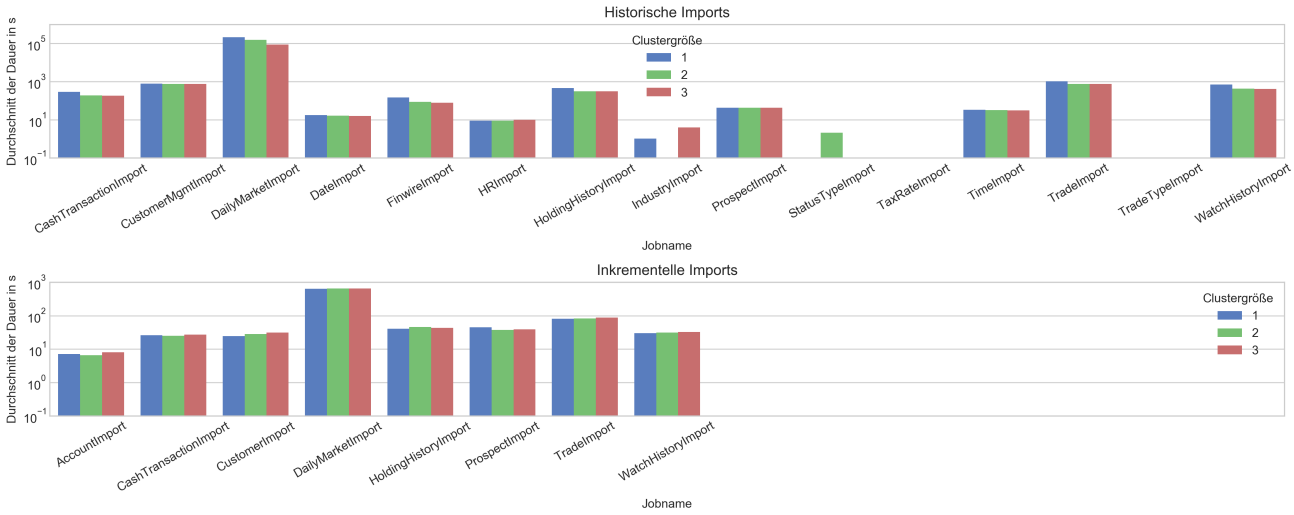

Abbildung 5.7.: Laufzeit aller Non-Streaming-Imports aller Clustergrößen mit Gobblin-Spark-Framework im Datenset *micro*, gemäß der Tabellen A.12 und A.13

Importjobs gezeigt. Es wurden nur die historischen Streaming-Importjobs durchgeführt, weshalb keine Messergebnisse für inkrementelle Imports vorliegen.

In der Abbildung 5.8 sind die Laufzeiten des Streaming-Importjobs *CustomerMgmtByStreamImport* abgebildet. Auf der x-Achse sind die Datensetgrößen aufgetragen, während auf der y-Achse die Laufzeit in Sekunden abzulesen ist. Die Laufzeit entspricht dem gemittelten Wert aus drei Messungen. In der linken Hälfte der Abbildung werden die Messungen mit der Clustergröße 1 und beiden Framework-Varianten verglichen. Die Messwerte des Talend-Frameworks sind in grün und die Werte der Gobblin-Spark-Kombination in blau dargestellt. In der rechten Hälfte sind die Gobblin-Spark-Messwerte der unterschiedlichen Clustergrößen gezeigt, die Clustergröße 1 wird mit blauen Balken, die Clustergröße 2 mit grünen Balken und die Clustergröße 3 mit roten Balken abgebildet.

5.2. Hardware-Nutzung

Während der Importjobs wurden, wie in Abschnitt 4.7 beschrieben, Metriken zur Hardware-Ressourcen-Nutzung erfasst. Um die Übersichtlichkeit der Ergebnisse gewährleisten zu können, werden im Folgenden nicht alle Metriken von allen Importjobs dargestellt. Stattdessen werden die Metriken einiger Jobs beispielhaft dargestellt.

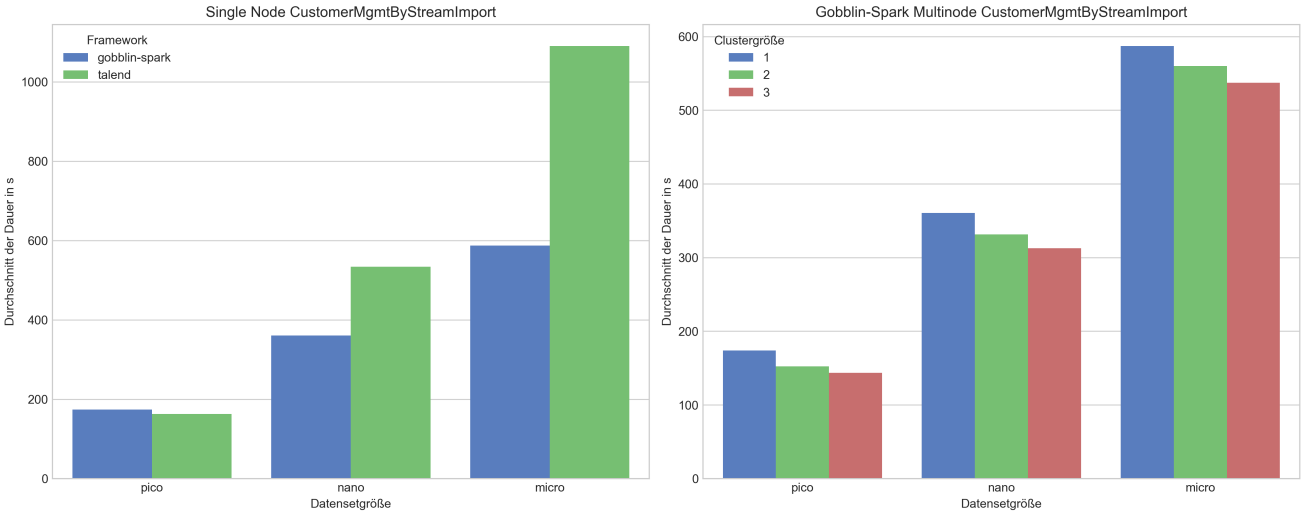

Abbildung 5.8.: Laufzeiten des Streaming-Importjobs *CustomerMgmtByStreamImport*, gemäß der Tabellen A.14 und A.15

Die Messwerte der unterschiedlichen Datensets weisen ähnliche Kurvenverläufe auf, die sich abhängig von der Importdauer strecken oder stauchen. Wegen des verwendeten Messverfahrens werden bei lang dauernden Messjobs mehr Messpunkte erfasst als bei Jobs, die kurz dauern. In diesem Abschnitt werden die Messwerte aus den Messungen mit der Zählnummer 1 und dem Datenset *nano* dargestellt. Bei diesen Messungen sind die Kurven am besten lesbar, da mehr Messpunkte vorhanden sind und die Kurven nicht zu sehr gestaucht sind.

Zusätzlich zu den Messwerten der historischen Imports, werden Nutzungswerte der inkrementellen Imports mit der BatchID 2 dargestellt.

In Abschnitt 5.1 wurden zu jedem Diagramm die dargestellten Werte tabellarisch im Anhang bereitgestellt. Die Werte der folgenden Abbildungen beruhen auf Daten, die sekündlich erfasst wurden. Es wird darauf verzichtet, diese Daten ebenfalls im Anhang bereitzustellen, da sie zu umfangreich sind. Würde man die sekündlich erfassten Daten reduzieren auf beispielsweise minütliche Werte, dann würden mögliche Spitzen und Tiefen fälschlicherweise entfernt werden. Die Daten wären in dem Fall nicht aussagekräftig. Die Messwerte der Hardware-Nutzung beschränken sich daher auf die folgenden Abschnitte.

5.2.1. Non-Streaming-Importjobs

Die Hardware-Nutzung wird beispielhaft anhand der ImportJobs *CustomerMgmtImport*, *DailyMarketImport* und *HoldingHistoryImport* gezeigt. Für jeden Job wird die CPU- und RAM-Nutzung jeweils für den historischen Import und, falls vorhanden, auch für den inkrementellen Import dargestellt. Für jede Framework- und Clustergrößenkombination gibt es jeweils eine Visualisierung. Es wurden diese drei Jobs ausgewählt, da unterschiedliche Anforderungen (siehe Abschnitt 3.5) an sie gestellt werden. Damit bieten sie die Grundlage für die anschließende Analyse.

In den Abbildungen 5.9, 5.10, 5.11 und 5.12 sind die Messergebnisse des historischen Importjobs *CustomerMgmtImport* dargestellt. In den Diagrammen A.1, A.2, A.3 und A.4 ist die Hardware-Nutzung des historischen Importjobs *DailyMarketImport* visualisiert. Die Messwerte der inkrementellen Variante sind in den Schaubildern A.5, A.6, A.7 und A.8 veranschaulicht. Die Abbildungen A.9, A.10, A.11 und A.12 beinhalten die Messwerte des historischen Imports *HoldingHistoryImport*, während die Abbildungen A.13, A.14, A.15 und A.16 die Messwerte desselben inkrementellen Jobs enthalten. Die Nutzungsdiagramme des Jobs *CustomerMgmtImport* sind in diesem Abschnitt zu finden. Die Diagramme der anderen beiden Jobs sind, um die Übersichtlichkeit des Abschnitts zu erhalten, im Anhang zu finden.

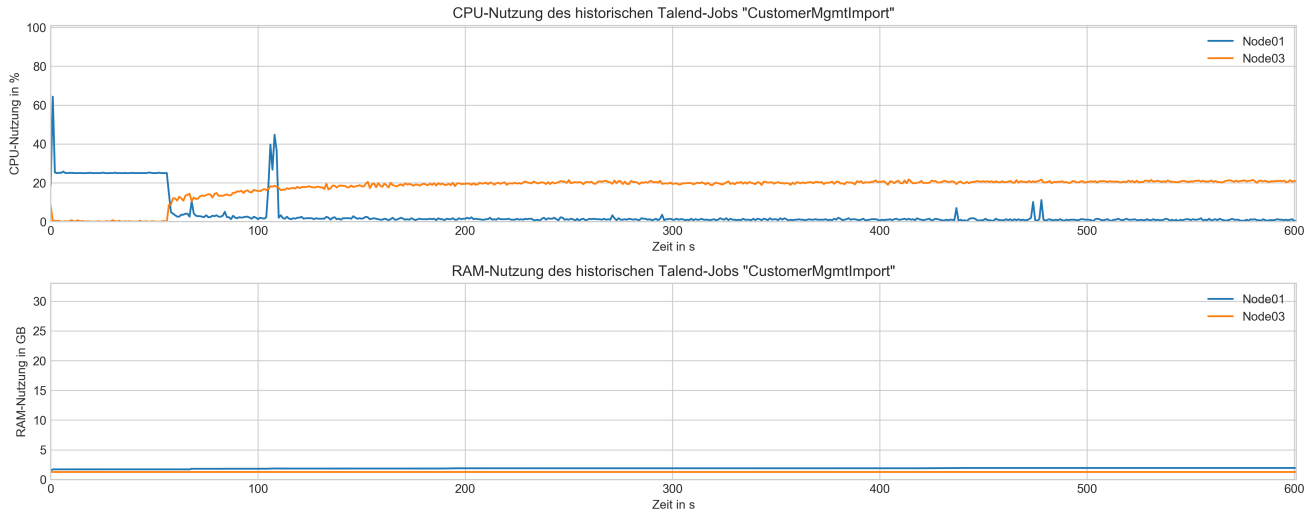

Abbildung 5.9.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *CustomerMgmtImport* mit Talend-Framework im Datenset *nano* mit der Clustergröße 1

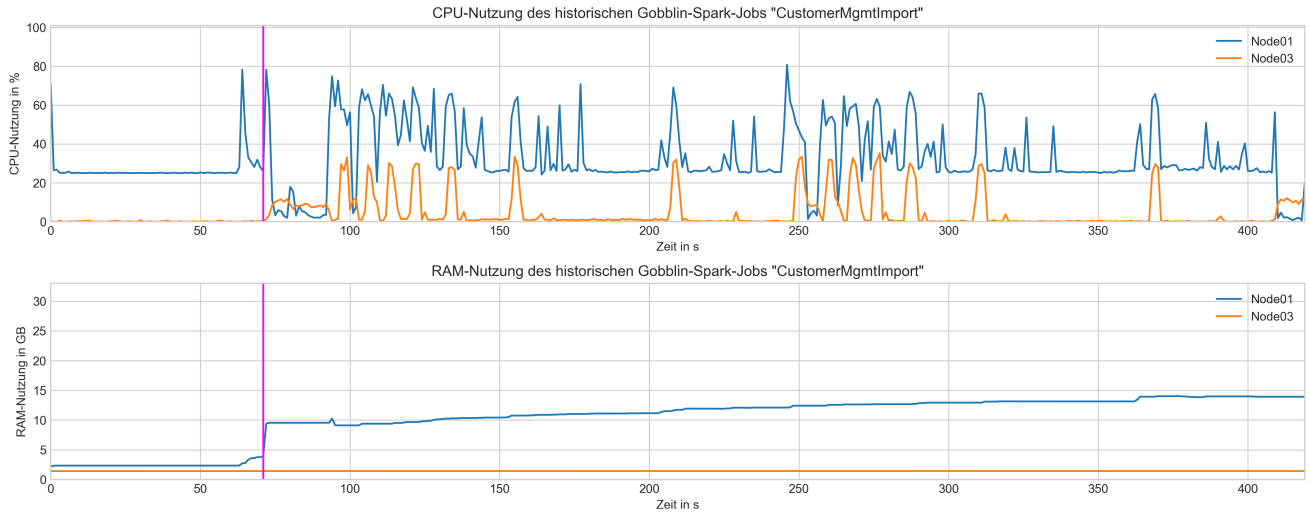

Abbildung 5.10.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *CustomerMgmtImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

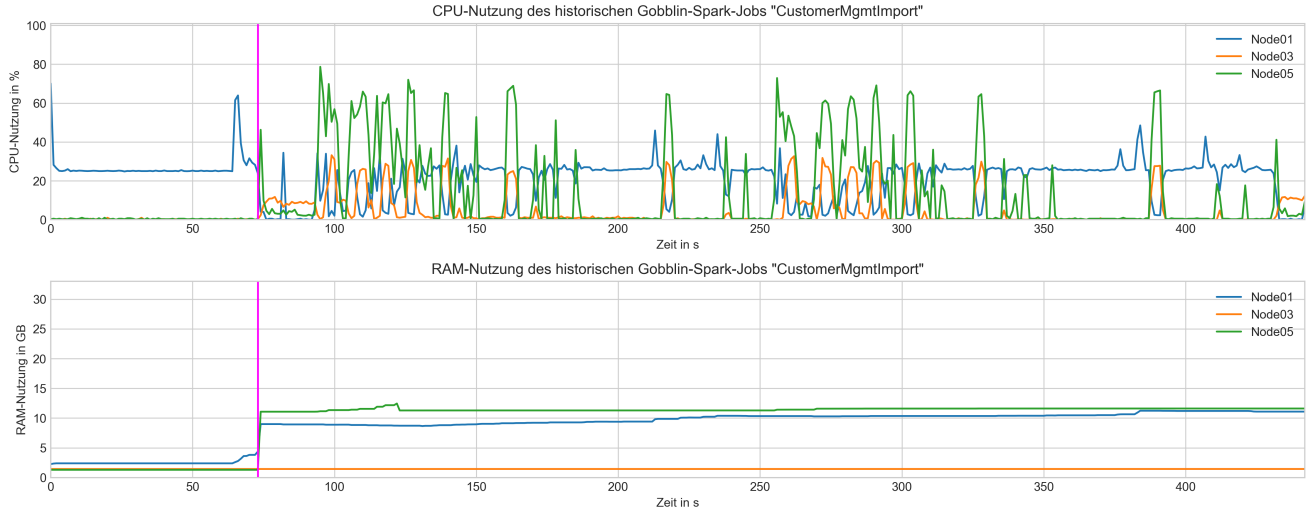

Abbildung 5.11.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *CustomerMgmtImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

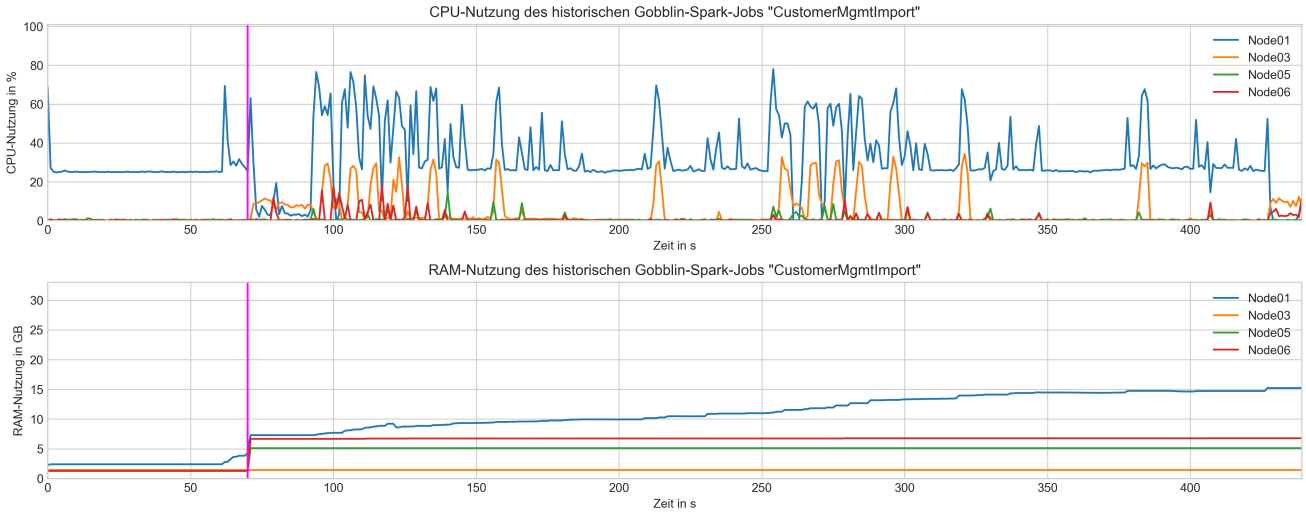

Abbildung 5.12.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *CustomerMgmtImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

Jede Abbildung besteht jeweils aus zwei Diagrammen. Das obere Diagramm trägt auf der y-Achse die CPU-Nutzung (in Prozent) der beteiligten Nodes, während auf dem unteren Diagramm an der y-Achse die Arbeitsspeichernutzung (in GB) angebracht ist. Auf der x-Achse ist jeweils die vergangene Zeit in Sekunden abgetragen. Für jeden beteiligten Node gibt es in den Diagrammen eine Kurve. In allen Abbildungen wird die Hardware-Nutzung des Nodes *01* mit einer blauen Kurve, die Nutzung des Nodes *03* mit einer orangen Kurve, die Nutzung des Nodes *05* mit einer grünen Kurve und die Nutzung des Nodes *06* mit einer roten Kurve dargestellt.

In vielen Abbildungen der Messwerte, die mit der Gobblin-Spark-Kombination erreicht wurden, ist außerdem eine senkrechte purpurfarbene Linie dargestellt. Diese Linie trennt die Messungen des Gobblin- vom Spark-Anteil. Die x-Werte, die sich links von der Trennlinie befinden, wurden während der Ausführung von Gobblin gemessen. Die Werte rechts von der Trennlinie wurden entsprechend bei der Spark-Ausführung aufgezeichnet. In einigen Gobblin-Spark Diagrammen, wie zum Beispiel in A.14, ist diese Trennlinie nicht dargestellt. In solchen Messungen wurde für den Gobblin-Teil eine Dauer von weniger als einer Sekunde gemessen. Da die Hardware-Metriken sekündlich erfasst werden, stehen bei diesen Jobs keine Nutzungsmetriken für den Gobblin-Teil zur Verfügung.

5.2.2. Streaming-Importjobs

Während der Messungen der Streaming-Importjobs, wurden Messdaten für den modifizierten Job *CustomerMgmtByStream Import* mit der BatchID 1 erhoben. Dies ist der einzige Job, der sich von den Non-Streaming-Imports unterscheidet. Daher werden nur die Messwerte für diesen Job dargestellt. Mit diesen Messdaten kann die Hypothese E bewertet werden.

Wie im vorangegangen Abschnitt auch, enthält jede Abbildung die CPU- und Arbeitsspeicher-Nutzung des historischen Imports. Die Hardware-Nutzung des Streaming-Jobs ist in den Abbildungen 5.13, 5.14, 5.15 und 5.16 dargestellt.

Die Abbildungen sind genauso zu lesen, wie auch die vorangegangenen Abschnitt: Jede Abbildung besteht aus zwei Diagrammen. In der oberen Hälfte ist auf der y-Achse die CPU-Nutzung in Prozent angebracht, während in der unteren Hälfte die Arbeitsspeichernutzung in GB auf der y-Achse abzulesen ist. Auf der x-Achse beider Diagramme ist die vergangene Zeit in Sekunden abgetragen. Jede Kurve stellt die Hardware-Nutzung eines Nodes dar. Die blaue Kurve zeigt die Messwerte des Nodes *01*, die orange Kurve die Werte des Nodes *03*, die grüne Kurve die Werte des Nodes *04*, die rote Kurve die Werte des Nodes *05* und die purpurne Kurve die Werte des Nodes *06*.

Im Vergleich zum vorangeganen Abschnitt enthalten die Streaming-Gobblin-Spark Diagramme keine purpurfarbene Trennlinie. Dies liegt daran, dass in der Streaming-Variante kein Gobblin-Job am Import beteiligt ist.

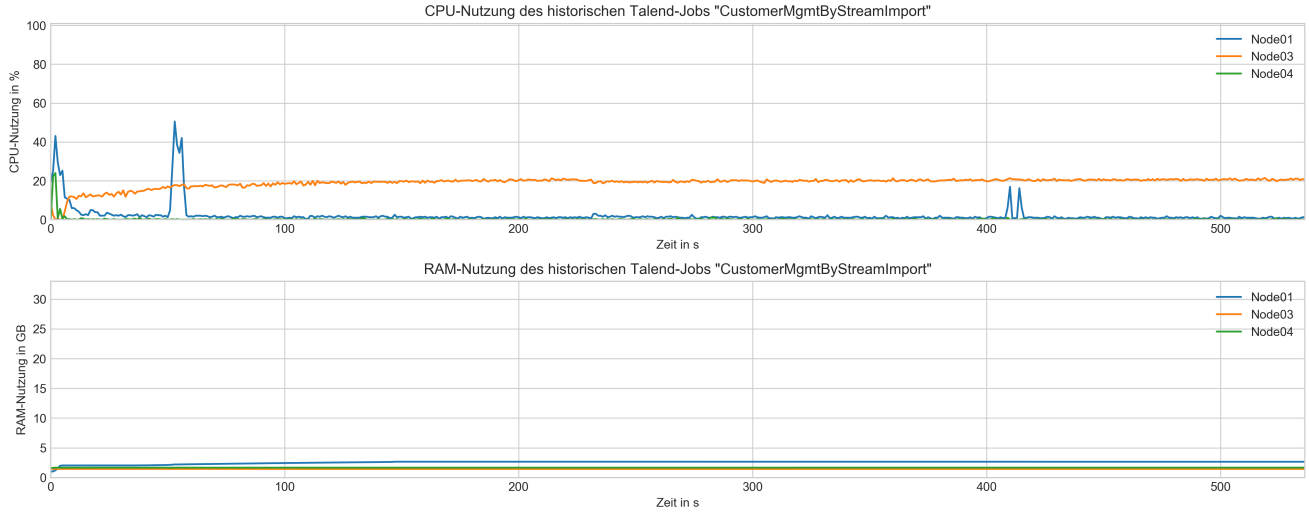

Abbildung 5.13.: Hardware-Nutzung des historischen Streaming-Importjobs *CustomerMgmtByStreamImport* mit Talend-Framework im Datenset *nano* mit der Clustergröße 1

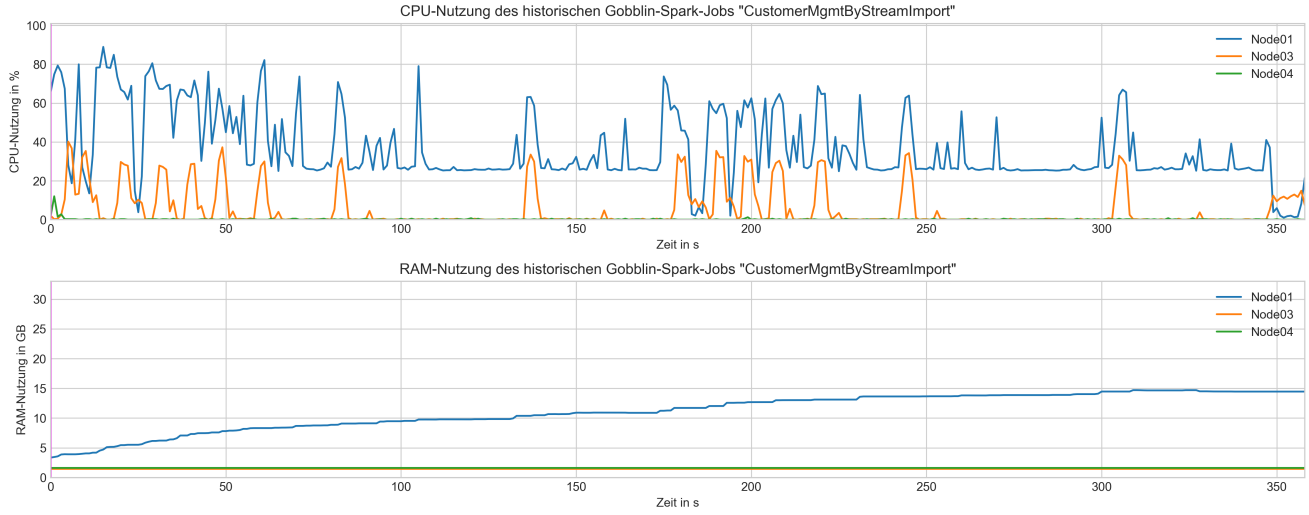

Abbildung 5.14.: Hardware-Nutzung des historischen Streaming-Importjobs *CustomerMgmtByStreamImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

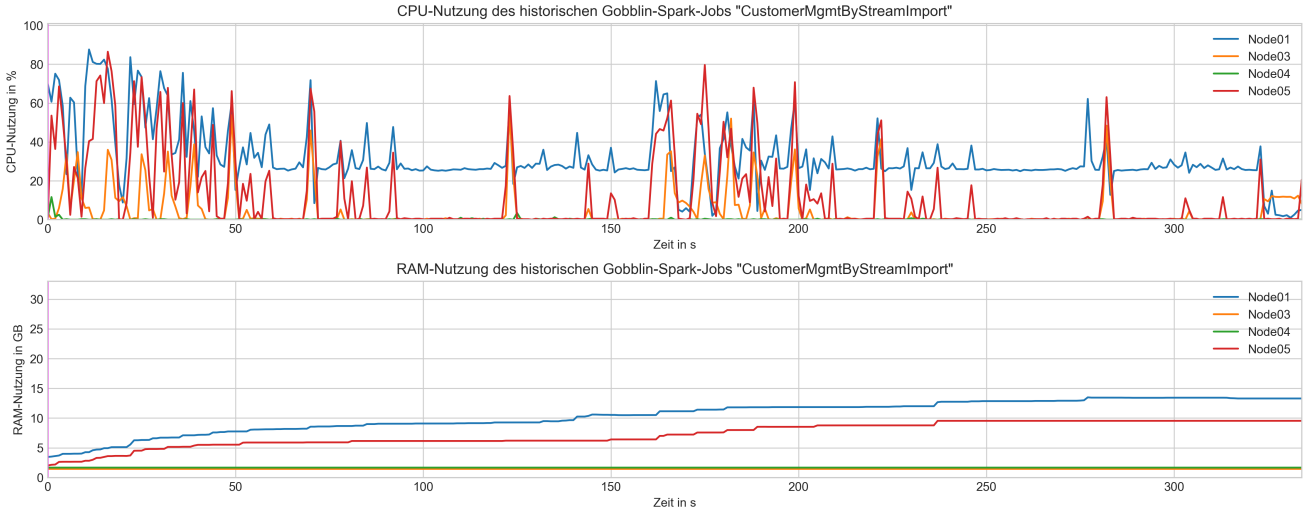

Abbildung 5.15.: Hardware-Nutzung des historischen Streaming-Importjobs *CustomerMgmtByStreamImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

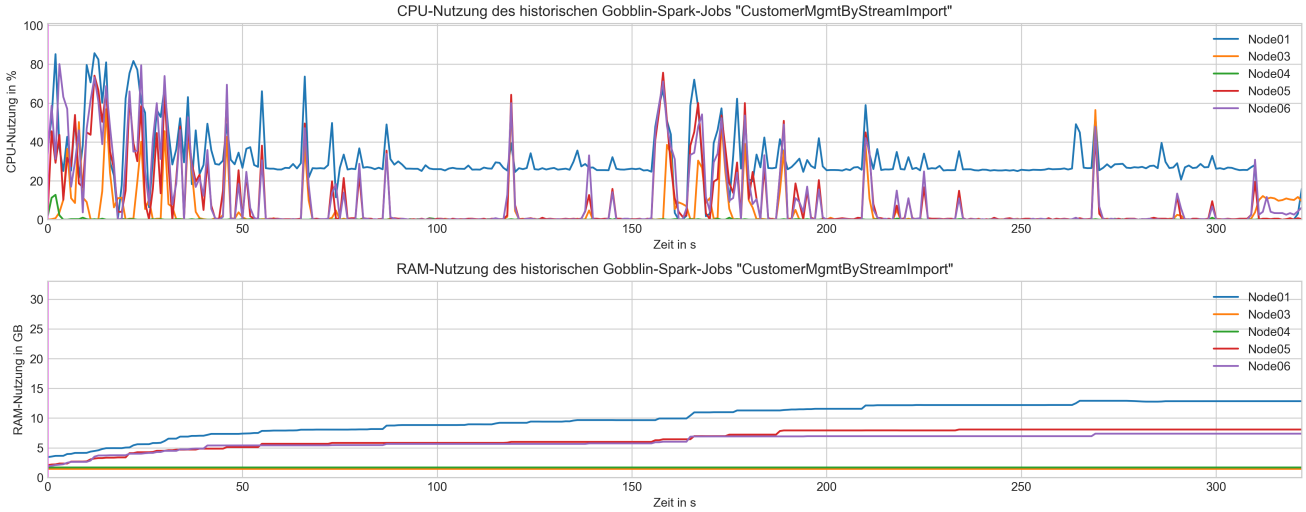

Abbildung 5.16.: Hardware-Nutzung des historischen Streaming-Importjobs *CustomerMgmtByStreamImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

6. Ergebnisanalyse

In diesem Kapitel wird die gemessenen Daten aus über 50 Experimenten im Detail analysiert. Folgenden werden aus den gemessenen Daten TPC-DI-Metriken gebildet, die anschließend mit den Ergebnissen von Bleuel (2016) verglichen werden. Außerdem werden die Laufzeiten und die erfasste Hardware-Nutzung im Detail analysiert. Abschließend werden die Hypothesen dieser Arbeit veri- oder falsifiziert.

6.1. TPC-DI Metriken

In Abschnitt 5.1.1 wurden die Laufzeiten jedes Frameworks pro Datenset und BatchID aufgeführt. Anhand der gemessenen Werte können die *Performance Metrik* und die *Preis-Performance-Metrik* der TPC-DI Spezifikation ermittelt werden. Dazu werden die exakten Laufzeit-Werte aus Tabelle A.7 herangezogen.

In den Tabellen 6.1 und 6.2 sind die TPC-DI Metriken und deren Formelzeichen jeweils für die Talend- und die Gobblin-Spark-Messung mit der Clustergröße 1 aufgeführt. Jede Spalte stellt jeweils ein Formelzeichen dar. Die Anzahl der Zeilen eines Datensets (R_H, R_{I1} und R_{I2}) wurden aus der Tabelle 4.2 entnommen.

Tabelle 6.1.: TPC-DI Metriken über alle Datensets für die Talend-Implementation mit Clustergröße 1

Datenset	E_H	E_{I1}	E_{I2}	T_H	T_{I1}	T_{I2}	TPC_DI_RPS
pico	19.288,3	293,8	296,9	404,6	18,5	18,6	86
nano	55.122,2	781,4	775,7	289,9	37,5	37,4	104
micro	105.896,3	1.350,3	1.364,4	228,1	56,5	56,5	113

Tabelle 6.2.: TPC-DI Metriken über alle Datensets für die Gobblin-Spark-Kombination mit Clustergröße 1

Datenset	E_H	E_{I1}	E_{I2}	T_H	T_{I1}	T_{I2}	TPC_DI_RPS
pico	7.121,9	155,1	151,8	1.095,9	18,5	18,6	142
nano	32.175,8	344,3	348,4	496,7	37,5	37,4	136
micro	215.550,9	805,8	814,4	112,1	56,5	56,5	79

Vergleicht man die einzelnen Werte der beiden Tabellen miteinander, dann fällt auf, dass die Gobblin-Spark-Messung im historischen Import in den Datensets *pico* und *nano* höhere Durchsätze (T_H) erreicht als die Talend-Messung. Obwohl die Gobblin-Spark-Imports auch kürzere Laufzeiten bei den inkrementellen Imports (E_{I1} und E_{I2}) aufweisen, erreichen sie denselben Durchsatzwert wie auch die Talend-Importjobs. Dies liegt an der Formel zur Berechnung der TPC-DI-Metriken aus Abschnitt 3.7. Zur Berechnung des Durchsatzes für inkrementelle Imports wird eine Laufzeitdauer von mindestens 1800 Sekunden verwendet. Da sowohl der inkrementelle Import in der Talend- als auch in der Gobblin-Spark-Implementation eine Laufzeitdauer von weniger als 1800 Sekunden erreichten, wird zur Berechnung des Durchsatzes dieselbe Laufzeitdauer verwendet.

Aus den Durchsätzen wird die Metrik TPC_DI_RPS gebildet. Schaut man sich diesen Wert für die Gobblin-Spark-Messung an, sieht man, dass der Wert von 142 im Datenset *pico* auf 136 im größeren Datenset *nano* fällt. Betrachtet man den Wert für die Talend-Messung, ist zu sehen, dass die Metrik TPC_DI_RPS beim größeren Datenset *nano* höher ist als im kleineren Datenset *pico*. Der Wert verhält sich also entgegengesetzt zum Messwert der Gobblin-Spark-Variante.

Die gegenläufige Veränderung der Metrik TPC_DI_RPS lässt sich durch die Entwicklung der Durchsatz-Werte erklären. Die inkrementellen Durchsätze beider Messkandidaten haben bei dem Wechsel vom Datenset *pico* zum Datenset *nano* eine Steigerung von mehr als 100% erfahren. Währenddessen hat sich der Durchsatz des historischen Imports in der Talend-Messung um 28% und in der Gobblin-Spark-Messung um 54% reduziert. Die Reduzierung des Wertes TPC_DI_RPS der Gobblin-Spark-Messung zwischen den Datensets lässt sich daher mit einem verhältnismäßig großen Verlust des historischen Durchsatzes erklären.

Es fällt auch auf, dass sich die Metrik TPC_DI_RPS für die Gobblin-Spark-Implementation mit größer werdenden Datensets reduziert. Dies ist auf die stark gestiegene Laufzeit im historischen Import zurückzuführen.

Bleuel (2016) hat mit dem Tool *Pentaho Kettle* die TPC-DI Spezifikation implementiert und eine Messung durchgeführt. Er erreichte einen Wert von 13 Zeilen pro Sekunde in der Metrik TPC_DI_RPS und 61,54 € pro Zeile pro Sekunde für die Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$. Er verwendete für die Messung ein Datenset, das kleiner ist als das hier verwendete Datenset *pico*. Möchte man das Ergebnis dieser Arbeit mit der von Bleuel vergleichen, dann sollten die Metrikwerte des Datensets *pico* verwendet werden. Des Weiteren hat Bleuel andere Hardware zur Messung verwendet als diese Arbeit. Man kann daher nicht ohne Weiteres die Performance-Metriken TPC_DI_RPS dieser Arbeit mit der von Bleuel vergleichen. Mit der Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ wird die errechnete Performance auf die Hardwarekosten umgelegt. Damit können auch Messungen, die mit unterschiedlicher Hardware durchgeführt wurden, miteinander verglichen werden.

Zur Berechnung der Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ für die Implementationen dieser Arbeit wird der Preis der verwendeten Computer benötigt. Der Hersteller Fujitsu stellt keine Preisliste oder eine *Unverbindliche Preisempfehlung* für das verwendete Modell bereit. Daher wird der Kaufpreis verwendet, der sich allerdings aus einem Rahmenvertrag ergibt. Jeder Computer hat einen Brutto-Kaufpreis von 1613,64 €. In den Single Node-Messungen werden zwei Computer verwendet, womit das SUT Kosten von 3227,28 € erzeugt. Nimmt man die Metrik TPC_DI_RPS der Messungen des Datensets *pico*, dann ergibt sich für die Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ in der Talend-Implementation der Wert **37,53 € pro Zeile pro Sekunde** und für die Gobblin-Spark-Implementation der Wert **22,73 € pro Zeile pro Sekunde**. Die Metrik gibt an, wie teuer es mit der jeweiligen Hardware ist, eine Zeile pro Sekunde zu verarbeiten. Es gilt daher, je niedriger der Wert für die Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ ist, desto besser steht das SUT im Vergleich zu anderen Messungen.

6.2. Analyse der Laufzeiten

Die Messungen wurden mit verschiedenen Parametern durchgeführt und entsprechend viele Messergebnisse wurden erfasst. Im Folgenden werden die gemessenen Laufzeiten der unterschiedlichen Varianten analysiert. Die Analyse wird unterteilt in Non-Streaming Laufzeiten mit der Clustergröße 1, Non-Streaming Laufzeiten mit mehreren Nodes und Streaming Laufzeiten.

6.2.1. Non-Streaming Single Node Laufzeiten

In Abschnitt 5.1 wurden die Laufzeiten unter Berücksichtigung der vielen Parametervariationen, die ein Messjob haben kann, visualisiert. Die Laufzeitergebnisse werden im Folgenden nachein-

ander analysiert. Es wird begonnen mit den Ergebnissen aus den Non-Streaming-Messungen mit der Clustergröße 1 - den Single Node-Messungen.

Die Laufzeiten jedes einzelnen Jobs sind in den Abbildungen 5.1, 5.2 und 5.3 für die Datensets *pico*, *nano* und *micro* dargestellt. Es werden zunächst die historischen Imports betrachtet. Dort fällt auf, dass es einige Jobs gibt, bei denen in der Talend-Messung eine Ausführungsdauer von wenigen Millisekunden erreicht wurde, während der gleiche Gobblin-Spark-Job mehrere Sekunden benötigte. Dies sind Imports, die statische Dateien importieren, wie zum Beispiel die Jobs *IndustryImport*, *StatusTypeImport* und *TaxRateImport*. Diese Auffälligkeit zieht sich durch alle Datensets, was daran liegt, dass die statischen Dateien in allen Datensetgrößen denselben Inhalt und dieselbe Größe haben.

Es fällt außerdem auf, dass die Talend-Implementation die Jobs *DateImport* und *HRImport* deutlich schneller durchführt als die Gobblin-Spark-Implementation. Vergleicht man die Messungen der Datensets *pico* und *nano* miteinander, dann ist zu sehen, dass die Laufzeitverhältnisse zwischen den Implementationen gleich sind. Die Implementation, die in den *pico*-Messungen eine kürzere Laufzeit aufweist, ist auch in den *nano*-Messungen schneller. Lediglich die Jobs *StatusTypeImport* und *CustomerMgmtImport* fallen aus diesem Muster. Bei dem Job *StatusTypeImport* ist dies auf eine Messungenauigkeit zurückzuführen, die bei der sehr kurzen Laufzeit stark auffällt. Der *CustomerMgmtImport* hingegen kann bei größer werdender Datenmenge mit der Gobblin-Spark-Implementation eine kürzere Laufzeit erreichen als mit der Talend-Variante.

Diese Verhältnisse werden durch die Messungen des Datensets *micro* bestätigt. Mit der größeren Datenmenge kann der Job *HoldingHistoryImport* in der Gobblin-Spark-Implementation ebenfalls kürzere Laufzeiten als in der Talend-Variante aufweisen. Allerdings ist mit dieser Datenmenge der *DailyMarketImport* in der Talend-Implementation das erste Mal schneller als in der Gobblin-Spark-Kombination.

Die Analyse zeigt, dass nicht pauschal gesagt werden kann, dass eine der Implementationen ab einer bestimmten Dateigröße immer schneller ist als die Vergleichsimplementation. Dies liegt daran, dass die Jobs unterschiedliche Charakteristika haben, wie in Abschnitt 3.5 beschrieben. Es können allerdings anhand der Messungen für einige Jobs Größengrenzwerte genannt werden, bis zu der eine Implementation schneller ist als die andere. Anhand der durchgeführten Messungen können folgende Grenzwerte bestimmt werden:

CustomerMgmtImport Ab einer Datengröße von 30MB zeigt die Gobblin-Spark-Implementation die kürzeste Laufzeit

DailyMarketImport Ab einer Datengröße von 455MB zeigt die Talend-Implementation die kürzeste Laufzeit

HoldingHistoryImport Ab einer Datengröße von 40MB zeigt die Gobblin-Spark-Implementation die kürzeste Laufzeit

WatchHistoryImport Ab einer Datengröße von 202MB zeigt die Gobblin-Spark-Implementation die kürzeste Laufzeit

Anschließend werden die inkrementellen Imports in den Abbildungen 5.1, 5.2 und 5.3 betrachtet. In diesen Imports werden, im Verhältnis zum historischen Import, kleine Dateien verarbeitet, weshalb hier die Talend-Implementation in den meisten Jobs geringere Laufzeiten erreicht als die Gobblin-Spark-Implementation. Die Jobs *DailyMarketImport* und *ProspectImport* werden auch in den inkrementellen Imports durch die Gobblin-Spark-Kombination schneller verarbeitet. Es fällt außerdem auf, dass die Jobs *CustomerImport* und *TradeImport* von Gobblin-Spark deutlich mehr Zeit benötigen als die von Talend. Dies liegt vermutlich am Vorgehensmodel der Gobblin-Spark-Implementation, das in Abschnitt 4.2 beschrieben wird.

Die Implementation lädt zu Beginn eines Jobs alle benötigten Datenbanktabellen zunächst in den Hauptspeicher. Beim inkrementellen *TradeImport* wird die komplette Tabelle *DimTrade* geladen, die nach dem historischen Import des Datensets *pico* 650.412 Einträge enthält. Bei größeren Datensets dauert dieser Vorgang entsprechend länger, da die Tabelle mehr Einträge enthält. Dies ist eine Vermutung, die mit dem aktuellen Messaufbau nicht belegt werden kann. Dazu müssen weitere Messpunkte festgelegt und anschließend analysiert werden. Durch die Vermutung ließen sich auch die verhältnismäßig langen Laufzeiten anderer inkrementeller Gobblin-Spark-Jobs erklären, in denen ebenfalls die benötigten Datenbanktabellen zu Beginn des Jobs in den Hauptspeicher geladen werden.

Summiert man alle Joblaufzeiten, die in einem Batch von den jeweiligen Messkandidaten durchgeführt werden, ergibt dies die Summen, die in Abbildung 5.4 dargestellt sind. Mit den Datensetgrößen *pico* und *nano* konnte die Gobblin-Spark-Implementation den historischen Import 1,6- bis 2,6-mal so schnell durchführen, wie die Talend-Implementation. Der historische Import des Datensets *micro* konnte hingegen 2-mal so schnell beendet werden von der Talend-Implementation im Vergleich zur Gobblin-Spark-Kombination. In den inkrementellen Imports ist die Gobblin-Spark-Implementation je nach Datensetgröße 1,5- bis 1,9-mal so schnell wie die Talend-Variante.

In der Heatmap-Abbildung 6.1 sind die Anteile der einzelnen Jobs an der Gesamtlaufzeit dargestellt. Es sind vier Diagramme zu sehen. Die zwei Diagramme in der oberen Hälfte zeigen die Anteile der Talend-Imports, während die zwei Diagramme in der unteren Hälfte die Anteile

der Gobblin-Spark-Imports abbilden. In der linken Hälfte sind die historischen Imports und in der rechten Hälfte die inkrementellen Imports mit der BatchID 2 dargestellt. Auf der y-Achse ist jeweils der betroffene Jobname und auf der x-Achse die Datensetgröße abgetragen. In jeder Zelle steht der Anteil der jeweiligen Joblaufzeit an der Gesamtlaufzeit des Batches in Prozent und ist mit einem Farbcode hinterlegt. Der Farbcode *gelb* bedeutet, dass ein Job in einem Datenset einen Anteil an der Gesamtlaufzeit von 0 Prozent hat, während der Farbcode *dunkelblau* je nach Diagramm für über 60 Prozent steht.

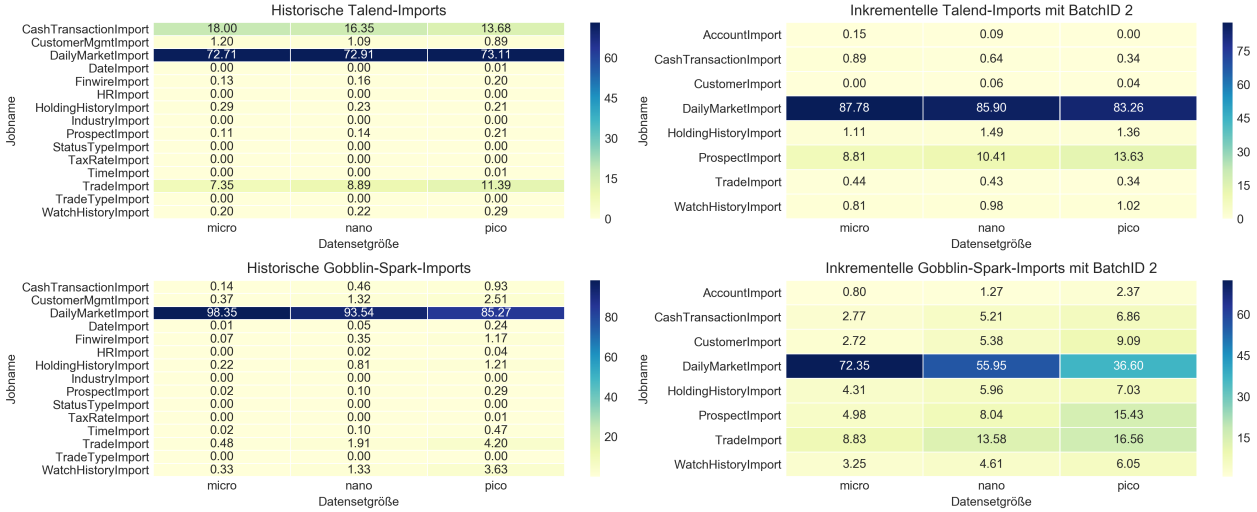

Abbildung 6.1.: Prozentualer Anteil der Importjob-Laufzeiten beider Frameworks über alle Datensetgrößen.

Die Abbildung macht deutlich, dass der Job *DailyMarket* in den historischen Imports - je nach Implementation und Datenset - für 72% bis 98% der Laufzeit verantwortlich ist. Auch in den inkrementellen Imports dominiert der Job *DailyMarketImport* mit mindestens 36% Gesamtlaufzeit-Anteil. Damit ist der *DailyMarketImport*-Job für den Großteil der Laufzeiten verantwortlich, die auch zur Berechnung der TPC-DI-Metriken in den Tabellen 6.1 6.2 verwendet werden.

6.2.2. Non-Streaming Multinode Laufzeiten

Die Messungen der Gobblin-Spark-Implementation wurden mit mehreren Clustergrößen durchgeführt. Diese Messungen unterscheiden sich von den Single Node-Messungen insofern, dass mehrere Nodes an der Durchführung eines Importjobs beteiligt waren. Diese verteilte Verarbei-

tung kann dazu führen, dass der Import im Vergleich zu einem Import durch einen einzelnen Node in kürzerer Zeit abgeschlossen werden kann.

Ob mehr Nodes zu einer kürzen Laufzeit führen, wird im Folgenden erörtert. Dazu werden die Abbildungen 5.5, 5.6 und 5.7 herangezogen. Sie stellen die Importlaufzeiten jedes Gobblin-Spark-Jobs für alle Clustergrößen mit den Datensets *pico*, *nano* und *micro* dar.

Zunächst werden die historischen Imports betrachtet. Wie auch in der Analyse der Single Node-Messungen fallen in den Abbildungen die statischen Imports auf. In allen Datensetgrößen verhalten sich die Importlaufzeiten der Jobs *IndustryImport*, *StatusTypeImport*, *TaxRateImport* und *TradeTypeImport* nicht wie erwartet. Es ist kein eindeutiges Skalierungsverhalten zu erkennen. Im Vergleich zu anderen Jobs weisen die statischen Imports eine sehr kurze Laufzeit von höchstens 1,5 Sekunden auf. Die statischen Jobs importieren in allen Datensetgrößen dieselbe Menge an Daten. Man kann daher annehmen, dass die statischen Imports in allen Datensetgrößen dieselbe Laufzeit aufweisen. Da sie sich allerdings in der Laufzeit unterscheiden, weist dies auf eine Messungenauigkeit hin, die bei diesen Jobs besonders auffällig ist. Durch die Messungenauigkeit kann für die statischen Importsjobs keine Aussage dahingehend getroffen werden, ob die Verarbeitung durch mehrere Nodes eine messbare Veränderung in der Laufzeit bringt.

Andere Jobs wie *TradeImport* und *WatchHistoryImport* im Datenset *pico* zeigen ein anderes Verhalten. Erwartungsgemäß dauern die Jobs mit einem Node am längsten. Mit drei Nodes ist die Laufzeit ebenfalls erwartungsgemäß kürzer als mit einem Node. Die kleinste Laufzeit wurde allerdings mit zwei Nodes erreicht. Die Kommunikation, die bei drei Nodes nötig ist, wiegt den Vorteil der größeren Rechenleistung auf und führt letztendlich zu einer Verzögerung. Betrachtet man die Jobs im Datenset *nano*, findet man bei den Jobs *TimeImport* und *WatchHistoryImport* dasselbe Verhalten. Bei dem Job *HoldingHistory* hingegen ist bei der Datenmengenerhöhung ein Skalierungsverhalten in geringem Umfang eingetreten. Die Laufzeit hat sich von 262,7s (Single Node) zu 173,7s (2 Nodes) zu 162,7 (3 Nodes) verändert. Da der einzige Unterschied zwischen den Messungen die Datensetgröße ist, können die Daten so interpretiert werden, dass das Skalierungsverhalten erst ab einer Importdatengröße von 26MB zu einer kürzeren Laufzeit führt.

Die Jobs *CashTransactionImport*, *DailyMarketImport* und *FinwireImport* zeigen bereits im Datenset *pico*, dass mehr beteiligte Nodes zu einer kürzeren Laufzeit führen.

In der *micro*-Messung verhalten sich die Jobs ähnlich wie in der *nano*-Messung. Lediglich der an den Jobs *TradeImport* und *WatchHistoryImport* beobachtete Skalierungseffekt, womit der Job mit zwei Nodes schneller ist als mit drei, ist aufgehoben.

Betrachtet man die inkrementellen Imports, kann man dort ähnliche Beobachtungen wie in den historischen Imports machen. Bei den Jobs *AccountImport*, *CustomerImport*, *DailyMarketImport*, *HoldingHistoryImport*, *TradeImport* und *WatchHistoryImport* im Datenset *pico* ist zu sehen, dass die Single Node-Messung die geringste Laufzeit aufweist. Sobald die Jobs auf mehreren Nodes verteilt verarbeitet wurden, führte dies zu einer verlängerten Laufzeit.

Mit dem nächstgrößeren Datenset *nano* tritt bei den Jobs *CashTransactionImport*, *DailyMarketImport* und *ProspectImport* eine geringe Reduzierung der Laufzeit mit mehreren Nodes ein. Wird die Datenmenge weiter auf *micro* erhöht, dann kann man auch für den Job *AccountImport* eine Laufzeitreduzierung mit zwei Nodes erkennen.

In den Abbildungen 6.2, 6.3 und 6.4 sind die Laufzeitveränderungen der Multi Node-Messungen relativ zur Single Node-Messung in den einzelnen Datensets dargestellt. Jeweils in der linken Hälfte sind die historischen Jobs und in der rechten Hälfte die inkrementellen Imports dargestellt. Auf y-Achse ist jeweils der Jobname abgetragen, während auf der x-Achse die Clustergröße abgebildet ist. Die Zellen beinhalten jeweils einen Prozentwert, der beschreibt, um wieviel Prozent die Importjob-Laufzeit mit der jeweiligen Clustergröße geringer oder größer wurde. Der *CashTransactionImport* beispielsweise wurde in der *pico*-Messung mit der Clustergröße 2 um 40,49% schneller erledigt als in der Single Node-Messung. Mit der Clustergröße 3 wurde der Job sogar um 46,83% schneller erledigt als in der Single Node-Messung. In den Diagrammen sind die statischen Imports aufgrund der starken Schwankungen in den Messwerten nicht dargestellt.

Wenn eine Zelle den Wert 0 enthält, dann hat sich die Laufzeit im Vergleich zur jeweiligen Single Node-Messung nicht verändert. Wenn die Zelle einen negativen Wert hat, dann bedeutet das, dass die Messung um den jeweiligen Prozentsatz langsamer war als die Single Node-Messung.

Die Visualisierung verdeutlicht, dass das erwartete Skalierungsverhalten abhängig vom Importjob erst ab einer gewissen Datenmenge einsetzt. Solange die Datenmenge unter diesem Schwellenwert ist, führt die Erhöhung der Clustergröße nicht zu einer Reduzierung der Laufzeit. Wie die Analyse zeigt, profitieren viele Jobs nur in einem sehr geringem Umfang von mehreren verarbeitenden Nodes. Am deutlichsten ist die Reduzierung bei dem Job *DailyMarketImport* zu sehen, der der Job mit dem meisten Zeitaufwand ist.

Auf nahezu keinen Job trifft zu, dass eine lineare Skalierung eingetreten ist. Eine ideale lineare Skalierung sähe so aus, dass bei einer Clustergröße von 2 die Laufzeit um 50% reduziert ist und bei einer Clustergröße von 3 um circa 66,67%. Diese idealen Laufzeitreduzierungen sind bei dem Job *DailyMarketImport* in den Datensets *pico* und *nano* zu finden. Bei der Erhöhung zum Datenset *micro* nimmt der Skalierungseffekt hingegen stark ab.

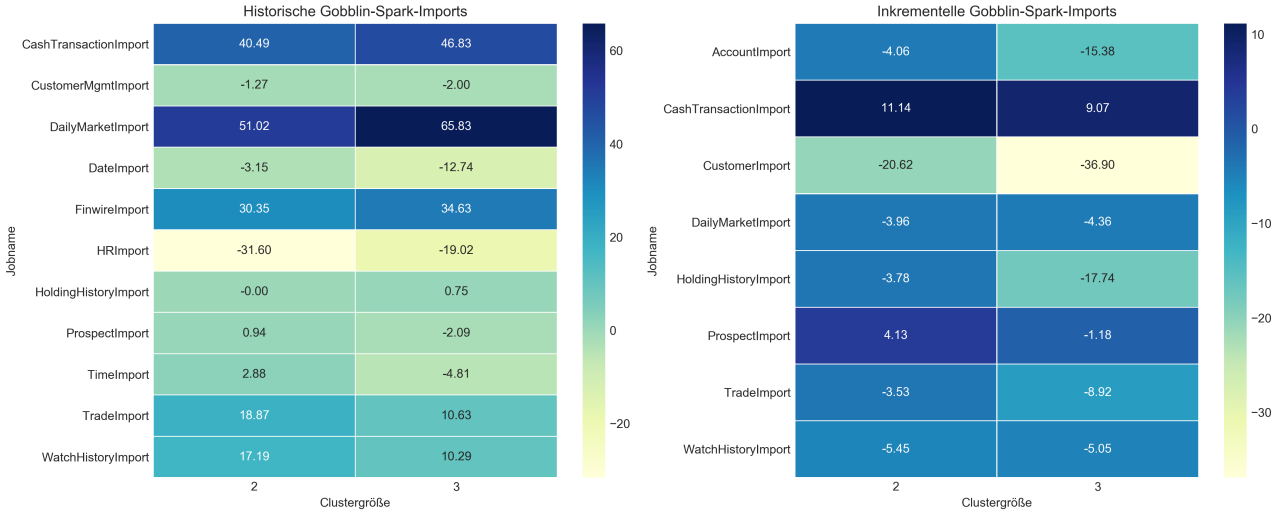

Abbildung 6.2.: Laufzeitveränderungen der Gobblin-Spark Multi-Node Messungen verglichen mit der Single Node-Messung des Datensets *pico*

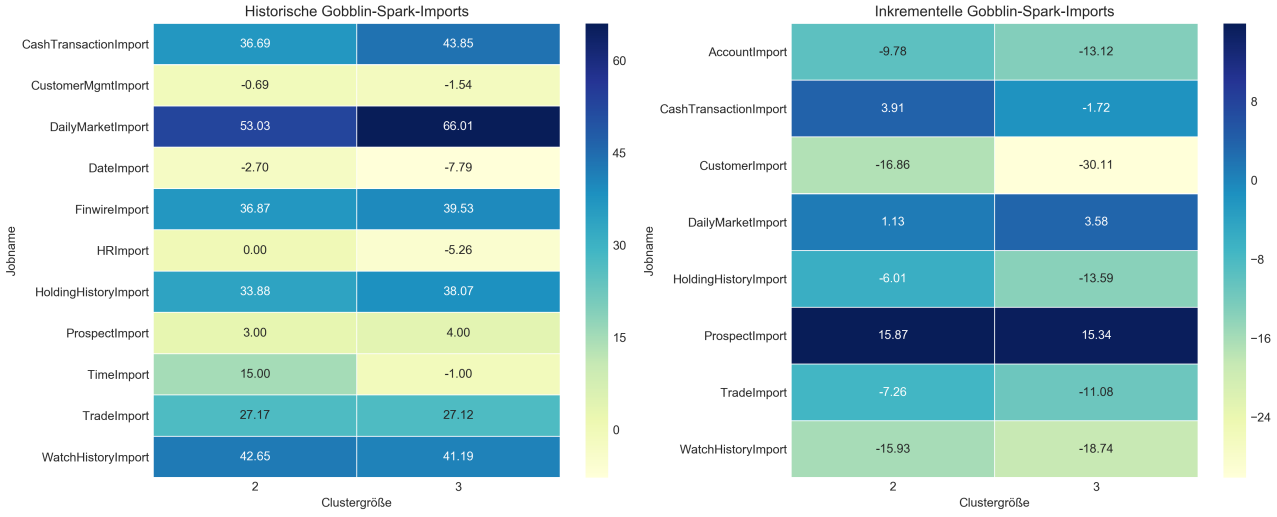

Abbildung 6.3.: Laufzeitveränderungen der Gobblin-Spark Multi-Node Messungen verglichen mit der Single Node-Messung des Datensets *nano*

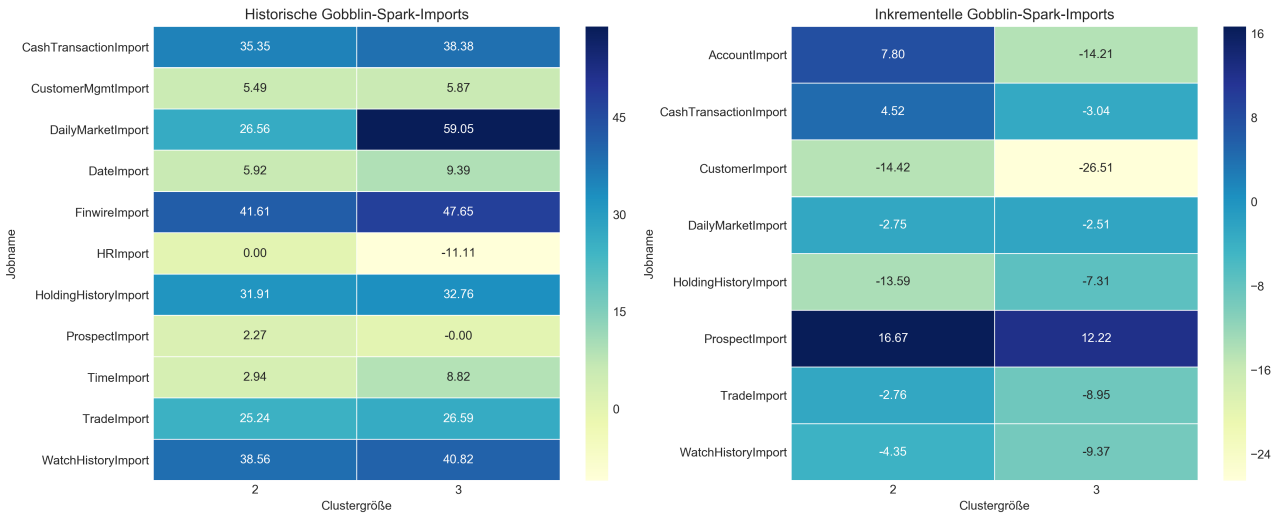

Abbildung 6.4.: Laufzeitveränderungen der Gobblin-Spark Multi-Node Messungen verglichen mit der Single Node-Messung des Datensets *micro*

6.2.3. Streaming Laufzeiten

Abschließend werden die Laufzeiten der Streaming-Modifikation analysiert. Die Ergebnisse des Streaming-Importjobs wurden in der Abbildung 5.8 dargestellt. Die Abbildung zeigt, was auch vorher schon in den Non-Streaming-Jobs wiederholt zu sehen war: mit einer kleinen Datenmenge (*pico*) erzielt die Talend-Implementation eine geringere Laufzeit als der Vergleichskandidat. Ab der Datenmenge *nano*, erreicht die Gobblin-Spark-Implementation eine kürzere Laufzeit als Talend. In der rechten Hälfte des Diagramms sind die Gobblin-Spark-Messungen mit mehreren Nodes dargestellt. Dort kann man erkennen, dass dieser Job mit mehreren Nodes eine Laufzeitreduzierung erfährt.

Neben dem einfachen Vergleich der Framework-Implementationen kann die Streaming-Variante außerdem mit den Non-Streaming-Kandidaten verglichen werden. Die Laufzeiten dieser Vergleichskandidaten sind in Abbildung 6.6 dargestellt. Dort sind die Laufzeiten der Jobs auf der y-Achse abgetragen und die Datensetgrößen auf der x-Achse. Jede Implementation ist durch eine Balkenfarbe repräsentiert: Die Non-Streaming Talend-Implementation ist in blau dargestellt, die Streaming Talend-Implementation in grün, die Non-Streaming Gobblin-Spark-Implementation in rot und die Streaming Gobblin-Spark-Implementation in purpur.

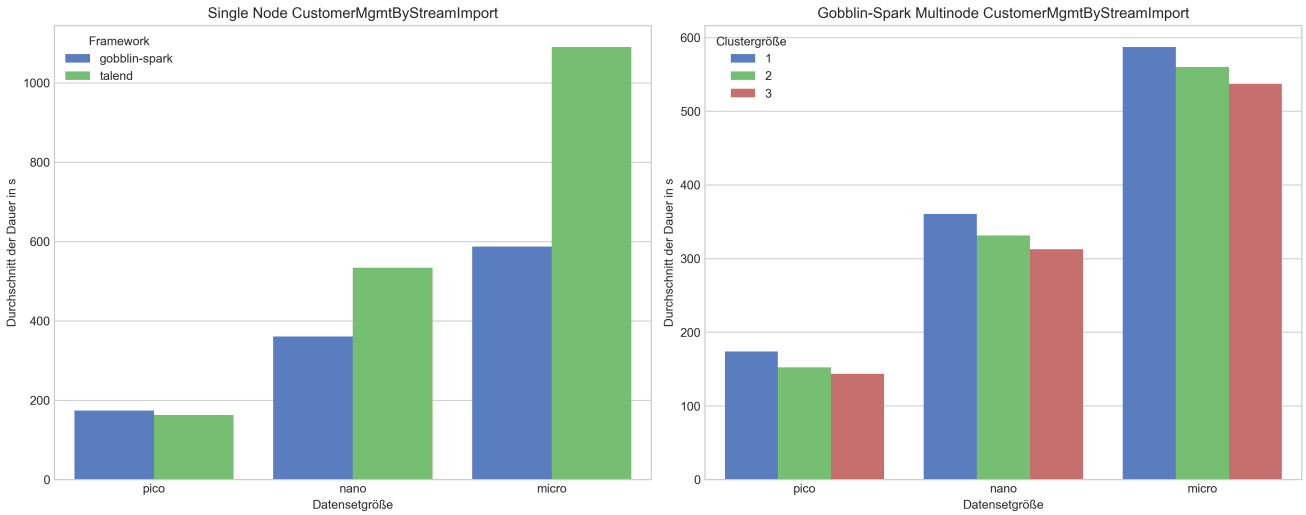

Abbildung 6.5.: Laufzeitvergleich des Streaming-Jobs *CustomerMgmtImport* mit beiden Implementationen in allen Datensets, gemäß der Tabellem A.14 und A.15

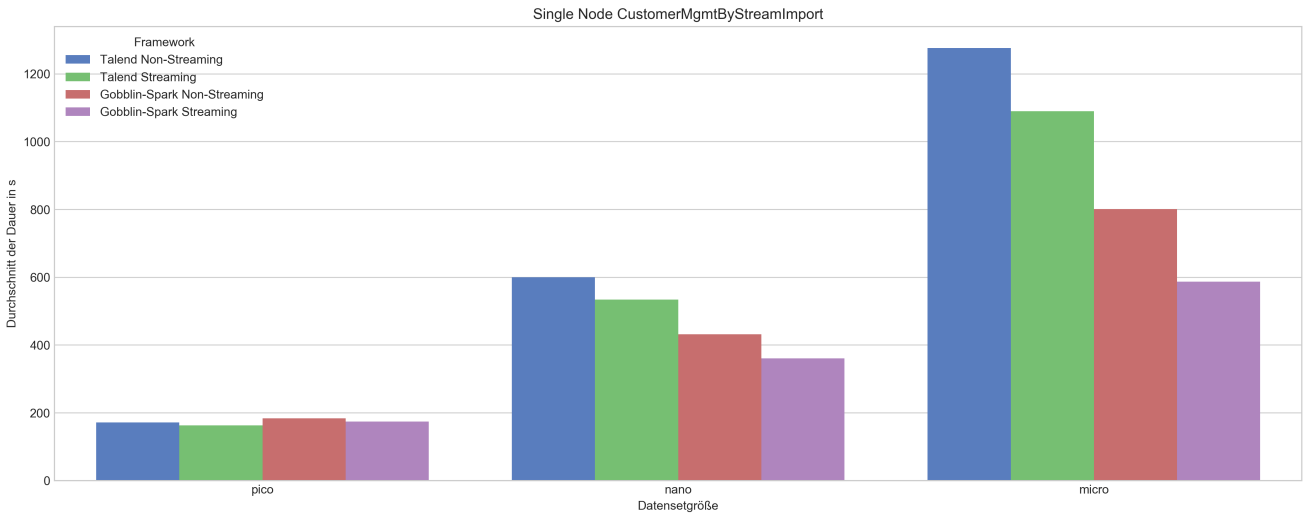

Abbildung 6.6.: Laufzeitvergleich des Streaming- und Non-Streaming-Jobs *CustomerMgmtImport* in allen Datensets

Die Visualisierung zeigt, dass die jeweiligen Streaming-Varianten kürzere Laufzeiten aufweisen als die zugehörige Non-Streaming-Implementation. Dies trifft auf alle Datensetgrößen zu.

6.3. Analyse der Hardware-Nutzung

Für jeden Job wurden während der Messung Hardware-Nutzungsmetriken erfasst. Es werden allerdings nicht alle Metriken für jeden Job analysiert. Wie auch in Abschnitt 5.2.1 beschränkt sich diese Analyse auf die CPU- und RAM-Nutzung der Jobs *CustomerMgmtImport*, *DailyMarketImport* und *HoldingHistoryImport*.

Im Folgenden werden die einzelnen Jobs zunächst in der Non-Streaming Single Node-Implementation und anschließend in der Non-Streaming Multi Node-Variante analysiert. Abschließend wird der modifizierte Streaming-Job mit seinem Non-Streaming-Äquivalent verglichen.

6.3.1. Non-Streaming Single Node Hardware-Nutzung

Die Messergebnisse, die in diesem Abschnitt analysiert werden, sind in den Abbildungen 5.9, 5.10, A.1, A.2, A.9, A.10, A.6, A.5, A.13 und A.14 visualisiert. Die ersten sechs Abbildungen beinhalten die Daten der historischen Imports und die letzten vier stellen die Daten der inkrementellen Imports dar.

Zunächst werden die historischen Importjobs betrachtet. Wie in Abschnitt 5.1.1 erläutert, steht die blaue Kurven in den Abbildungen jeweils für die Nutzung des Node *01*, auf dem im Single Node-Modus die Import-Implementationen ausgeführt werden. Die orange Kurve repräsentiert die Nutzung des Node *03* auf dem ausschließlich die Datensenke, der MySQL-Server, läuft.

Vergleicht man die Talend- und Gobblin-Spark-Implementation des Jobs *CustomerMgmtImport* (siehe Abbildung 5.9 und 5.10), kann man in den ersten 70 Sekunden eine ähnliche CPU-Auslastung beobachten. Da der Job eine XML-Datei importiert, ist es naheliegend, dass diese ähnliche Auslastung der ersten 70 Sekunden auf das Parsen der Importdatei zurückzuführen ist. Nach dem Parsen gibt es keine Gemeinsamkeiten mehr in den Kurvenverläufen.

Die Talend-Implementation hat eine, bis auf einige Spitzen, durchgängig gleichmäßige CPU- und RAM-Nutzung sowohl auf Node *01* als auch auf Node *03*. Die Ausführung von Talend belegt die CPU des Node *01* mit einer Nutzung, die knapp über 0% liegt, während der MySQL-Server seine CPU nahezu konstant mit einer Nutzung von 20% belegt. Die RAM-Nutzung beider Nodes ist ebenfalls nahezu konstant bei circa 2GB. Betrachtet man das Hardware-Diagramm

der Gobblin-Spark-Implementation nach dem XML-Parsen, findet man keine Gemeinsamkeiten mit dem Talend-Diagramm. Die Nutzung des Gobblin-Spark-Imports ist mit vielen Spitzen auf dem Node *01* durchzogen. Zu vielen dieser Spitzen gibt es zeitgleich eine intensive CPU-Nutzung auf dem MySQL-Server. Dies deutet darauf hin, dass zu diesen Zeitpunkten vom Importjob Daten aus vom Datenbankserver gelesen oder hinein geschrieben werden. Während der Gobblin-Spark-Imports nutzt der MySQL-Server konstant circa 2GB. Zu den Zeitpunkten, in denen die Spark-Verarbeitung stattfindet (rechts von der senkrechten purpurfarbenen Linie) beginnt die RAM-Nutzung bei 10GB und steigt im Verarbeitungsverlauf auf circa 14,5GB.

Anhand der Diagramme der beiden Implementationen kann man die unterschiedlichen Vorgehensweisen wiedererkennen, auf die in Abschnitt 4.2 eingegangen wird. Die Talend-Implementation verfolgt das Verarbeitungsparadigma *one-at-a-time*, weshalb während der Verarbeitung einer einzelnen Zeile verschiedene *SELECT*-, *INSERT*- und *UPDATE*-Anweisungen an die Datenbank gegeben werden. Damit wird eine gleichmäßige Last auf dem Node *03* erzeugt. Hingegen arbeitet der Spark-Anteil der Gobblin-Spark-Implementation mit einem Map-Reduce-ähnlichen Verarbeitungsparadigma. Daher beginnt der nächste Verarbeitungsschritt für alle Importdaten erst dann, sobald der vorherige Verarbeitungsschritt für alle Daten beendet wurde. Jede Spitze in der MySQL-CPU-Nutzung zeigt daher die Zeitpunkte, an denen ein Batch per *INSERT*- und *UPDATE*-Anweisungen an die Datensenke ausgeliefert wird. Zwischen diesen Spitzen verarbeitet der Gobblin-Spark-Job die Daten auf Node *01*, was die CPU-Nutzung zu den Zeitpunkten erklärt.

Die Talend-Implementation des Jobs *DailyMarketImport* (siehe Abbildung A.1) zeigt in der CPU-Nutzung einen ähnlichen Kurvenverlauf wie er zuvor bei dem Job *CustomerMgmtImport* beobachtet werden konnte. Zu Beginn des Jobs gibt es hohe schwankende CPU-Nutzung auf den Nodes *01* und *03*. In dieser Zeitspanne werden die Importdateien und Datenbanktabellen, die zum Verarbeiten der Daten notwendig sind, in den Hauptspeicher des Nodes *01* geladen. Circa ab dem Zeitpunkt von 4000s ist wieder die konstante CPU-Auslastung des Nodes *01* bei 5% und des Nodes *03* bei 20% zu sehen. Diese zieht sich bis zum Ende des Imports durch. Die RAM-Nutzung des Jobs *DailyMarketImport* ist höher als die des *CustomerMgmtImport*-Jobs, was darauf zurückzuführen ist, dass eine größere Datenmenge importiert wird und größere Tabellen in den Hauptspeicher geladen werden. Die RAM-Nutzung beginnt bei 5GB und erhöht sich stufenweise auf circa 8GB. Bis zum Ende des Imports fällt die Arbeitsspeicherauslastung auf 7,5GB.

Die Gobblin-Spark-Variante des *DailyMarketImport*-Jobs verhält sich ebenfalls anders als der *CustomerMgmtImport*-Job (siehe Abbildung A.2). Im Verhältnis zur Spark-Verarbeitung ist der Gobblin-Anteil (links von der purpurfarbenen senkrechten Linie) sehr kurz. Zur Zeit der Spark-

Verarbeitung liegt die CPU-Auslastung des Node *01* konstant bei circa 100%. Die CPU-Nutzung des MySQL-Servers hingegen liegt lediglich bei knapp über 0%. Es gibt auf dem Node *03* einige Spitzen, die allerdings nicht mit dem Importjob zusammenhängen. Zu den Zeitpunkten können interne MySQL-Aufgaben, wie zum Beispiel Aufräum- oder Indizierungsaufgaben stattgefunden haben. Zum Zeitpunkt 22500s sinkt die CPU-Nutzung des Node *01* auf 75% und circa zum Zeitpunkt 27500s steigt die CPU-Last des MySQL-Servers deutlich, was dafür spricht, dass der Importjob abschließend Daten in die Datensenke schreibt. Die Arbeitsspeichernutzung des Jobs beginnt mit 12,5GB und sinkt im Verlauf der Zeit, sodass der Job mit circa 7,5GB endet. Der Verlauf des belegten Arbeitsspeichers lässt sich damit erklären, dass zu Beginn die nötigen Importdateien und Datenbanktabellen geladen werden. Im Verlauf werden diese Daten im Transformationsprozess genutzt. Sobald die Daten nicht mehr benötigt werden, wird der zugehörige Arbeitsspeicher freigegeben.

Im Rahmen des Jobs *HoldingHistory* wird eine Datei und eine Datenbanktabelle zusammengeführt und anschließend werden einige Felder umbenannt. Der Job hat eine geringe Komplexität, es müssen allerdings verhältnismäßig viele Daten verarbeitet werden, was zu einer langen Laufzeit führt. Die Talend- und die Gobblin-Spark-Implementation weisen in diesem Job sehr ähnliche CPU-Nutzungskurven auf (siehe Abbildungen A.9 und A.10). Zu Beginn beider Implementationen gibt es Spitzen auf dem Node *01*, was sich auf das Zusammenführen der Importdaten zurückführen lässt. Anschließend steigt die CPU-Last des MySQL-Servers in beiden Messungen auf 40-50%, während die CPU-Nutzung auf dem ausführenden Node *01* zwischen 0% und 15% schwankt. Zu diesen Zeitpunkten wird die große Datenmenge in die Datensenke geschrieben. In der RAM-Nutzung der Implementationen gibt es Unterschiede. Die Talend-Variante hat eine nahezu konstante Nutzung von circa 7,5GB. Der Gobblin-Spark-Import nutzt im Gobblin-Anteil konstant circa 8GB und im Spark-Teil circa 16GB.

Die Hardware-Nutzung der inkrementellen Jobs ähnelt der Nutzung der historischen Jobs. Sie unterscheiden sich lediglich in der Laufzeit und damit der Anzahl der Messpunkte. Es wird daher auf eine detaillierte Analyse der Hardware-Nutzung der inkrementellen Single Node Imports verzichtet.

Die Analyse der Hardware-Nutzung zeigt, dass die Gobblin-Spark-Jobs dazu neigen die CPU des ausführenden Node *01* vollständig auszulasten, während die Talend-Jobs kaum die CPU des Nodes *01* beanspruchen und eher CPU-Last auf dem MySQL-Server erzeugen. Keine der Implementationen hat den zur Verfügung stehenden Arbeitsspeicher (oder andere gemessene Hardware-Ressourcen) vollständig genutzt, weshalb der RAM und andere Hardware-Ressourcen als beschränkende Ressourcen im Sinne eines Flaschenhals-Effekts ausgeschlossen werden können. Es wird daher die Kommunikation mit dem MySQL-Server als beschränkender

Faktor identifiziert für die Jobs, die nicht vollständig die CPU des Node *01* nutzen, was vor allem die Talend-Jobs betrifft.

Angenommen die Talend-Implementation würde weniger mit dem MySQL-Server kommunizieren oder der MySQL-Server könnte die SQL-Anweisungen der Talend-Implementation schneller abarbeiten, dann würde die Laufzeit der Talend-Jobs vermutlich drastisch reduziert werden.

6.3.2. Non-Streaming Multi Node Hardware-Nutzung

Nachdem die Hardware-Nutzung der Single Node-Messung in der Gobblin-Spark-Implementation analysiert wurde, folgt der Vergleich der Hardware-Nutzung mit mehreren verarbeitenden Nodes. Die erfassten Nutzungswerte für mehrere Nodes sind in den Abbildungen 5.11, 5.12, A.3, A.4, A.11 und A.12 dargestellt. Zusätzlich zu den Kurven aus der Single Node-Messung gibt es die grüne Kurve, die die Messwerte des Nodes *05* darstellt und die rote Kurve, die die Messwerte des Nodes *06* repräsentiert.

Zunächst werden die historischen Importjobs betrachtet. Vergleicht man die Messungen des Jobs *CustomerMgmtImport* mit den unterschiedlichen Clustergrößen, fällt auf, dass sie sich sehr ähneln. Wie auch zuvor in der Single Node-Messung gibt es Spitzen des MySQL-Servers. In der Messung mit der Clustergröße 2 (Abbildung 5.11) ist zu sehen, dass es Zeitpunkte gibt - beispielsweise 90s bis 150s und 250s bis 300s - an denen beide Nodes Daten verarbeiten. Die meiste Zeit - beispielsweise 150s bis 160s, 180s bis 220s und 355s bis 380s - verarbeitet der Node *01* den Importjob alleine. Es fällt ebenfalls auf, dass häufig, wenn Anweisungen an den MySQL-Server geschickt werden - beispielsweise 160s bis 165s, 220s bis 230s, 255s bis 260s, 265s bis 270s und viele weitere - wie erwartet die CPU-Last auf dem MySQL-Server steigt. Zeitgleich sinkt die CPU-Last auf dem Node *01*, der vorher noch eine CPU-Nutzung von circa 25% hatte, auf knapp über 0% und die CPU-Nutzung des Nodes *05* steigt auf über 60%. Dieser Status dauert einige wenige Sekunden bis die CPU-Last des Nodes *01* wieder auf circa 25% steigt und die des Nodes *05* auf knapp über 0% sinkt.

Dies ist ein unerwartetes Verhalten, das nicht optimal ist. Denn das bedeutet, dass auf dem Node *01* Daten verarbeitet und dann an den Node *05* übertragen werden, damit sie in die Datenbank geschrieben wird. Der Transformationsprozess sendet in dem Fall Daten über das Netzwerk, was vermieden werden kann. Betrachtet man die Hardware-Nutzung mit drei Nodes (Abbildung 5.12), ist zu beobachten, dass die zusätzlichen Nodes sich nicht nennenswert am Transformationsprozess beteiligen. Die CPU-Last der Nodes *05* und *06* steigt kaum über 0%, während der Node *01*, wie auch in der Single Node-Messung, eine Grundlast von circa 30% und einigen Spitzen aufweist. Mit einer Clustergröße von 3 findet keine gleichmäßige

Lastverteilung auf den beteiligten Nodes statt. Die Abbildungen der Hardware-Nutzung zeigen, dass die Skalierung des Jobs *CustomerMgmtImport* miserabel ist, denn sie findet kaum bis gar nicht statt.

Die Lastverteilung des Jobs *DailyMarketImport* verhält sich so, wie erwartet (Abbildungen A.3 und A.4). In allen Clustergrößen liegt die CPU-Last aller beteiligten Nodes bei circa 100%. Es ist ebenfalls zu beobachten, dass ab einem bestimmten Zeitpunkt - in der Clustergröße 1 ist es bei circa 22500s, in der Clustergröße 2 bei circa 9500s, in der Clustergröße 3 bei circa 4500s - die CPU-Last der beteiligten Nodes stufenweise absinkt. Bei der Auswertung der RAM-Nutzung fällt auf, dass der genutzte Arbeitsspeicher des Nodes *01* um circa 2-3GB höher ist als bei den Nodes *05* und *06*. Dies ist darauf zurückzuführen, dass auf dem Node *01* zusätzlich zum Spark-Slave der Spark-Master und der Spark-Driver laufen, die ebenfalls Arbeitsspeicher in Anspruch nehmen, wie in Abschnitt 4.5.4 erläutert.

Vergleicht man die CPU-Last des Jobs *HoldingHistoryImport* in den unterschiedlichen Clustergrößen, kann man erkennen, dass sie sich sehr ähneln (Abbildungen A.11 und A.12). Wie auch zuvor in der Single Node-Messung beschrieben, ist die CPU-Last des MySQL-Servers schwankend zwischen 40% und 70%. Betrachtet man dies genauer, dann sieht man, dass der MySQL-Server bei einer Clustergröße von 1 die CPU zwischen 40% und 60% nutzt, bei einer Clustergröße von 2 wird die CPU zwischen 45% und 65% und bei einer Clustergröße von 3 zwischen 45% und 70% genutzt. Abhängig von den beteiligten Nodes erhöht sich die CPU-Nutzung des Nodes *03*, was für ein Skalierungsverhalten auf dem MySQL-Server spricht. Dies wird durch die CPU-Last der Nodes *01*, *05* und *06* bestätigt, die, wie auch in der Single Node-Messung, auf allen beteiligten Nodes bei circa 5% bis 10% liegt. Es ist in diesem Job also ein Skalierungsverhalten in der CPU-Nutzung zu erkennen.

Bei dem Job *HoldingHistoryImport* werden zwar mehrere Nodes an der Verarbeitung beteiligt, die CPU der Nodes wird allerdings kaum beansprucht. Betrachtet man mit diesem Wissen nochmal die Laufzeiten der Multi Node-Messungen des Datensets *micro* in Abbildung 5.7, kann ein Muster entdeckt werden. Dort ist zu sehen, dass mit mehreren Nodes im Job *HoldingHistoryImport* kaum eine Laufzeitreduzierung eintritt. Dasselbe Muster ist bei den Jobs *FinwireImport*, *TradeImport* und *WatchHistoryImport* zu sehen. Möglicherweise beanspruchen die Jobs ebenfalls die CPU des verarbeitenden Node kaum, sodass sie nicht von einer verteilten Verarbeitung profitieren und die erwartete Laufzeitreduzierung nicht eintritt.

Um das zu überprüfen, wurden die Diagramme zur Hardware-Nutzung des Jobs *WatchHistoryImport* in den Abbildungen A.17, A.18 und A.19 in den Anhang gelegt. Vergleicht man diese Diagramme miteinander, dann ist zu sehen, dass auch dieser Job die CPU der ausführenden Nodes nur in einer Spitze komplett auslastet. Diese Spitze ist in der Single Node-Messung bei

325s. Links davon lädt der Job Daten aus der Datenbank, die zusammengeführt werden sollen. Daher ist zu der Zeit die CPU-Last auf dem MySQL-Server erhöht. Rechts von der Spitze werden die Ergebnisdaten in die Datensenke geschrieben, weshalb auch zu der Zeit die CPU-Last auf dem Datenbankserver erhöht ist. Nur zu dem Zeitpunkt, an dem die Spitze auftritt, passiert die tatsächliche Datentransformation. In diesem Zeitfenster von circa 20 Sekunden kann die Rechenleistung von bis zu drei Nodes verwendet werden. In der Zeit davor und danach wird lediglich mit dem Datenbankserver kommuniziert.

Die Analyse hat gezeigt, dass die Verwendung von mehreren Nodes bei der Datentransformation nur dann zu einer Laufzeitreduzierung führt, wenn es ein rechenaufwändiger Importjob ist, wie zum Beispiel der *DailyMarketImport*. Bei anderen Jobs, wie zum Beispiel dem *HoldingHistoryImport* und dem *WatchHistoryImport*, gibt es eine geringe Laufzeitreduzierung. In diesen Jobs werden viele Daten aus der Datenbank geladen, kurz verarbeitet und anschließend werden viele Daten in die Datensenke geschrieben. In solchen Jobs ist der Datenbankserver der begrenzende Faktor.

Auch bei den Multi Node-Messungen ähneln die inkrementellen Jobs den historischen Gegenstücken, weshalb auch hier auf eine detaillierte Analyse der inkrementellen Imports verzichtet wird.

6.3.3. Streaming Hardware-Nutzung

Abschließend wird die gemessene Hardware-Nutzung des modifizierten Streaming-Jobs analysiert. Die zu analysierenden Messdaten sind in den Abbildungen 5.13, 5.14, 5.15 und 5.16 dargestellt. Da dieser Job - bis auf die Datenquelle - dasselbe tut, wie die Non-Streaming-Variante, sind ähnliche Details wie in der vorherigen Analyse der Single Node-Messungen aus Abschnitt 6.3.1 zu erwarten. Im Folgenden werden daher die Unterschiede in der Hardware-Nutzung zwischen der Streaming-Modifikation des Jobs *CustomerMgmtImport* und der Non-Streaming-Variante erläutert.

Zunächst wird auf die Messung der Talend-Implementation eingegangen. Vergleicht man die CPU-Nutzungsdiagramme der Abbildungen 5.9 und 5.13, dann fällt auf, dass sie einander sehr ähnlich sind. Sie unterscheiden sich lediglich zu Beginn des Jobs, wenn die Importdaten eingelesen werden. In der Messung des Streaming-Jobs ist zu dort zu sehen, dass auf dem Node *04* - auf dem der Kafka-Broker betrieben wird - in den ersten Sekunden eine Spitze in der CPU-Last aufgetreten ist. Zu diesem Zeitpunkt wurden die Importdaten vom Kafka-Broker an den Node *01* übertragen. Die anschließenden Verarbeitungsschritte des Streaming-Jobs sind dieselben, wie auch in der Non-Streaming-Implementation, weshalb die Hardware-Nutzungskurven nach den ersten circa 50 Sekunden nahezu identisch verlaufen.

Dieselben Beobachtungen werden in der Single Node-Messung der Gobblin-Spark-Implementation gemacht. Zu Beginn der Messung gibt es eine CPU-Last-Spitze auf dem Node *04* und nach den ersten circa 10 Sekunden ähnelt der Streaming-Job der Non-Streaming-Messung. Es gibt hingegen eine Veränderung in der Belegung des Arbeitsspeichers auf Node *01*. Während der Non-Streaming-Messung wurden höchstens 14,5GB belegt. Mit der Streaming-Implementation erhöht sich die RAM-Nutzung etwas, sodass knapp über 15GB auf dem Node *01* genutzt werden.

Bei den Single Node-Messungen gibt es also wenig Unterschiede zwischen der Streaming- und Non-Streaming-Implementation. Vergleicht man aber die Multi Node-Messungen miteinander, dann sind deutliche Unterschiede zu sehen. In der Analyse der Non-Streaming Multi Node-Implementation wurde entdeckt, dass die Importdaten nicht von allen beteiligten Nodes gleichmäßig verarbeitet wurden.

Die Streaming Multi Node-Messungen zeigen allerdings die erwartete Lastverteilung: Die CPU-Auslastung jedes beteiligten Nodes hat zu den gleichen Zeitpunkten Spitzen. Während die CPU-Nutzung der Nodes *05* und *06* gelegentlich auf knapp über 0% abfällt, reduziert sich die CPU-Last auf Node *01* lediglich auf circa 30%. Dieser Unterschied lässt sich damit erklären, dass der Node *01* zusätzlich zum Spark-Slave die Aufgabe des Spark-Masters und Spark-Drivers erfüllt.

Mit dieser Erkenntnis ist bestätigt worden, dass die Gobblin-Spark-Implementation des Jobs *CustomerMgmtImport* zu einer verteilten Verarbeitung fähig ist. Die Streaming- und Non-Streaming-Implementation unterscheiden sich ausschließlich in der Datenquelle, weshalb die Datenquelle als Ursache für die nicht-optimale Nutzung der Hardware-Ressourcen in der Non-Streaming-Ausführung identifiziert wird.

6.4. Verifikation der Hypothesen

Vor Beginn der Messungen wurden in Abschnitt 4.1 Hypothesen aufgestellt. Der Messaufbau, die visualisierten Messergebnisse und die vorhergegangene Analyse zielten darauf ab, die Hypothesen zu bestätigen oder widerlegen zu können.

Die folgenden Abschnitte sind analog zu den Hypothesen durchnummeriert. Zu Beginn eines Abschnitts wird außerdem die jeweilige Hypothese wiederholt, um für den Leser den Kontext herzustellen.

6.4.1. Hypothese A

> *Einer der Messkandidaten kann alle Importjobs der TPC-DI mit einer kürzeren Laufzeit durchführen, als die Vergleichsimplementation.*

Diese Hypothese bezieht sich auf die Non-Streaming-Implementationen der Jobs, denn die TPC-DI Spezifikation kennt keine Jobs, die einen Event-Stream als Datenquelle verwenden.

Die Abbildungen 5.1, 5.2 und 5.3 zeigen die Messwerte, die für diese Hypothese relevant sind. Sie wurden in Abschnitt 6.2.1 im Detail analysiert.

In der Analyse wurde der Schluss gezogen, dass die Frameworks in den einzelnen Jobs ihre Stärken und Schwächen zeigen. Aber keine der Implementationen hat alle Jobs mit einer kürzeren Laufzeit durchgeführt als die Vergleichsimplementation. Diese Hypothese wird daher **widerlegt**.

6.4.2. Hypothese B

> *Je größer das zu verarbeitende Datenset ist, desto proportional länger dauert der Import.*

Zur Überprüfung dieser Hypothese werden die Daten verwendet, die zuvor zur Berechnung der TCP-DI Metriken in den Tabellen 6.1 und 6.2 verwendet wurden. Anhand der Zeilenanzahl werden die Größenverhältnisse zwischen den Batches der Datensets (VE_x) berechnet und es wird das Verhältnis der benötigten Importzeit der einzelnen Batches (VR_x) berechnet. Die Formeln sind wie folgt definiert:

Verhältnis der Laufzeit zu pico $VE_x = E_x / E_{x,pico}$

Verhältnis der Zeilenanzahl zu pico $VR_x = R_x / R_{x,pico}$

Wobei x den jeweiligen Batch identifiziert (H,$I1$ oder $I2$). $E_{x,pico}$ und $R_{x,pico}$ stellen die Laufzeit und die Zeilenanzahl des Datensets *pico* dar. Die berechneten Faktoren sind daher relativ zum kleinsten Datenset *pico* zu verstehen.

Die berechneten Verhältnisse sind in den Tabellen 6.3 und 6.4 dargestellt. Die Anzahl der Zeilen eines Datensets (R_H, R_{I1} und R_{I2}) wurden aus der Tabelle 4.2 entnommen.

Tabelle 6.3.: Verhältnis zwischen Datensetgröße und Laufzeit für die Talend-Implementation mit Clustergröße 1

Datenset	E_H	E_{I1}	E_{I2}	VE_H	VE_{I1}	VE_{I2}	VR_H	VR_{I1}	VR_{I2}
pico	19.288,31	293,80	296,86	1,00	1,00	1,00	1,00	1,00	1,00
nano	55.122,16	781,39	775,68	2,86	2,66	2,61	2,05	2,02	2,01
micro	105.896,31	1.350,35	1.364,35	5,49	4,60	4,60	3,10	3,05	3,04

Tabelle 6.4.: Verhältnis zwischen Datensetgröße und Laufzeit für die Gobblin-Spark-Kombination mit Clustergröße 1

Datenset	E_H	E_{I1}	E_{I2}	VE_H	VE_{I1}	VE_{I2}	VR_H	VR_{I1}	VR_{I2}
pico	7.121,86	155,11	151,78	1,00	1,00	1,00	1,00	1,00	1,00
nano	32.175,84	344,30	348,44	4,52	2,22	2,30	2,05	2,02	2,01
micro	215.550,92	805,78	814,36	30,27	5,20	5,37	3,10	3,05	3,04

Um eine lineare Skalierung zwischen Laufzeit und Datensetgröße zu finden, wie sie die Hypothese erwartet, muss man die Werte VE_x und VR_x miteinander vergleichen. Wenn die Werte gleich sind, dann liegt die erwartete lineare Skalierung vor. Die relativen Zahlen beispielsweise der Talend-Messung lesen sich wie folgt: Das Datenset der *nano*-Messung ist im historischen Import 2,05-mal so groß wie das *pico*-Datenset. Die Talend-Implementation hat für den historischen Import im *nano*-Datenset allerdings 2,86-mal so lange gebraucht wie für *pico*. Die Abweichungen bei den Gobblin-Spark-Messungen sind einem VE_H von 30,27 im Datenset *micro* noch größer. Diese Hypothese ist aufgrund der zu großen Relationen **widerlegt**.

6.4.3. Hypothese C

> *Je mehr Nodes an einem Importjob beteiligt sind, desto kürzer wird die Laufzeit des Imports.*

Diese Hypothese betrifft nur die Gobblin-Spark-Implementation, da nur diese die Verarbeitung auf mehreren Nodes verteilen kann. Zur Überprüfung dieser Hypothese können die Abbildungen 6.2, 6.3 und 6.4 sowie die Analyse der Multi Node-Laufzeiten herangezogen werden.

Während der Analyse wurde der Schluss gezogen, dass nur bei wenigen Importjobs die Laufzeit durch mehr Nodes reduziert wird. Bei einigen Jobs wie zum Beispiel dem *WatchHistoryImport* in den Datensets *pico* und *nano* ist es am effektivsten eine Clustergröße von 2 zu wählen. Bei Jobs, die verhältnismäßig kleine Dateien verarbeiten wie der Job *DateImport* hat das erhöhen der Clustergröße einen negativen Effekt, sodass die Laufzeit *erhöht* wird. Letztendlich konnte eine lineare Skalierung nur bei dem verarbeitungsaufwändigen Job *DailyMarketImport* mit den Datensets *pico* und *nano* beobachtet werden. Die Hypothese ist daher **widerlegt**.

6.4.4. Hypothese D

Während eines Imports ist die CPU- und RAM-Last des ausführenden Nodes höher als die der restlichen Nodes.

Zur Überprüfung dieser Hypothese wird die Analyse der Hardware-Nutzung aus Abschnitt 6.3 herangezogen. Während der Ausführung jedes Jobs der Talend-Implementation ist die CPU-Last des ausführenden Nodes 01 knapp über 0%, während die auf dem MySQL-Server durchgängig 20-30% der CPU verwendet werden. In der Gobblin-Spark-Implementation erzeugen die rechenaufwändigen Jobs wie *DailyMarketImport* auf den ausführenden Nodes fast durchgängig eine CPU-Last von 100%

Die Hypothese trifft nur auf eine der Implementationen zu und ist damit **widerlegt**.

6.4.5. Hypothese E

Nimmt die Verarbeitung weniger Zeit in Anspruch, wenn ein Event-Stream statt eines Dateisystems als Datenquelle verwendet wird?

In dieser Hypothese werden die Laufzeiten des modifizierten Streaming-Jobs und der Non-Streaming-Variante miteinander verglichen. Dazu wird auf die Analyse der Streaming-Laufzeiten in Abschnitt 6.2.3 und dessen Abbildung 6.6 zurückgegriffen.

Während der Analyse der Streaming-Laufzeiten wurde der Schluss gezogen, dass die Streaming-Implementation des Jobs *CustomerMgmtImport* eine kleinere Laufzeit aufweist als die Non-Streaming-Implementation.

Bezogen auf den modifizierten Job ist die Hypothese daher **bestätigt**. Für die restlichen Jobs, für die es keine Streaming-Modifikation gibt, kann keine Aussage getroffen werden.

7. Fazit

Abschließend wird dieses Kapitel den Inhalt dieser Arbeit zusammenfassen und die Ergebnisse kompakt darstellen. Es wird außerdem ein Ausblick gestellt, welche weiteren Themen mit den Ergebnissen dieser Arbeit bearbeitet werden können. In der Analyse wurden Auffälligkeiten entdeckt, die in weiteren Forschungsarbeiten hinterfragt und beantwortet werden können.

7.1. Zusammenfassung

Im Rahmen dieser Arbeit wurde die ETL-Benchmark Spezifikation TPC-DI erstmals mit den Messkandidaten *Talend Open Studio for Big Data* und einer neuartigen Kombination aus *Apache Gobblin* und *Apache Spark* implementiert. Diese Implementationen wurden in einem komplexen Messaufbau mit drei unterschiedlich großen Datensets und in mehr als 50 Experimenten gemessen.

Während der Messungen wurden nicht nur die Laufzeiten der einzelnen Importjobs registriert, sondern es wurden auch die CPU- und RAM-Nutzung erfasst. Diese Messwerte wurden anschließend ausgewertet und detailliert analysiert. Im Rahmen der Analyse wurde herausgefunden, dass Talend sich für kleine Datenmengen eignet und den unverhältnismäßig aufwändigen Job *DailyMarketImport* ab einer Importdatengröße von 455MB. Bei der Durchführung von Imports, die rechenaufwändig sind - außer dem *DailyMarketImport* - , kann die Gobblin-Spark-Implementation eine kürzere Laufzeit aufweisen als die Talend-Implementation. Während die Gobblin-Spark-Imports dazu neigen die CPU konstant zu fast 100% zu nutzen und nur gelegentlich mit dem Datenbankserver kommuniziert, erzeugen die Talend-Jobs eine deutlich geringere CPU-Last von knapp über 0%. Die Talend-Jobs generieren auf dem Datenbankserver allerdings eine konstante CPU-Last von 20-30%.

Neben den Messungen, die einen Node nutzen, gibt es Messjobs mit der Gobblin-Spark-Implementation die auf mehreren Nodes ausgeführt werden. Diese Multi Node-Messungen wurden ebenfalls analysiert. Hier wurde festgestellt, dass die Nutzung von mehreren Nodes erst ab einem bestimmten Verarbeitungsaufwand zu einer Laufzeitreduzierung führt. Es konnte allerdings nur bei dem Job *DailyMarketImport* eine Korrelation zwischen der Anzahl der Nodes und der Laufzeit festgestellt werden. Bei vielen Jobs wird die meiste Zeit damit verbracht,

Importdaten aus der Datenbank zu lesen und die Ergebnisdaten zu schreiben, während die tatsächliche Datentransformation in sehr kurzer Zeit stattfindet. Bei solchen Jobs wurde der Datenbankserver als begrenzender Faktor identifiziert.

Ein ausgewählter Messjob wurde in beiden Implementationen so modifiziert, dass er statt einer Datei einen Event-Stream als Datenquelle verwendet. Die Messergebnisse zeigten, dass durch die Modifikation - bei gleichbleibender Verarbeitungslogik - in allen Datensetgrößen eine Laufzeitreduzierung im Vergleich zur Non-Streaming-Implementation auftritt.

Die Analyse dieser Arbeit hat außerdem festgestellt, dass mehr als 70% der Messlaufzeit vom Job *DailyMarketImport* genutzt wird. Dieser Importjob dauert sehr lange in der Ausführung und bietet damit ein hohes Fehlerpotenzial. Durch diesen Importjob dauert die Messung beispielsweise mit einem 10GB großen Datenset mehrere Wochen. Falls während der Messung Fehler auftreten, muss die Messung wiederholt werden, was zu einer Verzögerung der Benchmarkergebnisse führt. Dieses Problem hat auch Bleuel (2016) in seiner Arbeit angesprochen, der die TPC-DI mit dem Tool *Pentaho Kettle* implementiert hat. Besonders bei dem größten gemessenen Datenset *micro* hat der unverhältnismäßig lange Job für eine starke Verzerrung in den Gobblin-Spark-Messungen gesorgt.

In der TPC-DI-Spezifikation werden Skripte referenziert, die es auf der Webseite der TPC geben soll, die allerdings nicht im Rahmen dieser Arbeit gefunden werden konnten. Diese Skripte werden für die Phase des *automatisierten Audits* benötigt. Da diese Skripte nicht verfügbar sind, wurde das Audit im Rahmen dieser Arbeit nicht gemäß der Spezifikation durchgeführt.

Die TPC-DI definiert drei Metriken, anhand derer die Benchmarkergebnisse unterschiedlicher Implementationen miteinander verglichen werden können. Die Talend-Implementation hat in der Performance-Metrik TPC_DI_RPS für das Datenset *pico* den Wert $86\ row/s$ und in der Kosten-Performance-Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ den Wert $37,53\ €/row/s$ erreicht. Die Gobblin-Spark-Implementation hat in der Performance-Metrik TPC_DI_RPS für das Datenset *pico* den Wert $142\ row/s$ und in der Kosten-Performance-Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ den Wert $22,73\ €/row/s$ erzielt. Die Pentaho-Implementation von Bleuel (2016) hingegen hat für ein Datenset, das kleiner als *pico* ist, in der Performance-Metrik TPC_DI_RPS für das Datenset *pico* den Wert $13\ row/s$ und in der Kosten-Performance-Metrik $Price\text{-}per\text{-}TPC\_DI\_RPS$ den Wert $61,54\ €/row/s$ erreicht.

Die Gobblin-Spark-Implementation hat im Vergleich mit den bekannten ETL-Tools Talend und Pentaho gezeigt, dass sie auch ETL-Prozesse verarbeiten kann. Laut den TPC-DI-Metriken führt diese Kombination den Import der Datensets *pico* und *nano* schneller durch als die Vergleichsimplementationen. Mit dem größten gemessenen Datenset *micro* sinkt allerdings der

Wert TPC_DI_RPS der Gobblin-Spark-Implementation, sodass die Talend-Implementation erstmals die kürzere Gesamtlaufzeit aufweist.

7.2. Ausblick

Während des Schreibens dieser Arbeit und der Datenanalyse wurden Aspekte der bearbeiteten Thematik entdeckt, die nicht im Rahmen dieser Arbeit besprochen wurden. Im Folgenden gibt es daher eine Auflistung von Forschungsthemen, die fortgeführt werden können:

- Der Job *DailyMarketImport* hat die Laufzeit einer Messung um ein vielfaches verlängert und ein hohes Fehlerpotenzial geboten. Dadurch wird es erschwert, einen Benchmark nach der TPC-DI-Spezifikation mit großen Datensets durchzuführen. Eine weitere Arbeit kann darin bestehen, weitere Messungen mit größeren Datensets als 1,5GB durchzuführen, wobei der Job *DailyMarketImport* entweder ausgeschlossen wird oder modifiziert wird, sodass er weniger Zeit in Anspruch nimmt.
- Mit dem Datenset *micro* hat die Talend-Implementation eine kürzere Laufzeit als die Gobblin-Spark-Implementation. Es stellt sich die Frage, wie sich die Laufzeiten bei größeren Datensets verhalten. Sobald der Job *DailyMarketImport*, wie zuvor beschrieben, modifiziert wurde, sind solche Messungen durchführbar.
- Diese Arbeit hat für einen Job einen Event-Stream als Datenquelle verwendet, was zu einer Laufzeitverkürzung geführt hat. In einer weiteren Arbeit kann verifiziert werden, ob sich dieser Effekt auch auf andere Importjobs der TPC-DI-Spezifikation übertragen lässt. Ziel einer solchen Arbeit kann es sein, die komplette TPC-DI-Spezifikation auf eine Event-basierte Verarbeitung umzustellen.
- Diese Arbeit hat die Datenquelle der Spezifikation durch die Big Data-Frameworks *Apache Hadoop* und *Apache Kafka* ersetzt. Interessant ist, wie sich die Messergebnisse verhalten, wenn weitere Komponenten der Spezifikation ersetzt werden. Es kann beispielsweise statt einer relationalen Datenbank eine *NoSQL*-Datenbank als Datensenke verwendet werden.
- In dieser Arbeit wurde die TPC-DI-Spezifikation mit dem Framework *Apache Spark* implementiert. Die Messung kann mit weiteren OpenSource *Processing Engines* durchgeführt werden, wie zum Beispiel *Apache Flink*.

- Es ist denkbar, mit den beiden Messkandidaten einen weiteren Benchmark durchzuführen, der auf realen Anwendungsfällen und Daten basiert. Damit würde kein standardisierte Spezifikation verwendet werden, man hätte allerdings Ergebnisse, die auf realen Anwendungsbedingungen basieren.

Zu Beginn der Arbeit wurde festgestellt, dass die Themenbereiche Big Data und ETL ähnliche Problemstellungen haben. Vassiliadis und Simitsis (2009) schreiben über die Probleme von *Near Real Time ETL*, dass man die Daten eines ETL-Prozesses mit möglichst wenig Latenz braucht. Dasselbe Problem gibt es auch im *Near Real Time*-Bereich der Big Data-Welt. Es kann in einer weiteren Arbeit besprochen werden, warum nicht mehr Big Data-Technologien im ETL-Bereich verwendet werden. Möglicherweise werden an Datenintegrationstools andere versteckte Anforderungen gestellt, denen die Big Data-Tools nicht genügen.

Abkürzungsverzeichnis

ETL Extract-Transform-Load

ELT Extract-Load-Transform

FDR Full Disclosure Report

HDFS Hadoop Distributed File System

HR Human Resources

JVM Java Virtual Machine

KETL Kinetic ETL

SUT System Under Test

TPC Transaction Processing Performance Council

TPC-DI TPC Data Integration

A. Anhang

A.1. CD

Die CD beinhaltet:

- Diese Arbeit als PDF-Dokument
- Den Quellcode der Talend-Implementation
- Den Quellcode der Gobblin-Implementation
- Den Quellcode der Spark-Implementation
- Den Quellcode zur Ausführung der Messung
- Den Quellcode zur Einrichtung der Nodes
- Die Messergebnisse
- Die Jupyter-Notebooks zur Analyse der Messergebnisse

A.2. Messergebnisse

A.2.1. Laufzeiten tabellarisch

Tabelle A.1.: Laufzeiten aller historischen Non-Streaming-Imports im Datenset *pico* mit Clustergröße 1

Importjob	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CashTransactionImport	gobblin-spark	pico	1	68.3	66.0	70.0
CashTransactionImport	talend	pico	1	2639.3	2626.0	2664.0
CustomerMgmtImport	gobblin-spark	pico	1	183.7	182.0	186.0
CustomerMgmtImport	talend	pico	1	171.3	171.0	172.0
DailyMarketImport	gobblin-spark	pico	1	6233.7	6036.0	6419.0
DailyMarketImport	talend	pico	1	14101.3	14052.0	14166.0
DateImport	gobblin-spark	pico	1	17.3	16.0	19.0
DateImport	talend	pico	1	1.0	1.0	1.0
FinwireImport	gobblin-spark	pico	1	85.7	84.0	88.0
FinwireImport	talend	pico	1	38.3	38.0	39.0
HRImport	gobblin-spark	pico	1	2.7	2.0	3.0
HRImport	talend	pico	1	0.0	0.0	0.0
HoldingHistoryImport	gobblin-spark	pico	1	88.7	87.0	91.0
HoldingHistoryImport	talend	pico	1	40.7	40.0	41.0
IndustryImport	gobblin-spark	pico	1	0.0	0.0	0.1
IndustryImport	talend	pico	1	0.0	0.0	0.0
ProspectImport	gobblin-spark	pico	1	21.4	21.0	21.6
ProspectImport	talend	pico	1	41.0	40.0	42.0
StatusTypeImport	gobblin-spark	pico	1	0.1	0.1	0.1
StatusTypeImport	talend	pico	1	0.2	0.2	0.2
TaxRateImport	gobblin-spark	pico	1	0.8	0.7	0.8
TaxRateImport	talend	pico	1	0.0	0.0	0.0
TimeImport	gobblin-spark	pico	1	34.7	34.0	35.0
TimeImport	talend	pico	1	2.7	2.0	3.0
TradeImport	gobblin-spark	pico	1	307.3	301.0	315.0
TradeImport	talend	pico	1	2197.3	2177.0	2212.0
TradeTypeImport	gobblin-spark	pico	1	0.2	0.1	0.5
TradeTypeImport	talend	pico	1	0.0	0.0	0.0
WatchHistoryImport	gobblin-spark	pico	1	265.7	257.0	273.0
WatchHistoryImport	talend	pico	1	55.0	55.0	55.0

Tabelle A.2.: Laufzeiten aller inkrementellen Non-Streaming-Imports im Datenset *pico* mit Clustergröße 1

Importjob	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
AccountImport	gobblin-spark	pico	1	4.4	4.0	5.1
AccountImport	talend	pico	1	0.0	0.0	0.0
CashTransactionImport	gobblin-spark	pico	1	12.7	12.0	14.1
CashTransactionImport	talend	pico	1	1.0	1.0	1.0
CustomerImport	gobblin-spark	pico	1	17.0	16.1	17.5
CustomerImport	talend	pico	1	0.2	0.0	0.4
DailyMarketImport	gobblin-spark	pico	1	66.9	64.0	70.0
DailyMarketImport	talend	pico	1	245.8	242.0	249.0
HoldingHistoryImport	gobblin-spark	pico	1	13.1	13.1	13.1
HoldingHistoryImport	talend	pico	1	4.7	4.0	6.0
ProspectImport	gobblin-spark	pico	1	28.3	26.0	29.0
ProspectImport	talend	pico	1	38.8	36.0	41.0
TradeImport	gobblin-spark	pico	1	31.8	30.0	33.3
TradeImport	talend	pico	1	1.3	1.0	2.0
WatchHistoryImport	gobblin-spark	pico	1	10.8	10.0	12.0
WatchHistoryImport	talend	pico	1	3.2	3.0	4.0

Tabelle A.3.: Laufzeiten aller historischen Non-Streaming-Imports im Datenset *nano* mit Clustergröße 1

Importjob	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CashTransactionImport	gobblin-spark	nano	1	149.0	149.0	149.0
CashTransactionImport	talend	nano	1	9013.7	8988.0	9046.0
CustomerMgmtImport	gobblin-spark	nano	1	432.0	419.0	444.0
CustomerMgmtImport	talend	nano	1	600.0	597.0	602.0
DailyMarketImport	gobblin-spark	nano	1	30515.0	28318.0	31983.0
DailyMarketImport	talend	nano	1	40189.7	39886.0	40729.0
DateImport	gobblin-spark	nano	1	17.1	15.0	18.6
DateImport	talend	nano	1	0.3	0.0	1.0
FinwireImport	gobblin-spark	nano	1	113.0	110.0	116.0
FinwireImport	talend	nano	1	86.7	86.0	88.0
HRImport	gobblin-spark	nano	1	6.3	6.0	7.0
HRImport	talend	nano	1	0.3	0.0	0.9
HoldingHistoryImport	gobblin-spark	nano	1	262.7	258.0	267.0
HoldingHistoryImport	talend	nano	1	126.7	124.0	130.0
IndustryImport	gobblin-spark	nano	1	0.4	0.0	0.7
IndustryImport	talend	nano	1	0.0	0.0	0.0
ProspectImport	gobblin-spark	nano	1	33.3	33.0	34.0
ProspectImport	talend	nano	1	78.3	75.0	80.0
StatusTypeImport	gobblin-spark	nano	1	0.3	0.1	0.7
StatusTypeImport	talend	nano	1	0.1	0.0	0.2
TaxRateImport	gobblin-spark	nano	1	0.4	0.0	1.1
TaxRateImport	talend	nano	1	0.0	0.0	0.0
TimeImport	gobblin-spark	nano	1	33.3	32.0	34.0
TimeImport	talend	nano	1	2.7	2.0	3.0
TradeImport	gobblin-spark	nano	1	622.0	610.0	635.0
TradeImport	talend	nano	1	4901.0	4865.0	4940.0
TradeTypeImport	gobblin-spark	nano	1	0.7	0.5	0.9
TradeTypeImport	talend	nano	1	0.0	0.0	0.0
WatchHistoryImport	gobblin-spark	nano	1	435.3	432.0	440.0
WatchHistoryImport	talend	nano	1	122.7	121.0	125.0

Tabelle A.4.: Laufzeiten aller inkrementellen Non-Streaming-Imports im Datenset *nano* mit Clustergröße 1

Importjob	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
AccountImport	gobblin-spark	nano	1	5.2	5.0	6.1
AccountImport	talend	nano	1	0.8	0.0	1.0
CashTransactionImport	gobblin-spark	nano	1	21.1	20.0	22.1
CashTransactionImport	talend	nano	1	5.0	5.0	5.0
CustomerImport	gobblin-spark	nano	1	21.6	21.1	22.1
CustomerImport	talend	nano	1	0.2	0.0	0.7
DailyMarketImport	gobblin-spark	nano	1	221.7	217.3	231.3
DailyMarketImport	talend	nano	1	671.0	669.0	674.0
HoldingHistoryImport	gobblin-spark	nano	1	25.1	23.1	28.1
HoldingHistoryImport	talend	nano	1	11.2	10.0	12.0
ProspectImport	gobblin-spark	nano	1	31.5	31.0	32.0
ProspectImport	talend	nano	1	77.3	71.0	83.0
TradeImport	gobblin-spark	nano	1	56.7	52.4	59.4
TradeImport	talend	nano	1	3.0	2.0	4.0
WatchHistoryImport	gobblin-spark	nano	1	19.9	18.0	23.2
WatchHistoryImport	talend	nano	1	9.7	7.0	13.0

Tabelle A.5.: Laufzeiten aller historischen Non-Streaming-Imports im Datenset *micro* mit Clustergröße 1

Importjob	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CashTransactionImport	gobblin-spark	micro	1	297.0	297.0	297.0
CashTransactionImport	talend	micro	1	19057.0	19057.0	19057.0
CustomerMgmtImport	gobblin-spark	micro	1	801.0	801.0	801.0
CustomerMgmtImport	talend	micro	1	1276.0	1276.0	1276.0
DailyMarketImport	gobblin-spark	micro	1	212774.0	212774.0	212774.0
DailyMarketImport	talend	micro	1	76996.0	76996.0	76996.0
DateImport	gobblin-spark	micro	1	17.7	17.7	17.7
DateImport	talend	micro	1	1.0	1.0	1.0
FinwireImport	gobblin-spark	micro	1	149.0	149.0	149.0
FinwireImport	talend	micro	1	142.0	142.0	142.0
HRImport	gobblin-spark	micro	1	9.0	9.0	9.0
HRImport	talend	micro	1	0.0	0.0	0.0
HoldingHistoryImport	gobblin-spark	micro	1	467.0	467.0	467.0
HoldingHistoryImport	talend	micro	1	312.0	312.0	312.0
IndustryImport	gobblin-spark	micro	1	1.0	1.0	1.0
IndustryImport	talend	micro	1	0.0	0.0	0.0
ProspectImport	gobblin-spark	micro	1	44.0	44.0	44.0
ProspectImport	talend	micro	1	118.0	118.0	118.0
StatusTypeImport	gobblin-spark	micro	1	0.1	0.1	0.1
StatusTypeImport	talend	micro	1	0.2	0.2	0.2
TaxRateImport	gobblin-spark	micro	1	0.1	0.1	0.1
TaxRateImport	talend	micro	1	0.0	0.0	0.0
TimeImport	gobblin-spark	micro	1	34.0	34.0	34.0
TimeImport	talend	micro	1	2.0	2.0	2.0
TradeImport	gobblin-spark	micro	1	1038.0	1038.0	1038.0
TradeImport	talend	micro	1	7779.0	7779.0	7779.0
TradeTypeImport	gobblin-spark	micro	1	0.1	0.1	0.1
TradeTypeImport	talend	micro	1	0.0	0.0	0.0
WatchHistoryImport	gobblin-spark	micro	1	708.0	708.0	708.0
WatchHistoryImport	talend	micro	1	213.0	213.0	213.0

Tabelle A.6.: Laufzeiten aller inkrementellen Non-Streaming-Imports im Datenset *micro* mit Clustergröße 1

Importjob	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
AccountImport	gobblin-spark	micro	1	7.1	7.1	7.1
AccountImport	talend	micro	1	2.0	2.0	2.0
CashTransactionImport	gobblin-spark	micro	1	26.3	24.5	28.1
CashTransactionImport	talend	micro	1	12.0	12.0	12.0
CustomerImport	gobblin-spark	micro	1	24.5	24.0	25.1
CustomerImport	talend	micro	1	0.5	0.0	1.0
DailyMarketImport	gobblin-spark	micro	1	636.7	634.0	639.4
DailyMarketImport	talend	micro	1	1186.5	1185.0	1188.0
HoldingHistoryImport	gobblin-spark	micro	1	40.6	38.1	43.1
HoldingHistoryImport	talend	micro	1	17.5	15.0	20.0
ProspectImport	gobblin-spark	micro	1	45.0	44.0	46.0
ProspectImport	talend	micro	1	118.0	117.0	119.0
TradeImport	gobblin-spark	micro	1	81.0	78.0	84.0
TradeImport	talend	micro	1	5.0	4.0	6.0
WatchHistoryImport	gobblin-spark	micro	1	29.8	28.7	31.0
WatchHistoryImport	talend	micro	1	15.5	11.0	20.0

Tabelle A.7.: Summierte Laufzeit aller Non-Streaming-Imports über alle Datensets mit Clustergröße 1

Framework	Datenset	Clustergröße	BatchID	Mean (in s)	Min (in s)	Max (in s)
gobblin-spark	micro	1	1	216339.9	216339.9	216339.9
gobblin-spark	micro	1	2	883.8	883.8	883.8
gobblin-spark	micro	1	3	898.4	898.4	898.4
gobblin-spark	nano	1	1	32620.8	30391.7	34093.8
gobblin-spark	nano	1	2	398.0	391.0	408.0
gobblin-spark	nano	1	3	407.4	404.2	410.2
gobblin-spark	pico	1	1	7310.2	7125.1	7472.6
gobblin-spark	pico	1	2	185.8	185.1	186.3
gobblin-spark	pico	1	3	184.4	179.9	190.0
talend	micro	1	1	105896.3	105896.3	105896.3
talend	micro	1	2	1350.0	1350.0	1350.0
talend	micro	1	3	1364.0	1364.0	1364.0
talend	nano	1	1	55122.1	54819.1	55628.1
talend	nano	1	2	781.2	778.7	783.0
talend	nano	1	3	775.3	775.0	776.0
talend	pico	1	1	19288.3	19206.3	19355.3
talend	pico	1	2	293.5	290.0	298.0
talend	pico	1	3	296.6	294.4	300.4

Tabelle A.8.: Laufzeiten aller historischen Non-Streaming-Imports aller Clustergrößen mit Spark-Framework im Datenset *pico*

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CashTransactionImport	gobblin-spark	pico	1	68.3	66.0	70.0
CashTransactionImport	gobblin-spark	pico	2	40.7	40.0	41.0
CashTransactionImport	gobblin-spark	pico	3	36.3	36.0	37.0
CustomerMgmtImport	gobblin-spark	pico	1	183.7	182.0	186.0
CustomerMgmtImport	gobblin-spark	pico	2	186.0	185.0	188.0
CustomerMgmtImport	gobblin-spark	pico	3	187.3	186.0	188.0
DailyMarketImport	gobblin-spark	pico	1	6233.7	6036.0	6419.0
DailyMarketImport	gobblin-spark	pico	2	3053.3	3016.0	3081.0
DailyMarketImport	gobblin-spark	pico	3	2130.3	2061.0	2223.0

DateImport	gobblin-spark	pico	1	17.3	16.0	19.0
DateImport	gobblin-spark	pico	2	17.9	17.0	18.6
DateImport	gobblin-spark	pico	3	19.5	18.6	21.0
FinwireImport	gobblin-spark	pico	1	85.7	84.0	88.0
FinwireImport	gobblin-spark	pico	2	59.7	59.0	61.0
FinwireImport	gobblin-spark	pico	3	56.0	55.0	58.0
HRImport	gobblin-spark	pico	1	2.7	2.0	3.0
HRImport	gobblin-spark	pico	2	3.5	3.0	4.5
HRImport	gobblin-spark	pico	3	3.2	3.0	3.5
HoldingHistoryImport	gobblin-spark	pico	1	88.7	87.0	91.0
HoldingHistoryImport	gobblin-spark	pico	2	88.7	85.0	91.0
HoldingHistoryImport	gobblin-spark	pico	3	88.0	83.0	91.0
IndustryImport	gobblin-spark	pico	1	0.0	0.0	0.1
IndustryImport	gobblin-spark	pico	2	2.4	2.0	3.0
IndustryImport	gobblin-spark	pico	3	2.4	1.1	3.1
ProspectImport	gobblin-spark	pico	1	21.4	21.0	21.6
ProspectImport	gobblin-spark	pico	2	21.2	20.0	22.0
ProspectImport	gobblin-spark	pico	3	21.9	20.0	23.0
StatusTypeImport	gobblin-spark	pico	1	0.1	0.1	0.1
StatusTypeImport	gobblin-spark	pico	2	0.5	0.1	0.7
StatusTypeImport	gobblin-spark	pico	3	1.4	0.1	3.1
TaxRateImport	gobblin-spark	pico	1	0.8	0.7	0.8
TaxRateImport	gobblin-spark	pico	2	0.5	0.1	1.2
TaxRateImport	gobblin-spark	pico	3	0.0	0.0	0.1
TimeImport	gobblin-spark	pico	1	34.7	34.0	35.0
TimeImport	gobblin-spark	pico	2	33.7	30.0	36.0
TimeImport	gobblin-spark	pico	3	36.3	35.0	38.0
TradeImport	gobblin-spark	pico	1	307.3	301.0	315.0
TradeImport	gobblin-spark	pico	2	249.3	242.0	254.0
TradeImport	gobblin-spark	pico	3	274.7	264.0	293.0
TradeTypeImport	gobblin-spark	pico	1	0.2	0.1	0.5
TradeTypeImport	gobblin-spark	pico	2	0.4	0.1	0.6
TradeTypeImport	gobblin-spark	pico	3	0.2	0.1	0.6
WatchHistoryImport	gobblin-spark	pico	1	265.7	257.0	273.0
WatchHistoryImport	gobblin-spark	pico	2	220.0	218.0	222.0

WatchHistoryImport	gobblin-spark	pico	3	238.3	230.0	248.0

Tabelle A.9.: Laufzeiten aller inkrementellen Non-Streaming-Imports aller Clustergrößen mit Spark-Framework im Datenset *pico*

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
AccountImport	gobblin-spark	pico	1	4.4	4.0	5.1
AccountImport	gobblin-spark	pico	2	4.6	4.1	5.1
AccountImport	gobblin-spark	pico	3	5.1	4.1	6.1
CashTransactionImport	gobblin-spark	pico	1	12.7	12.0	14.1
CashTransactionImport	gobblin-spark	pico	2	11.3	11.1	12.1
CashTransactionImport	gobblin-spark	pico	3	11.6	11.1	12.1
CustomerImport	gobblin-spark	pico	1	17.0	16.1	17.5
CustomerImport	gobblin-spark	pico	2	20.5	20.1	21.1
CustomerImport	gobblin-spark	pico	3	23.3	22.1	24.1
DailyMarketImport	gobblin-spark	pico	1	66.9	64.0	70.0
DailyMarketImport	gobblin-spark	pico	2	69.5	66.2	75.2
DailyMarketImport	gobblin-spark	pico	3	69.8	67.2	74.2
HoldingHistoryImport	gobblin-spark	pico	1	13.1	13.1	13.1
HoldingHistoryImport	gobblin-spark	pico	2	13.6	13.1	14.1
HoldingHistoryImport	gobblin-spark	pico	3	15.4	13.1	18.1
ProspectImport	gobblin-spark	pico	1	28.3	26.0	29.0
ProspectImport	gobblin-spark	pico	2	27.2	26.0	28.0
ProspectImport	gobblin-spark	pico	3	28.7	27.0	31.0
TradeImport	gobblin-spark	pico	1	31.8	30.0	33.3
TradeImport	gobblin-spark	pico	2	32.9	31.0	34.4
TradeImport	gobblin-spark	pico	3	34.7	32.3	37.3
WatchHistoryImport	gobblin-spark	pico	1	10.8	10.0	12.0
WatchHistoryImport	gobblin-spark	pico	2	11.4	11.0	12.2
WatchHistoryImport	gobblin-spark	pico	3	11.4	10.1	13.1

Tabelle A.10.: Laufzeiten aller historischen Non-Streaming-Imports aller Clustergrößen mit Spark-Framework im Datenset *nano*

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CashTransactionImport	gobblin-spark	nano	1	149.0	149.0	149.0
CashTransactionImport	gobblin-spark	nano	2	94.3	94.0	95.0
CashTransactionImport	gobblin-spark	nano	3	83.7	83.0	85.0
CustomerMgmtImport	gobblin-spark	nano	1	432.0	419.0	444.0
CustomerMgmtImport	gobblin-spark	nano	2	435.0	429.0	442.0
CustomerMgmtImport	gobblin-spark	nano	3	438.7	425.0	452.0
DailyMarketImport	gobblin-spark	nano	1	30515.0	28318.0	31983.0
DailyMarketImport	gobblin-spark	nano	2	14333.3	13819.0	14595.0
DailyMarketImport	gobblin-spark	nano	3	10373.0	10142.0	10552.0
DateImport	gobblin-spark	nano	1	17.1	15.0	18.6
DateImport	gobblin-spark	nano	2	17.5	17.0	18.0
DateImport	gobblin-spark	nano	3	18.4	18.0	18.6
FinwireImport	gobblin-spark	nano	1	113.0	110.0	116.0
FinwireImport	gobblin-spark	nano	2	71.3	70.0	72.0
FinwireImport	gobblin-spark	nano	3	68.3	66.0	70.0
HRImport	gobblin-spark	nano	1	6.3	6.0	7.0
HRImport	gobblin-spark	nano	2	6.3	6.0	7.0
HRImport	gobblin-spark	nano	3	6.7	6.0	8.0
HoldingHistoryImport	gobblin-spark	nano	1	262.7	258.0	267.0
HoldingHistoryImport	gobblin-spark	nano	2	173.7	169.0	181.0
HoldingHistoryImport	gobblin-spark	nano	3	162.7	155.0	171.0
IndustryImport	gobblin-spark	nano	1	0.4	0.0	0.7
IndustryImport	gobblin-spark	nano	2	1.7	0.1	3.1
IndustryImport	gobblin-spark	nano	3	2.7	1.5	3.5
ProspectImport	gobblin-spark	nano	1	33.3	33.0	34.0
ProspectImport	gobblin-spark	nano	2	32.3	32.0	33.0
ProspectImport	gobblin-spark	nano	3	32.0	32.0	32.0
StatusTypeImport	gobblin-spark	nano	1	0.3	0.1	0.7
StatusTypeImport	gobblin-spark	nano	2	0.5	0.1	0.7
StatusTypeImport	gobblin-spark	nano	3	1.6	0.7	3.2
TaxRateImport	gobblin-spark	nano	1	0.4	0.0	1.1
TaxRateImport	gobblin-spark	nano	2	0.2	0.1	0.5
TaxRateImport	gobblin-spark	nano	3	0.0	0.0	0.1

TimeImport	gobblin-spark	nano	1	33.3	32.0	34.0
TimeImport	gobblin-spark	nano	2	28.3	27.0	31.0
TimeImport	gobblin-spark	nano	3	33.7	33.0	35.0
TradeImport	gobblin-spark	nano	1	622.0	610.0	635.0
TradeImport	gobblin-spark	nano	2	453.0	439.0	466.0
TradeImport	gobblin-spark	nano	3	453.3	442.0	463.0
TradeTypeImport	gobblin-spark	nano	1	0.7	0.5	0.9
TradeTypeImport	gobblin-spark	nano	2	0.0	0.0	0.1
TradeTypeImport	gobblin-spark	nano	3	0.9	0.6	1.5
WatchHistoryImport	gobblin-spark	nano	1	435.3	432.0	440.0
WatchHistoryImport	gobblin-spark	nano	2	249.7	244.0	255.0
WatchHistoryImport	gobblin-spark	nano	3	256.0	248.0	260.0

Tabelle A.11.: Laufzeiten aller inkrementellen Non-Streaming-Imports aller Clustergrößen mit Spark-Framework im Datenset *nano*

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
AccountImport	gobblin-spark	nano	1	5.2	5.0	6.1
AccountImport	gobblin-spark	nano	2	5.8	5.1	6.1
AccountImport	gobblin-spark	nano	3	5.9	5.1	6.1
CashTransactionImport	gobblin-spark	nano	1	21.1	20.0	22.1
CashTransactionImport	gobblin-spark	nano	2	20.2	19.0	22.1
CashTransactionImport	gobblin-spark	nano	3	21.4	20.1	23.1
CustomerImport	gobblin-spark	nano	1	21.6	21.1	22.1
CustomerImport	gobblin-spark	nano	2	25.2	24.1	26.1
CustomerImport	gobblin-spark	nano	3	28.1	27.1	29.1
DailyMarketImport	gobblin-spark	nano	1	221.7	217.3	231.3
DailyMarketImport	gobblin-spark	nano	2	219.2	207.3	229.3
DailyMarketImport	gobblin-spark	nano	3	213.7	201.3	221.3
HoldingHistoryImport	gobblin-spark	nano	1	25.1	23.1	28.1
HoldingHistoryImport	gobblin-spark	nano	2	26.6	26.0	27.1
HoldingHistoryImport	gobblin-spark	nano	3	28.5	27.1	29.1
ProspectImport	gobblin-spark	nano	1	31.5	31.0	32.0
ProspectImport	gobblin-spark	nano	2	26.5	25.0	28.0
ProspectImport	gobblin-spark	nano	3	26.7	25.0	28.0
TradeImport	gobblin-spark	nano	1	56.7	52.4	59.4
TradeImport	gobblin-spark	nano	2	60.8	54.0	67.0
TradeImport	gobblin-spark	nano	3	62.9	58.0	66.4
WatchHistoryImport	gobblin-spark	nano	1	19.9	18.0	23.2
WatchHistoryImport	gobblin-spark	nano	2	23.1	21.2	24.2
WatchHistoryImport	gobblin-spark	nano	3	23.7	21.2	26.2

Tabelle A.12.: Laufzeiten aller historischen Non-Streaming-Imports aller Clustergrößen mit Spark-Framework im Datenset *micro*

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CashTransactionImport	gobblin-spark	micro	1	297.0	297.0	297.0
CashTransactionImport	gobblin-spark	micro	2	192.0	192.0	192.0
CashTransactionImport	gobblin-spark	micro	3	183.0	183.0	183.0

CustomerMgmtImport	gobblin-spark	micro	1	801.0	801.0	801.0
CustomerMgmtImport	gobblin-spark	micro	2	757.0	757.0	757.0
CustomerMgmtImport	gobblin-spark	micro	3	754.0	754.0	754.0
DailyMarketImport	gobblin-spark	micro	1	212774.0	212774.0	212774.0
DailyMarketImport	gobblin-spark	micro	2	156259.0	156259.0	156259.0
DailyMarketImport	gobblin-spark	micro	3	87136.0	87136.0	87136.0
DateImport	gobblin-spark	micro	1	17.7	17.7	17.7
DateImport	gobblin-spark	micro	2	16.6	16.6	16.6
DateImport	gobblin-spark	micro	3	16.0	16.0	16.0
FinwireImport	gobblin-spark	micro	1	149.0	149.0	149.0
FinwireImport	gobblin-spark	micro	2	87.0	87.0	87.0
FinwireImport	gobblin-spark	micro	3	78.0	78.0	78.0
HRImport	gobblin-spark	micro	1	9.0	9.0	9.0
HRImport	gobblin-spark	micro	2	9.0	9.0	9.0
HRImport	gobblin-spark	micro	3	10.0	10.0	10.0
HoldingHistoryImport	gobblin-spark	micro	1	467.0	467.0	467.0
HoldingHistoryImport	gobblin-spark	micro	2	318.0	318.0	318.0
HoldingHistoryImport	gobblin-spark	micro	3	314.0	314.0	314.0
IndustryImport	gobblin-spark	micro	1	1.0	1.0	1.0
IndustryImport	gobblin-spark	micro	2	0.1	0.1	0.1
IndustryImport	gobblin-spark	micro	3	4.0	4.0	4.0
ProspectImport	gobblin-spark	micro	1	44.0	44.0	44.0
ProspectImport	gobblin-spark	micro	2	43.0	43.0	43.0
ProspectImport	gobblin-spark	micro	3	44.0	44.0	44.0
StatusTypeImport	gobblin-spark	micro	1	0.1	0.1	0.1
StatusTypeImport	gobblin-spark	micro	2	2.1	2.1	2.1
StatusTypeImport	gobblin-spark	micro	3	0.0	0.0	0.0
TaxRateImport	gobblin-spark	micro	1	0.1	0.1	0.1
TaxRateImport	gobblin-spark	micro	2	0.1	0.1	0.1
TaxRateImport	gobblin-spark	micro	3	0.1	0.1	0.1
TimeImport	gobblin-spark	micro	1	34.0	34.0	34.0
TimeImport	gobblin-spark	micro	2	33.0	33.0	33.0
TimeImport	gobblin-spark	micro	3	31.0	31.0	31.0
TradeImport	gobblin-spark	micro	1	1038.0	1038.0	1038.0
TradeImport	gobblin-spark	micro	2	776.0	776.0	776.0

TradeImport	gobblin-spark	micro	3	762.0	762.0	762.0
TradeTypeImport	gobblin-spark	micro	1	0.1	0.1	0.1
TradeTypeImport	gobblin-spark	micro	2	0.1	0.1	0.1
TradeTypeImport	gobblin-spark	micro	3	0.1	0.1	0.1
WatchHistoryImport	gobblin-spark	micro	1	708.0	708.0	708.0
WatchHistoryImport	gobblin-spark	micro	2	435.0	435.0	435.0
WatchHistoryImport	gobblin-spark	micro	3	419.0	419.0	419.0

Tabelle A.13.: Laufzeiten aller inkrementellen Non-Streaming-Imports aller Clustergrößen mit Spark-Framework im Datenset *micro*

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
AccountImport	gobblin-spark	micro	1	7.1	7.1	7.1
AccountImport	gobblin-spark	micro	2	6.5	6.0	7.1
AccountImport	gobblin-spark	micro	3	8.1	7.1	9.1
CashTransactionImport	gobblin-spark	micro	1	26.3	24.5	28.1
CashTransactionImport	gobblin-spark	micro	2	25.1	24.1	26.1
CashTransactionImport	gobblin-spark	micro	3	27.1	26.1	28.1
CustomerImport	gobblin-spark	micro	1	24.5	24.0	25.1
CustomerImport	gobblin-spark	micro	2	28.1	28.1	28.1
CustomerImport	gobblin-spark	micro	3	31.0	30.0	32.1
DailyMarketImport	gobblin-spark	micro	1	636.7	634.0	639.4
DailyMarketImport	gobblin-spark	micro	2	654.2	638.4	670.0
DailyMarketImport	gobblin-spark	micro	3	652.7	632.3	673.0
HoldingHistoryImport	gobblin-spark	micro	1	40.6	38.1	43.1
HoldingHistoryImport	gobblin-spark	micro	2	46.1	45.1	47.1
HoldingHistoryImport	gobblin-spark	micro	3	43.6	41.1	46.0
ProspectImport	gobblin-spark	micro	1	45.0	44.0	46.0
ProspectImport	gobblin-spark	micro	2	37.5	37.0	38.0
ProspectImport	gobblin-spark	micro	3	39.5	37.0	42.0
TradeImport	gobblin-spark	micro	1	81.0	78.0	84.0
TradeImport	gobblin-spark	micro	2	83.2	81.5	85.0
TradeImport	gobblin-spark	micro	3	88.3	79.5	97.0
WatchHistoryImport	gobblin-spark	micro	1	29.8	28.7	31.0
WatchHistoryImport	gobblin-spark	micro	2	31.1	26.3	36.0
WatchHistoryImport	gobblin-spark	micro	3	32.6	27.0	38.3

Tabelle A.14.: Laufzeiten aller Streaming-Imports mit der Clustergröße 1

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CustomerMgmtByStreamImport	gobblin-spark	micro	1	587.0	587.0	587.0
CustomerMgmtByStreamImport	gobblin-spark	nano	1	360.3	358.0	363.0
CustomerMgmtByStreamImport	gobblin-spark	pico	1	174.0	171.0	176.0
CustomerMgmtByStreamImport	talend	micro	1	1090.0	1090.0	1090.0
CustomerMgmtByStreamImport	talend	nano	1	534.0	533.0	536.0
CustomerMgmtByStreamImport	talend	pico	1	163.0	162.0	164.0

Tabelle A.15.: Laufzeiten aller Streaming-Imports aller Clustergrößen mit Spark-Framework

Jobname	Framework	Datenset	Clustergröße	Mean (in s)	Min (in s)	Max (in s)
CustomerMgmtByStreamImport	gobblin-spark	micro	1	587.0	587.0	587.0
CustomerMgmtByStreamImport	gobblin-spark	micro	2	560.0	560.0	560.0
CustomerMgmtByStreamImport	gobblin-spark	micro	3	537.0	537.0	537.0
CustomerMgmtByStreamImport	gobblin-spark	nano	1	360.3	358.0	363.0
CustomerMgmtByStreamImport	gobblin-spark	nano	2	331.3	321.0	339.0
CustomerMgmtByStreamImport	gobblin-spark	nano	3	312.7	301.0	322.0
CustomerMgmtByStreamImport	gobblin-spark	pico	1	174.0	171.0	176.0
CustomerMgmtByStreamImport	gobblin-spark	pico	2	152.3	147.0	157.0
CustomerMgmtByStreamImport	gobblin-spark	pico	3	143.7	143.0	145.0

A.2.2. Hardware-Nutzung visuell

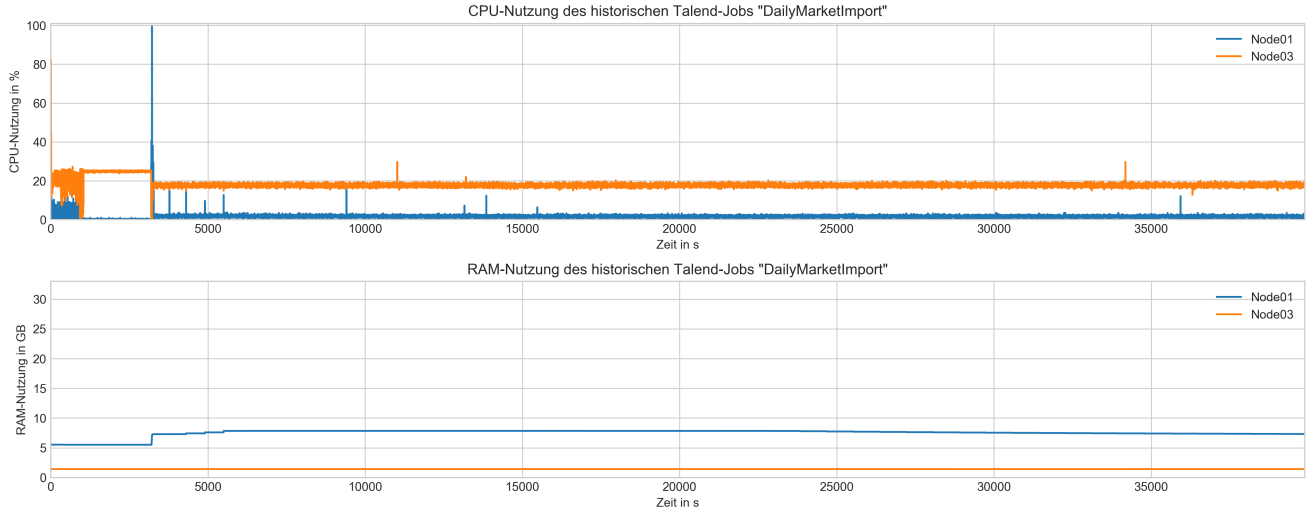

Abbildung A.1.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *DailyMarketImport* mit Talend-Framework im Datenset *nano* mit der Clustergröße 1

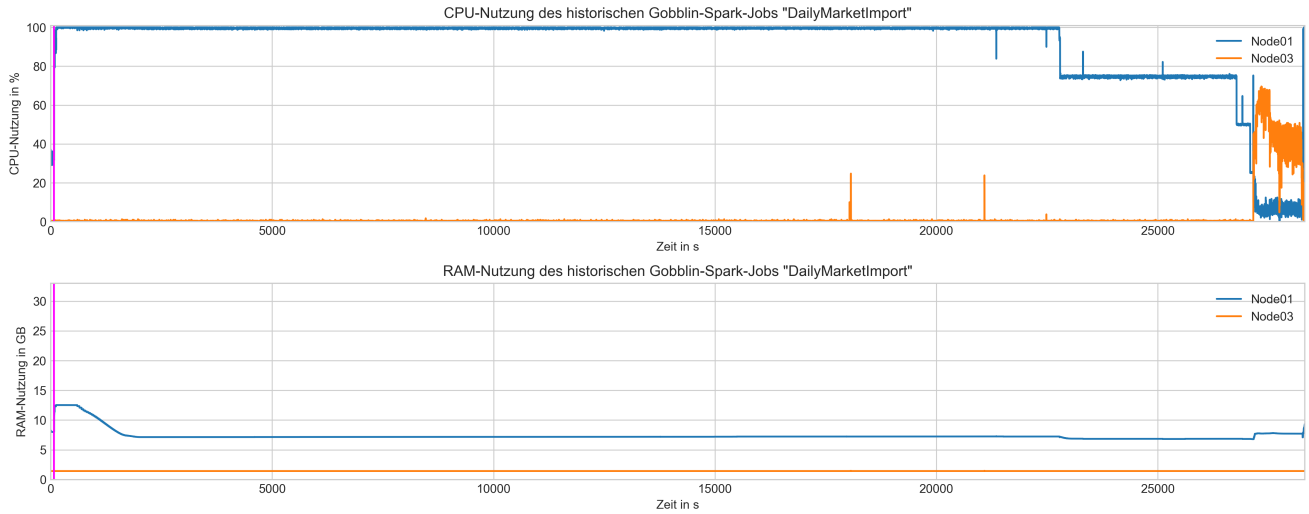

Abbildung A.2.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *DailyMarketImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

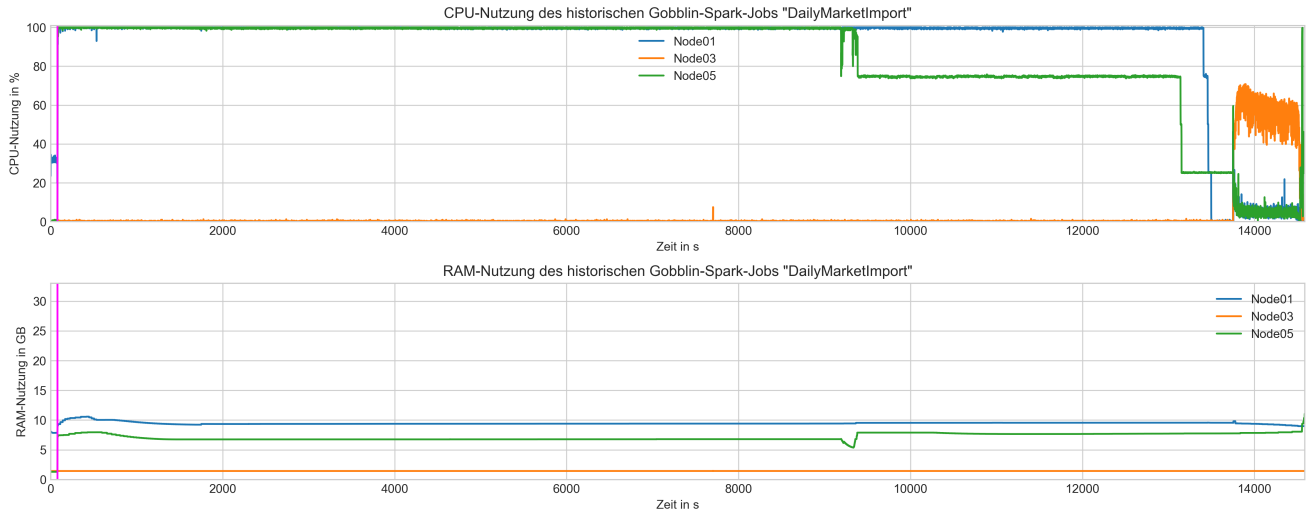

Abbildung A.3.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *DailyMarketImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

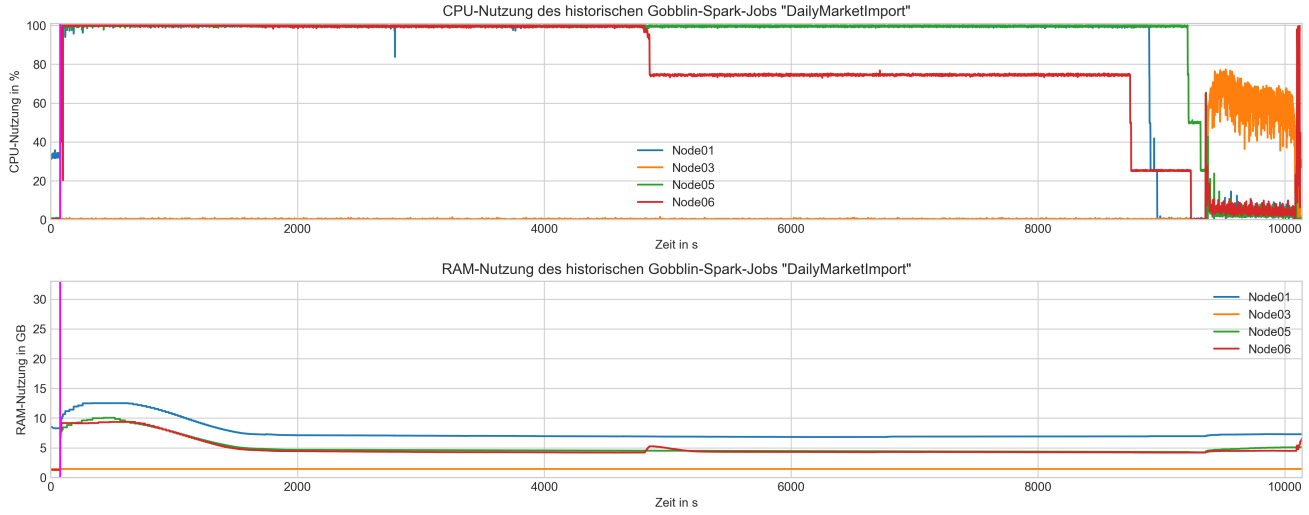

Abbildung A.4.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *DailyMarketImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

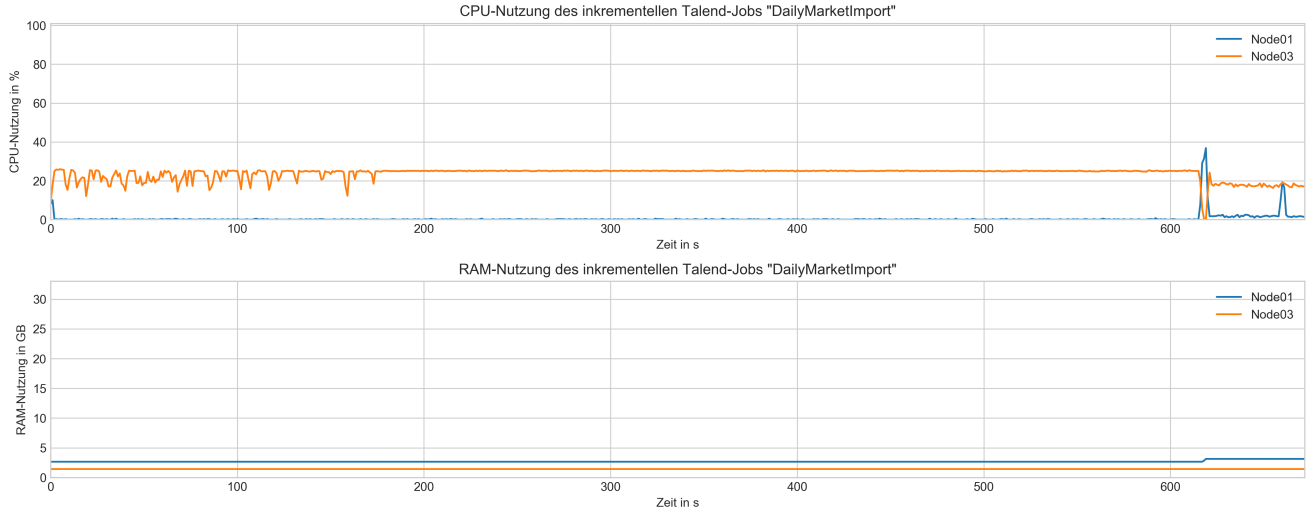

Abbildung A.5.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *DailyMarketImport* mit Talend-Framework im Datenset *nano* mit der Clustergröße 1

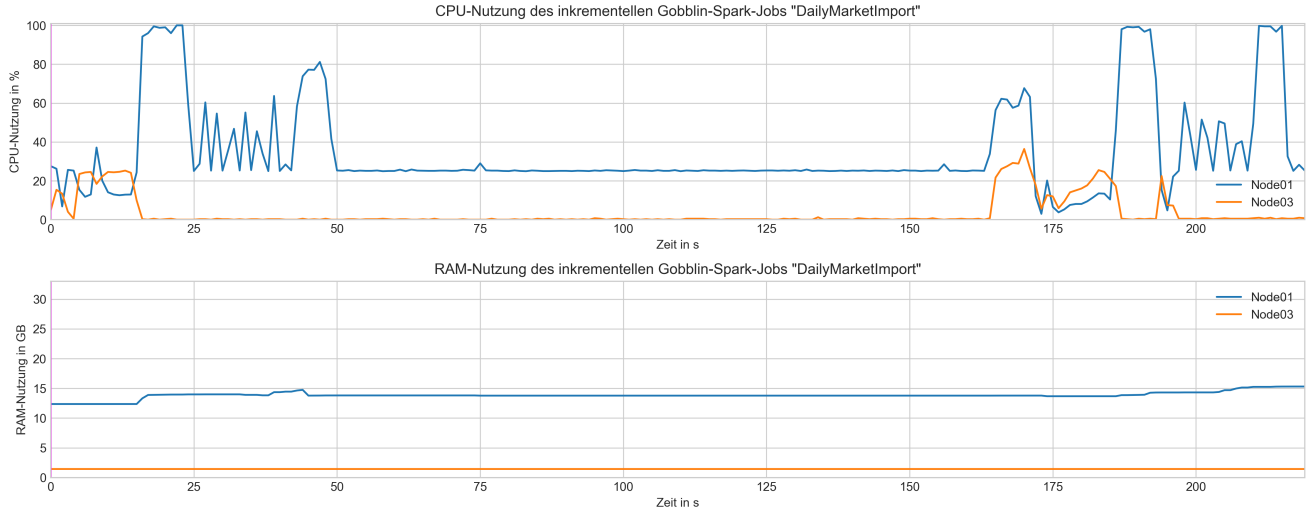

Abbildung A.6.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *DailyMarketImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

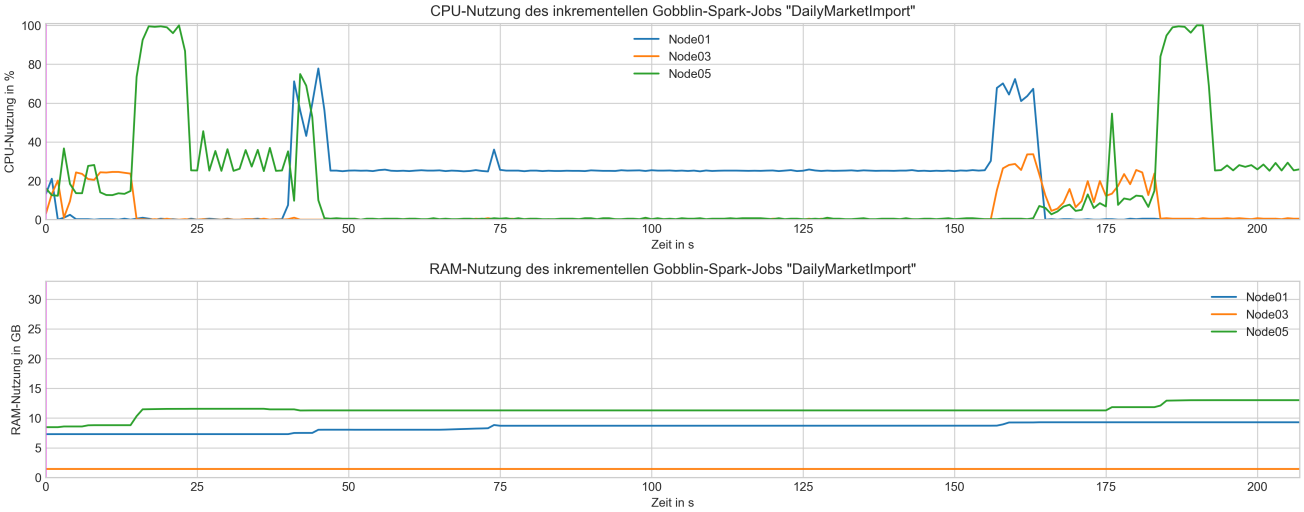

Abbildung A.7.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *DailyMarketImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

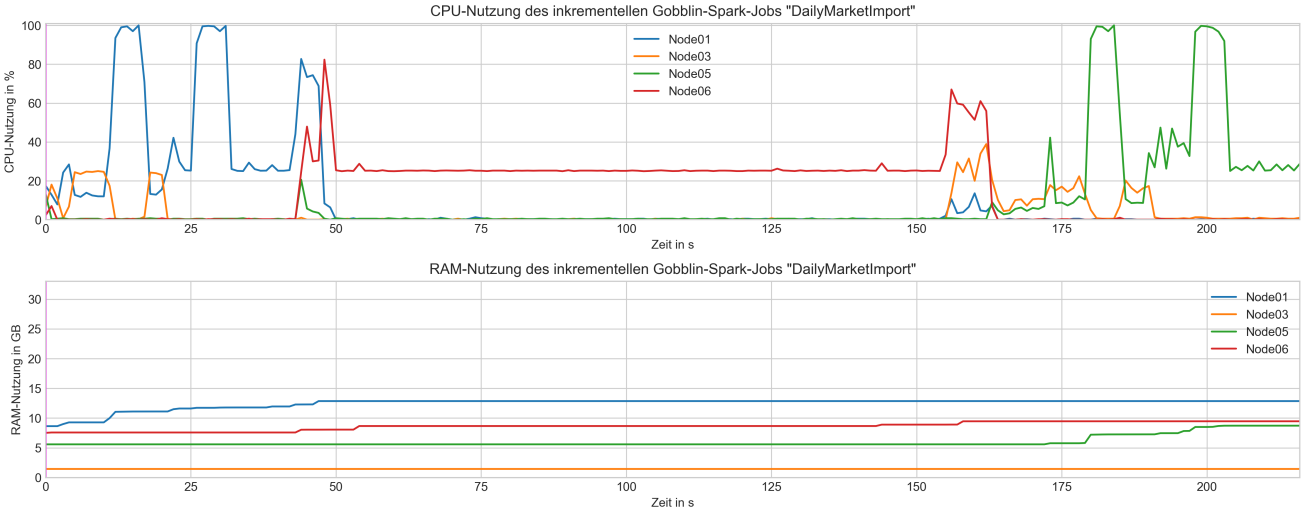

Abbildung A.8.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *DailyMarketImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

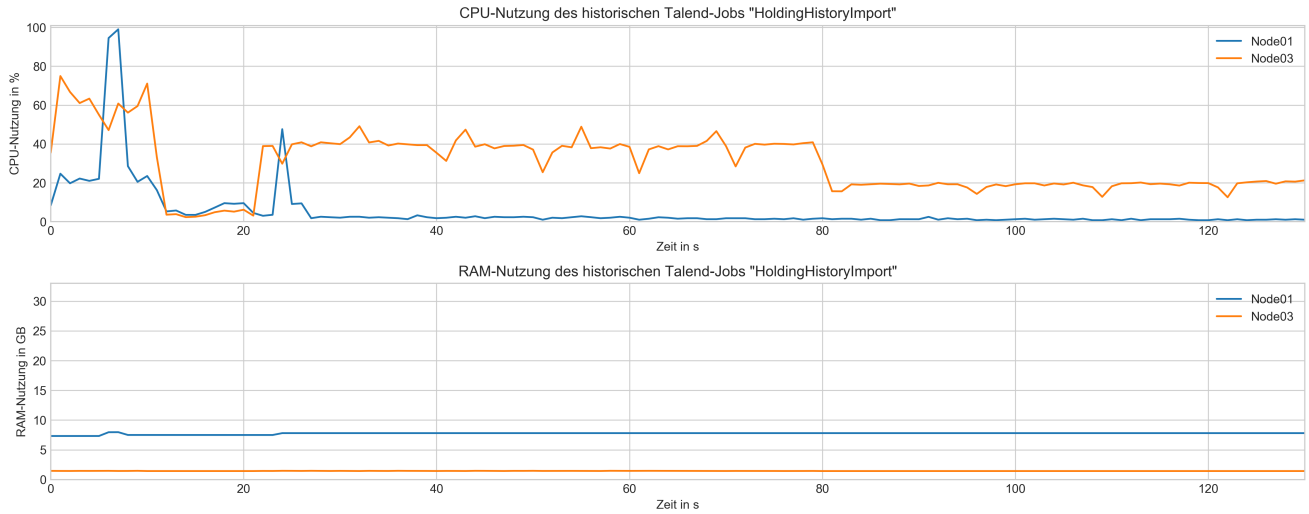

Abbildung A.9.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *HoldingHistoryImport* mit Talend-Framework im Datenset *nano* mit der Clustergröße 1

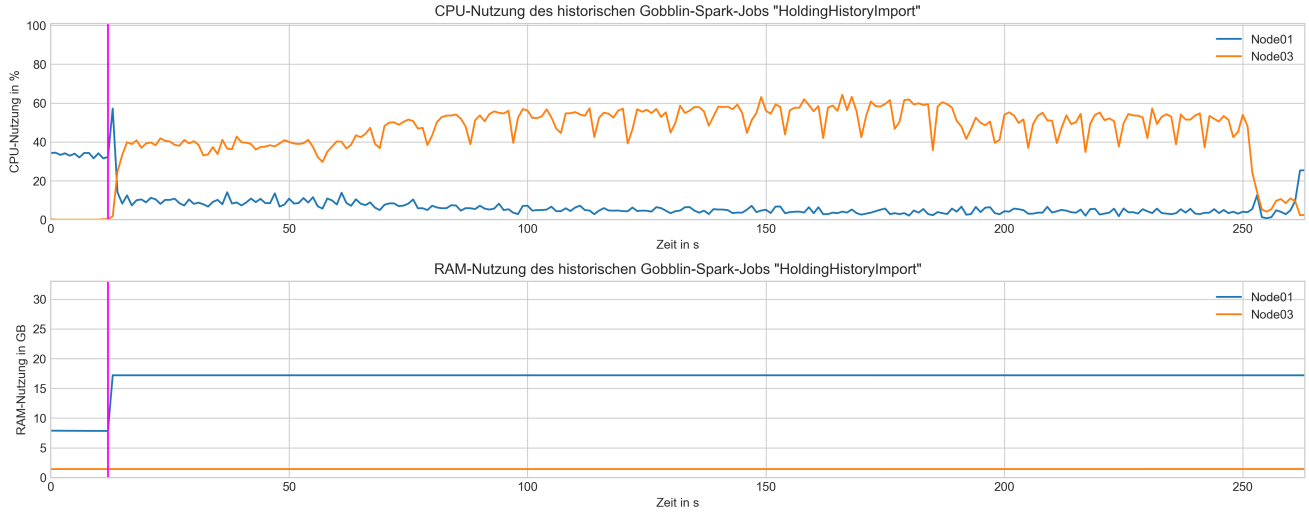

Abbildung A.10.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *HoldingHistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

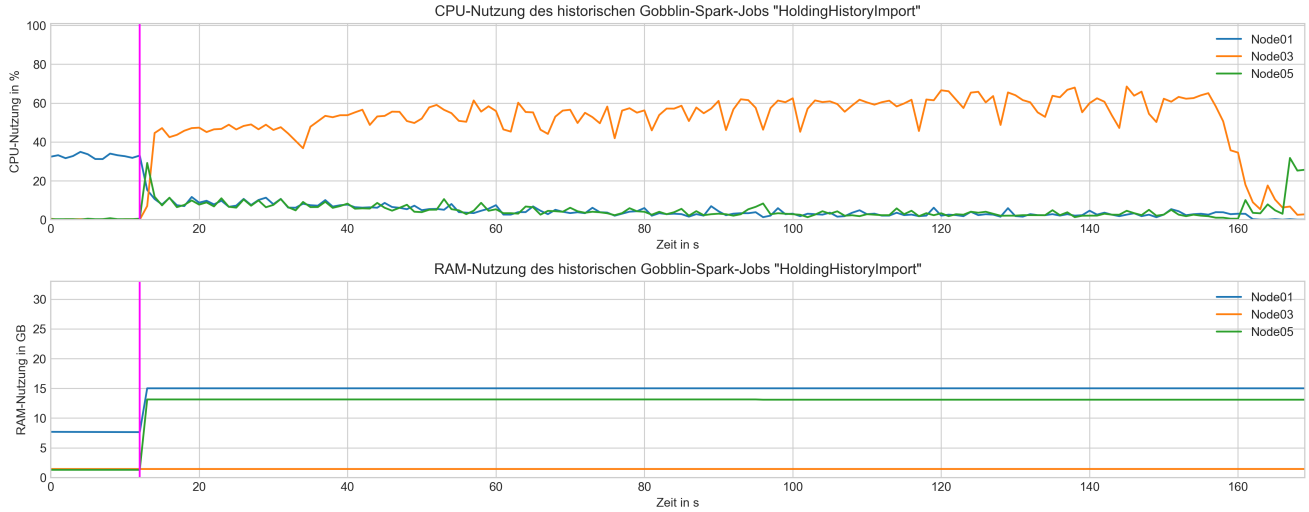

Abbildung A.11.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *Holding-HistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

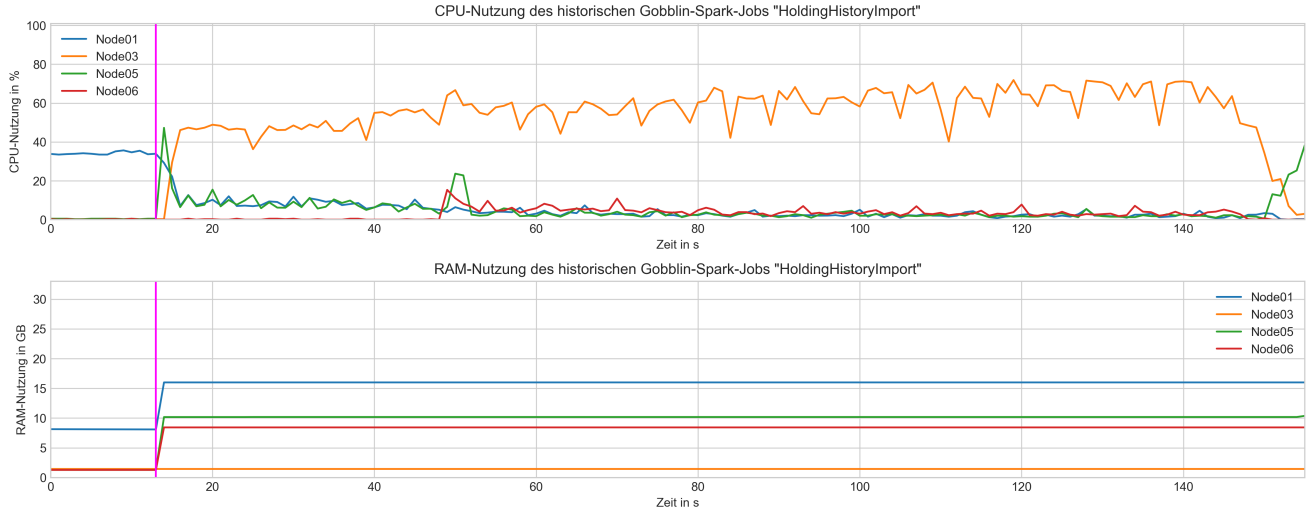

Abbildung A.12.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *Holding-HistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

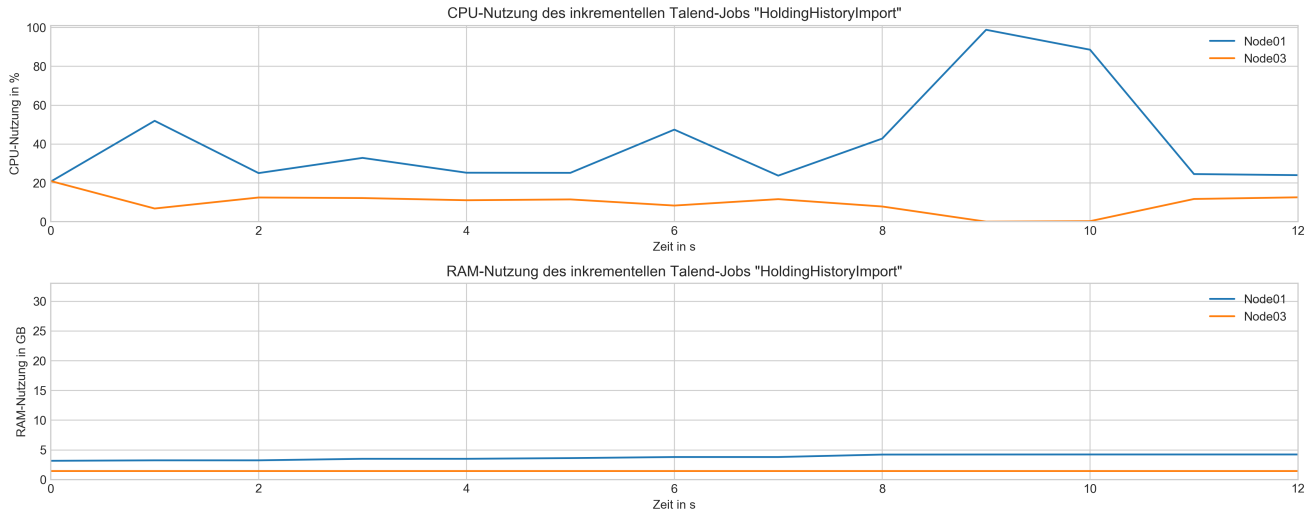

Abbildung A.13.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *Holding-HistoryImport* mit Talend-Framework im Datenset *nano* mit der Clustergröße 1

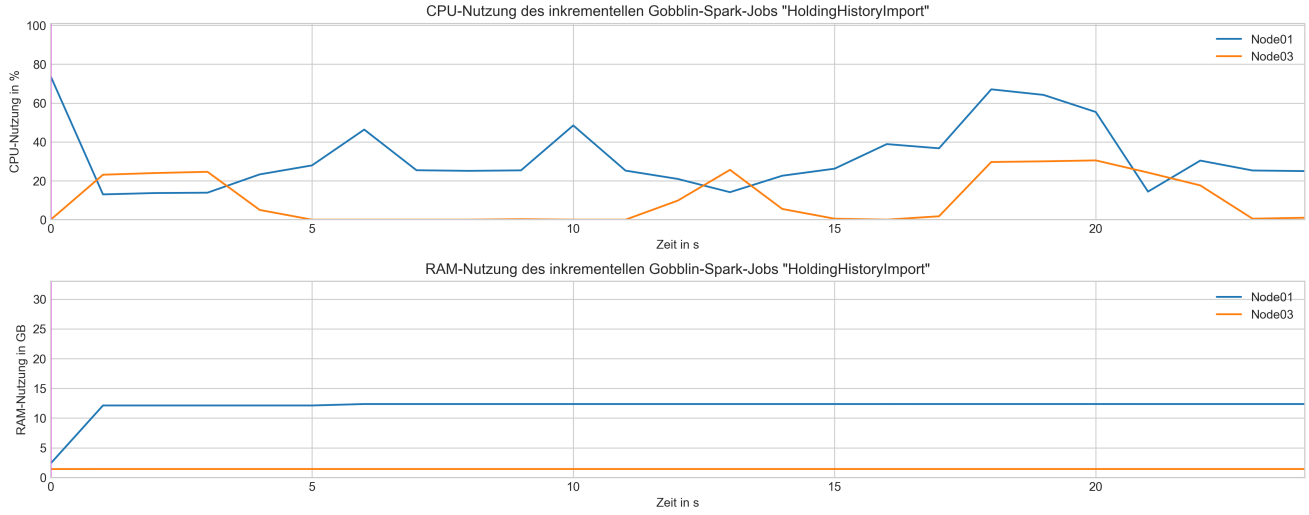

Abbildung A.14.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *Holding-HistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

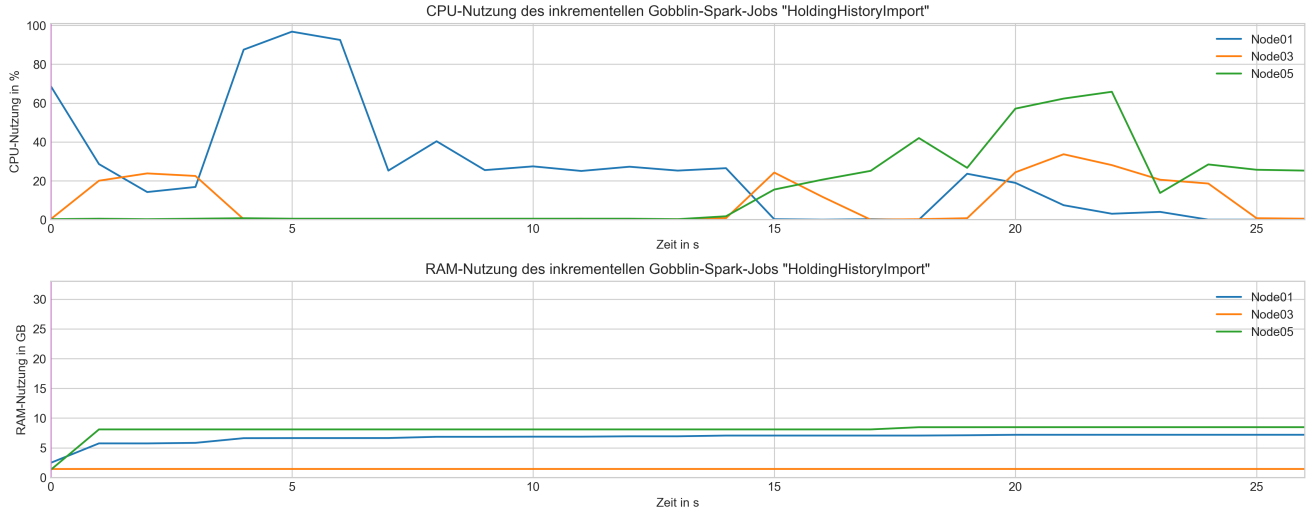

Abbildung A.15.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *Holding-HistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

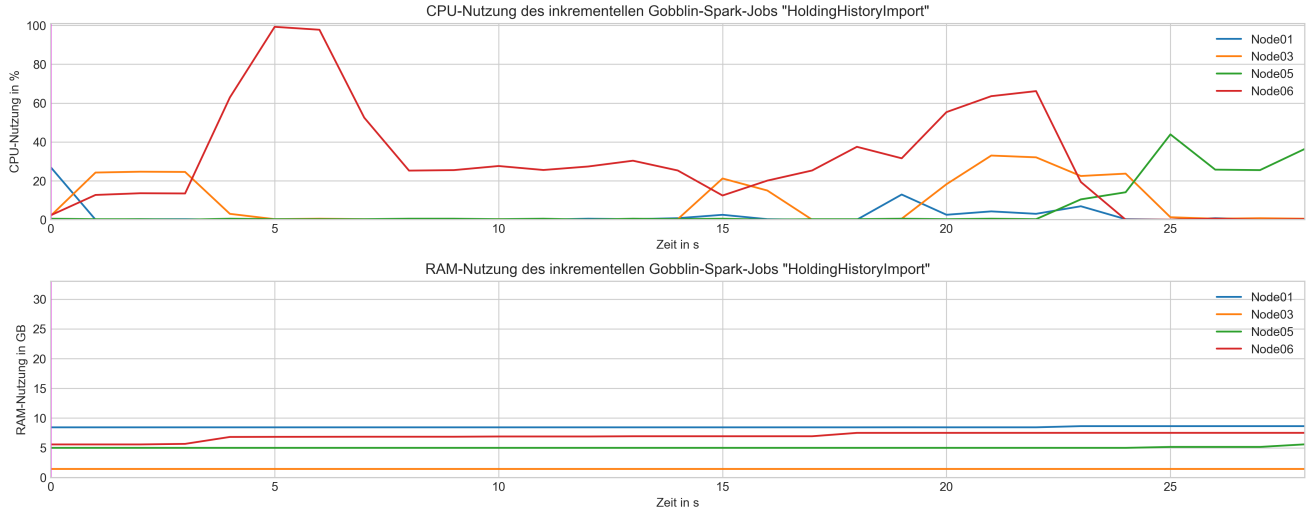

Abbildung A.16.: Hardware-Nutzung des inkrementellen Non-Streaming-Importjobs *Holding-HistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

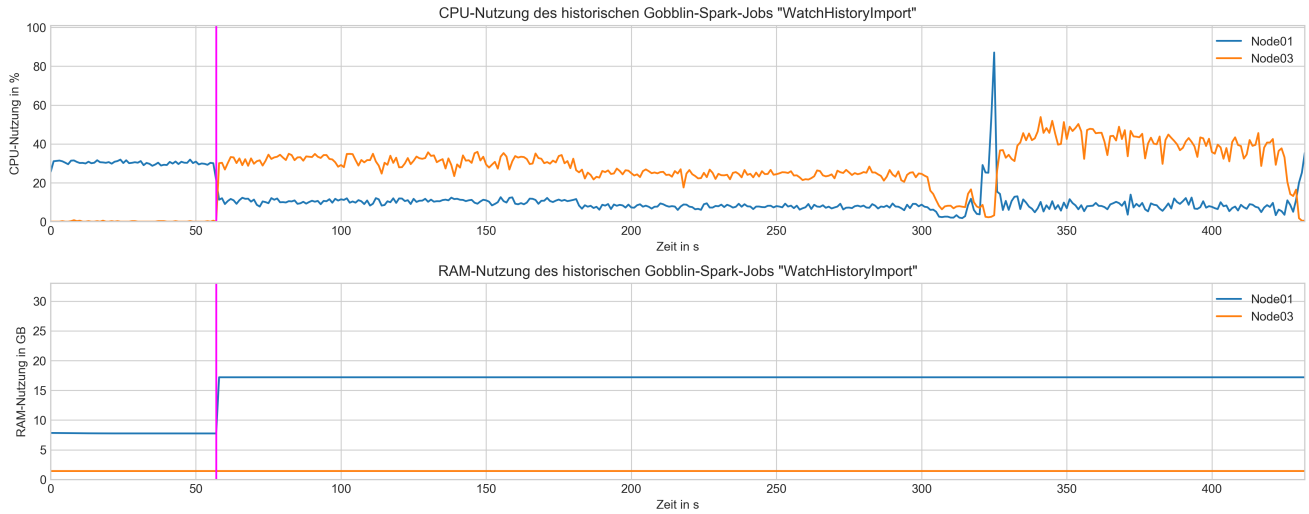

Abbildung A.17.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *WatchHistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 1

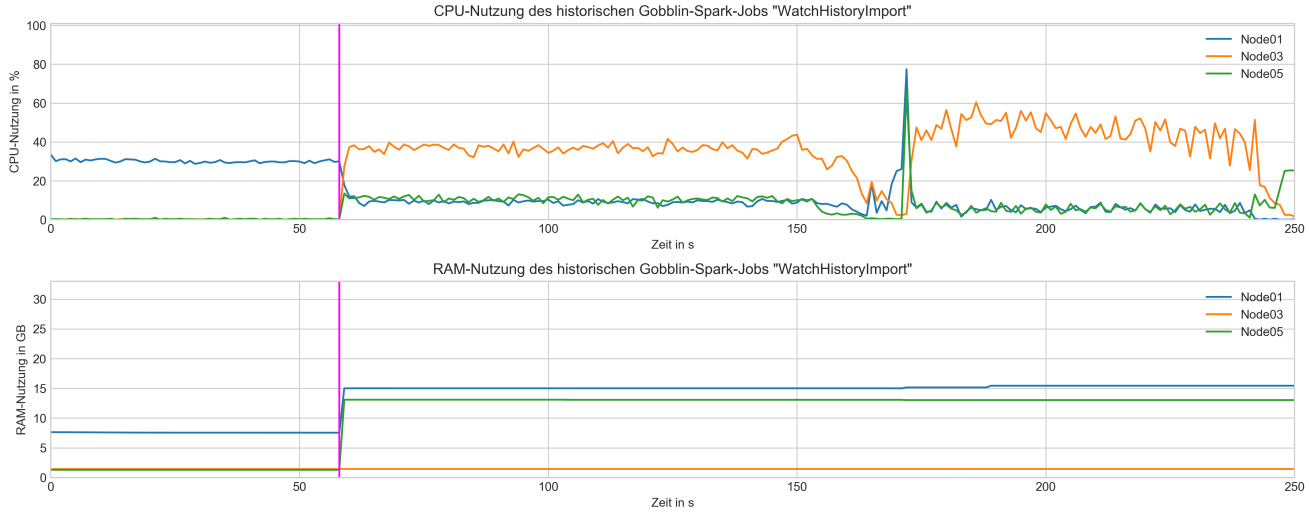

Abbildung A.18.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *WatchHistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 2

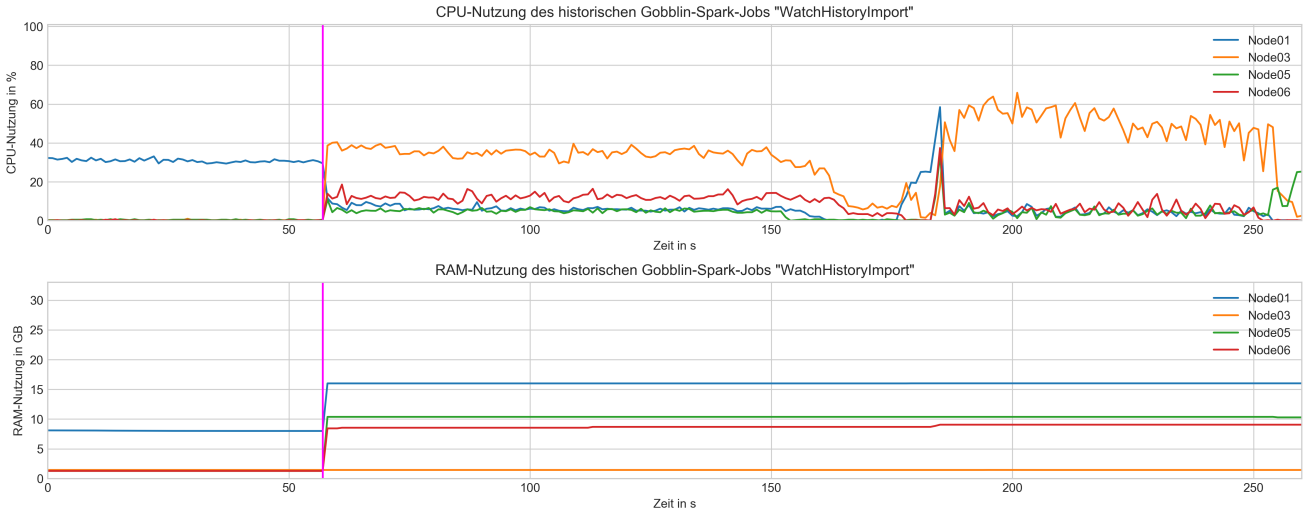

Abbildung A.19.: Hardware-Nutzung des historischen Non-Streaming-Importjobs *WatchHistoryImport* in der Gobblin-Spark-Kombination im Datenset *nano* mit der Clustergröße 3

Literaturverzeichnis

[Apache-Foundation 2018a] Apache-Foundation: *Apache Avro Project*. 2018. – URL https://avro.apache.org/. – accessed 14-03-2018

[Apache-Foundation 2018b] Apache-Foundation: *Apache Gobblin Documentation*. 2018. – URL https://gobblin.readthedocs.io/en/latest/. – accessed 16-03-2018

[Apache-Foundation 2018c] Apache-Foundation: *Apache Gobblin GitHub Repository*. 2018. – URL https://github.com/apache/incubator-gobblin/tree/master. – accessed 30-03-2018

[Apache-Foundation 2018d] Apache-Foundation: *Apache Gobblin Project*. 2018. – URL https://gobblin.apache.org/. – accessed 14-03-2018

[Apache-Foundation 2018e] Apache-Foundation: *Apache Hadoop Project*. 2018. – URL https://hadoop.apache.org/. – accessed 14-03-2018

[Apache-Foundation 2018f] Apache-Foundation: *Apache Incubator*. 2018. – URL https://incubator.apache.org/. – accessed 16-03-2018

[Apache-Foundation 2018g] Apache-Foundation: *Apache Kafka Project*. 2018. – URL https://kafka.apache.org/. – accessed 14-03-2018

[Apache-Foundation 2018h] Apache-Foundation: *Apache Projects*. 2018. – URL https://projects.apache.org/. – accessed 18-03-2018

[Apache-Foundation 2018i] Apache-Foundation: *Apache Spark Documentation*. 2018. – URL https://spark.apache.org/docs/latest/. – accessed 21-03-2018

[Apache-Foundation 2018j] Apache-Foundation: *Apache Spark Project*. 2018. – URL https://spark.apache.org/. – accessed 14-03-2018

[Armbrust u. a. 2015] Armbrust, Michael ; Xin, Reynold S. ; Lian, Cheng ; Huai, Yin ; Liu, Davies ; Bradley, Joseph K. ; Meng, Xiangrui ; Kaftan, Tomer ; Franklin, Michael J. ; Ghodsi, Ali ; Zaharia, Matei: Spark SQL: Relational Data Processing in Spark. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2015 (SIGMOD '15), S. 1383–1394. – URL http://doi.acm.org/10.1145/2723372.2742797. – ISBN 978-1-4503-2758-9

[Beyer u. a. 2017] Beyer, Mark ; Thoo, Eric ; Selvage, Mei Y. ; Zaidi, Ehtisham: *Magic Quadrant for Data Integration Tools*. 2017. – URL https://www.gartner.com/doc/3777464/magic-quadrant-data-integration-tools. – accessed 22-03-2018

[Bleuel 2016] Bleuel, Maurice: *Implementation and Evaluation of the TPC-DI Benchmark for Data Integration Systems*, Humboldt-Universität zu Berlin, Masterthesis, Jun 2016. – URL https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/teaching/studienDiplomArbeiten/finished/2016/expose_bleuel.pdf

[Chen u. a. 2013] Chen, Jinchuan ; Chen, Yueguo ; Du, Xiaoyong ; Li, Cuiping ; Lu, Jiaheng ; Zhao, Suyun ; Zhou, Xuan: Big data challenge: a data management perspective. In: *Frontiers of Computer Science* 7 (2013), Apr, Nr. 2, S. 157–164. – URL https://doi.org/10.1007/s11704-013-3903-7. – ISSN 2095-2236

[Dayal u. a. 2009] Dayal, Umeshwar ; Castellanos, Malu ; Simitsis, Alkis ; Wilkinson, Kevin: Data Integration Flows for Business Intelligence. In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. New York, NY, USA : ACM, 2009 (EDBT '09), S. 1–11. – URL http://doi.acm.org/10.1145/1516360.1516362. – ISBN 978-1-60558-422-5

[Dong und Srivastava 2013] Dong, X. L. ; Srivastava, D.: Big data integration. In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, April 2013, S. 1245–1248. – ISSN 1063-6382

[Geerdink 2016] Geerdink, Bas: *Get Rid of traditional ETL, Move to Spark!* 2016. – URL https://www.slideshare.net/SparkSummit/spark-summit-eu-talk-by-bas-geerdink-68139317. – accessed 22-03-2018

[HortonWorks 2013] HortonWorks: *Apache Hadoop 2 is GA.* 2013. – URL https://hortonworks.com/blog/apache-hadoop-2-is-ga/. – accessed 18-03-2018

[IBM 2013] IBM: *Annual Report.* 2013. – URL https://www.ibm.com/annualreport/2013/bin/assets/2013_ibm_annual.pdf. – accessed 28-03-2018

[InfluxData 2018] InfluxData: *Telegraf Project.* 2018. – URL https://www.influxdata.com/time-series-platform/telegraf/. – accessed 14-03-2018

[Informatica 2017] Informatica: *Magic Quadrant for Data Integration Tools.* 2017. – URL https://www.informatica.com/de/data-integration-magic-quadrant.html. – accessed 22-03-2018

[Kimball und Ross 2011] Kimball, Ralph ; Ross, Margy: *The data warehouse toolkit: the complete guide to dimensional modeling.* John Wiley & Sons, 2011

[Lenzerini 2002] Lenzerini, Maurizio: Data Integration: A Theoretical Perspective. In: *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems.* New York, NY, USA : ACM, 2002 (PODS '02), S. 233–246. – URL http://doi.acm.org/10.1145/543613.543644. – ISBN 1-58113-507-6

[Majchrzak u. a. 2011] Majchrzak, Tim A. ; Jansen, Tobias ; Kuchen, Herbert: Efficiency Evaluation of Open Source ETL Tools. In: *Proceedings of the 2011 ACM Symposium on Applied Computing.* New York, NY, USA : ACM, 2011 (SAC '11), S. 287–294. – URL http://doi.acm.org/10.1145/1982185.1982251. – ISBN 978-1-4503-0113-8

[Marz und Warren 2015] Marz, Nathan ; Warren, James: *Big Data: Principles and Best Practices of Scalable Realtime Data Systems.* 1st. Greenwich, CT, USA : Manning Publications Co., 2015. – ISBN 1617290343, 9781617290343

[Ohlden 2008] Ohlden, Anna: *Syncsort And Vertica Shatter Database ETL World Record Using HP Bladesystem C-Class.* 2008. – URL http://www.science20.com/newswire/syncsort_and_vertica_shatter_database_etl_world_record_using_hp_bladesystem_cclass. – accessed 22-03-2018

[Oracle 2018] Oracle: *MySQL Project.* 2018. – URL https://www.mysql.com/. – accessed 14-03-2018

[Poess u. a. 2014] Poess, Meikel ; Rabl, Tilmann ; Jacobsen, Hans-Arno ; Caufield, Brian: TPC-DI: the first industry benchmark for data integration. In: *Proceedings of the VLDB Endowment* 7 (2014), Nr. 13, S. 1367–1378

[Qiao u. a. 2015] Qiao, Lin ; Li, Yinan ; Takiar, Sahil ; Liu, Ziyang ; Veeramreddy, Narasimha ; Tu, Min ; Dai, Ying ; Buenrostro, Issac ; Surlaker, Kapil ; Das, Shirshanka ; Botev, Chavdar: Gobblin: Unifying Data Ingestion for Hadoop. In: *Proc. VLDB Endow.* 8 (2015), August, Nr. 12, S. 1764–1769. – URL http://dx.doi.org/10.14778/2824032.2824073. – ISSN 2150-8097

[Qualitz 2013] Qualitz, Sascha: *Vergleich von Open-Source und kommerziellen Programmen zur Durchführung eines ETL-Prozesses*, Humboldt-Universität zu Berlin, Masterthesis, Nov 2013. – URL https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/teaching/studienDiplomArbeiten/finished/2013/qualitz_expose_130420.pdf

[Rabl u. a. 2011] Rabl, Tilmann ; Frank, Michael ; Sergieh, Hatem M. ; Kosch, Harald: A Data Generator for Cloud-Scale Benchmarking. In: Nambiar, Raghunath (Hrsg.) ; Poess, Meikel (Hrsg.): *Performance Evaluation, Measurement and Characterization of Complex Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 41–56. – ISBN 978-3-642-18206-8

[Sagiroglu und Sinanc 2013] Sagiroglu, S. ; Sinanc, D.: Big data: A review. In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*, May 2013, S. 42–47

[Simitsis u. a. 2009] Simitsis, Alkis ; Vassiliadis, Panos ; Dayal, Umeshwar ; Karagiannis, Anastasios ; Tziovara, Vasiliki: Benchmarking ETL Workflows. In: Nambiar, Raghunath (Hrsg.) ; Poess, Meikel (Hrsg.): *Performance Evaluation and Benchmarking*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, S. 199–220. – ISBN 978-3-642-10424-4

[Talend 2018a] Talend: *Talend Documentation*. 2018. – URL https://help.talend.com/. – accessed 18-03-2018

[Talend 2018b] Talend: *Talend Open Studio for Big Data*. 2018. – URL https://www.talend.com/products/big-data/big-data-open-studio/. – accessed 14-03-2018

[Talend 2018c] Talend: *Talend Resources*. 2018. – URL https://www.talend.com/resources/l-big-data-integration/. – accessed 21-03-2018

[Transaction Processing Performance Council 2014] Transaction Processing Performance Council: *TPC-DI*. 2014. – URL http://www.tpc.org/tpcdi/. – Version 1.1.0, accessed 08-03-2018

[Transaction Processing Performance Council 2018a] Transaction Processing Performance Council: *About the TPC*. 2018. – URL http://www.tpc.org/information/about/abouttpc.asp. – accessed 12-04-2018

[Transaction Processing Performance Council 2018b] Transaction Processing Performance Council: *TPC-Pricing*. 2018. – URL http://www.tpc.org/pricing/. – Version 2.3.0, accessed 08-03-2018

[Vassiliadis und Simitsis 2009] Vassiliadis, Panos ; Simitsis, Alkis: *Near Real Time ETL*. S. 1–31. In: Kozielski, Stanislaw (Hrsg.) ; Wrembel, Robert (Hrsg.): *New Trends in Data Warehousing and Data Analysis*. Boston, MA : Springer US, 2009. – URL https://doi.org/10.1007/978-0-387-87431-9_2. – ISBN 978-0-387-87431-9

[Wyatt 2008] Wyatt, Len: *ETL World Record!* 2008. – URL https://blogs.msdn.microsoft.com/sqlperf/2008/02/27/etl-world-record/. – accessed 22-03-2018

[Wyatt u. a. 2009] Wyatt, Len ; Caufield, Brian ; Pol, Daniel: Principles for an ETL Benchmark. In: Nambiar, Raghunath (Hrsg.) ; Poess, Meikel (Hrsg.): *Performance Evaluation and Benchmarking*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, S. 183–198. – ISBN 978-3-642-10424-4

[Zaharia u. a. 2010] Zaharia, Matei ; Chowdhury, Mosharaf ; Franklin, Michael J. ; Shenker, Scott ; Stoica, Ion: Spark: Cluster computing with working sets. In: *HotCloud* 10 (2010), Nr. 10-10, S. 95

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 25.04.2018 Birger Kamp