Systemübergreifende Kostennormalisierung für Integrationsprozesse

Matthias Böhm[1], Dirk Habich[2], Wolfgang Lehner[2], und Uwe Wloka[1]

[1] Hochschule für Technik und Wirtschaft Dresden (FH), Fachgebiet Datenbanksysteme
mboehm@informatik.htw-dresden.de
wloka@informatik.htw-dresden.de
[2] Technische Universität Dresden, Lehrstuhl Datenbanken
dirk.habich@tu-dresden.de
wolfgang.lehner@tu-dresden.de

Zusammenfassung Auf Grundlage der Vielzahl proprietärer Integrationssysteme ist zunehmend die Entwicklung von Ansätzen zur modellbasierten Generierung von Integrationsprozessen zu beobachten. Eine derartige Generierung bietet weiterhin die Möglichkeit der Auswahl des optimalen Integrationssystems, welche ein hohes Optimierungspotenzial in sich birgt. Die Grundlage für eine solche Entscheidung ist jedoch eine integrationssystemübergreifende Kostennormalisierung, um die Vergleichbarkeit von Verarbeitungsstatistiken zu ermöglichen. Basierend auf einem plattformunabhängigen Kostenmodell und der systemübergreifenden Kostennormalisierung kann eine kostenbasierte Optimalitätsentscheidung hinsichtlich des effizientesten Integrationssystems getroffen werden. Folglich können hierbei veränderliche Workload-Charakteristika in die Betrachtung einbezogen werden. In diesem Papier zeigen wir zunächst die Probleme der systemübergreifenden Kostenmodellierung und Kostennormalisierung auf, welche bislang noch nicht betrachtet wurden. Darauf aufbauend diskutieren wir die plattforminvariante Kostenmodellierung und die Erhebung von Verarbeitungsstatistiken. Weiterhin führen wir drei Algorithmen zur Kostennormalisierung ein, mit denen die Vergleichbarkeit von Verarbeitungsstatistiken sichergestellt wird. Als Konsequenz kann das vorgestellte Konzept in beliebigen Ansätzen der Generierung von Integrationsprozessen zum Einsatz kommen.

1 Einleitung

Integrationsprozesse – im Sinne von workflowbasierten Integrationsaufgaben – gewinnen zunehmend an Bedeutung, da ein kontinuierlicher Wandel des Datenmanagements von der Verwaltung zentral vorgehaltener Daten hin zur Verwaltung von hochgradig verteilten und heterogenen Datenbeständen und Anwendungen erfolgt. In diesem Kontext ist die Effizienz kompletter IT-Infrastrukturen von den eingesetzten Integrationsplattformen abhängig. Beispiele für derartige Plattformen sind EAI-Server (Enterprise Application Integration), ETL-Werkzeuge (Extraction Transformation Loading), föderierte DBMS, WfMS (Workflow Management Systems) und WSMS (Web Service Management Systems). Um die Probleme des hohen Entwicklungsaufwands und der geringen Portabilität zu reduzieren, wurden Ansätze [BWHL08,DHW+08,AN08] zur modellbasierten Generierung von Integrationsprozessen entwickelt. Derartige Generierungsansätze bieten die Möglichkeit der Auswahl des optimalen Integrationssystems.

Um sich verändernde Workload-Charakteristika in ausreichendem Maße einzubeziehen, sollte diese Optimalitätsentscheidung kostenbasiert (auf Basis von Verarbeitungsstatistiken) erfolgen. Dies eröffnet jedoch das Problem des Kostenvergleichs heterogener Integrationsplattformen, im Sinne der plattformunabhängigen Kostenmodellierung und der entsprechenden Kostennormalisierung konkreter Statistiken in das plattformunabhängige Modell. Mit dem Ziel der Herstellung einer systemübergreifenden Vergleichbarkeit sind die Aspekte Parallelität, Ressourcenverwendung, unterschiedliche Hardware und Verarbeitungsmodelle, Semantik der erhobenen Statistiken sowie fehlende und inkonsistente Statistiken im Rahmen der Normalisierung zu betrachten.

Um das Problem der systemübergreifenden Kostennormalisierung und einen möglichen Lösungsansatz aufzuzeigen, leisten wir in diesem Papier folgende Beiträge:

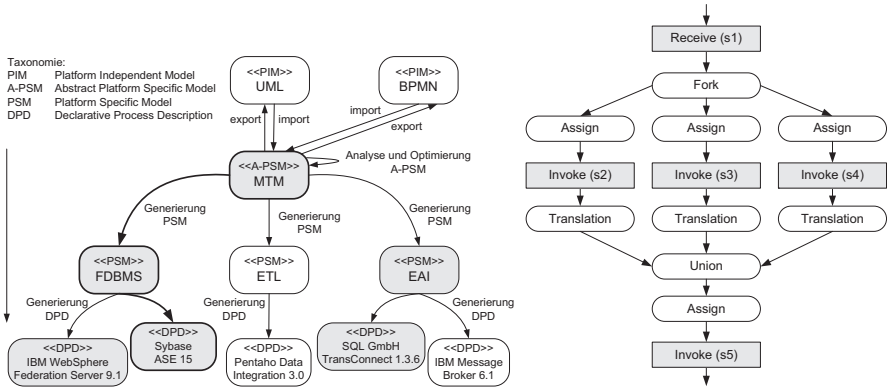
- In Abschnitt 2 erläutern wir die wesentliche Problemstellung der systemübergreifenden Kostennormalisierung für heterogene Integrationssysteme. Dabei präsentieren wir auch unsere generellen Annahmen zur Auswahlmöglichkeit von Integrationssystemen und beschreiben unsere Vision des *Invisible Deployments*.
- Aufbauend auf der Problembeschreibung diskutieren wir die plattformunabhängige Modellbildung in Abschnitt 3. Hier stellen wir sowohl ein plattforminvariantes Kostenmodell als auch Anforderungen an die Erhebung von Statistiken dar.
- In Abschnitt 4 führen wir dann drei Algorithmen zur Kostennormalisierung ein, welche plattformspezifische Verarbeitungsstatistiken in das plattforminvariante Kostenmodell überführen und damit die Vergleichbarkeit herstellen.
- Danach präsentieren wir die experimentelle Evaluierung in Abschnitt 5.

Abschließend zeigen wir mögliche Anwendungsgebiete im Rahmen der Analyse verwandter Arbeiten in Abschnitt 6 und fassen das Papier in Abschnitt 7 zusammen.

2 Vision und Problembeschreibung

In diesem Abschnitt sollen die Problematik der plattformübergreifenden Kostenmodellierung und die entsprechende Kostennormalisierung näher diskutiert werden. Deshalb wollen wir zunächst kurz auf unser GCIP Framework (Generation of Complex Integration Processes) [BWHL08] (als ein repräsentativer Ansatz für die modellgetriebene Generierung von Integrationsprozessen) eingehen. Das Grundkonzept der Generierung bei diesem Ansatz ist in Abbildung 1(a) dargestellt. In diesem Rahmen können Integrationsprozesse auf plattformunabhängiger Ebene (PIM[3]) mit UML-Aktivitätsdiagrammen oder BPMN-Prozessbeschreibungen modelliert werden. Diese Modelle werden anschließend in eine zentrale Repräsentation (das sogenannte Message Transformation Model (MTM) [BBW+07]) importiert. Ausgehend von dieser Repräsentation erlaubt das Framework die Generierung von Integrationsaufgaben für unterschiedliche Integrationssystemtypen (PSM[4]). Momentan werden die Typen FDBMS, ETL und EAI unterstützt. Ausgehend von den plattformspezifischen Repräsentationen werden dann Prozessbeschreibungen für unterschiedliche konkrete Integrationssysteme generiert. Das folgende Beispiel veranschaulicht einen typischen Integrationsprozess.

[3] Platform-Independent Model (PIM): Eine plattformunabhängige Prozessbeschreibung
[4] Platform-Specific Model (PSM): Eine systemtypspezifische Prozessbeschreibung



(a) Generierungskonzept (b) Beispiel Integrationsprozess

Abbildung 1: Grundkonzept GCIP Framework

Beispiel 1 Integrationsprozess: *Abbildung 1(b) zeigt ein Beispiel für die zentrale Repräsentation (MTM). Dieser Integrationsprozess beschreibt den Empfang einer Nachricht von System $s1$ sowie drei nebenläufige Teilprozesse, in denen die externen Systeme $s2$, $s3$ und $s4$ angefragt werden und ein Schema-Mapping der Resultate in ein einheitliches Schema vorgenommen wird. Abschließend erfolgt die Vereinigung der Ergebnismengen und die Übertragung an ein System $s5$. Dieser Integrationsprozess kann auf eine Reihe von Integrationssystemtypen abgebildet werden. Beispielsweise kann ein Trigger für FDBMS oder ein WSBPEL-Prozess für EAI-Server generiert werden.*

2.1 Grundlegende Annahmen

Die Schlussfolgerungen der Generierung von Integrationsprozessen lassen sich zu zwei wesentlichen Annahmen konkretisieren:

Annahme 1 Generierbarkeit: *Integrationsprozesse können plattformunabhängig modelliert werden. Darauf aufbauend ist es möglich, proprietäre Beschreibungen von Integrationsprozessen für konkrete Integrationssysteme zu generieren.*

Annahme 2 Auswahlmöglichkeit: *In einer typischen IT-Infrastruktur existieren mehrere Integrationssysteme mit überlappenden Funktionalitäten. Folglich besteht die Möglichkeit der Auswahl des optimalen Integrationssystems.*

Aufbauend auf der Annahme der Generierbarkeit von Prozessbeschreibungen konkreter Integrationssysteme (durch verwandte Arbeiten [BWHL08,DHW+08,AN08] getestet und nachgewiesen), ist folglich die Voraussetzung einer Wahlmöglichkeit zwischen alternativen Integrationssystemen gegeben. Weiterhin ist auch die Praxisrelevanz dieser Wahlmöglichkeit dadurch belegt, dass typische IT-Infrastrukturen eine Menge an unterschiedlichen Integrationssystemen mit überlappenden Funktionalitäten (wie beispielsweise spezielle Operatoren, unterstützte externe Systeme, Möglichkeiten auf externe Ereignisse zu reagieren oder transaktionale Eigenschaften) beinhalten [Sto02]. Folglich kann ein gegebener Integrationsprozess mit Hilfe unterschiedlicher Integrationssysteme realisiert werden, ohne das externe Verhalten zu beeinflussen. Somit kann die Auswahlentscheidung

auf Grundlage von nichtfunktionalen Eigenschaften, wie beispielsweise Effizienz, Skalierbarkeit und Ressourcenverbrauch – und damit kostenbasiert – getroffen werden.

2.2 Vision des *Invisible Deployment*

Auf den beschriebenen Annahmen beruht unsere langfristige Vision des *Invisible Deployment* von Integrationsprozessen. Die Kernidee dieser Vision ist es die Ebene der Integrationssysteme und -technologien systematisch zu abstrahieren und letztendlich transparent zu gestalten (zu virtualisieren). Neben Konfigurationsmanagement und einigen anderen Herausforderungen, besteht hier die Notwendigkeit der Auswahl des optimalen Integrationssystems. Diese Auswahl soll in einem zweistufigen Verfahren erfolgen. In einem ersten Schritt (*Schritt 1: Auswahl Kandidatenmenge*) ist zunächst eine Kandidatenmenge von Integrationssystemen zu bestimmen, welche aus funktionaler Sicht zur Realisierung eines gegebenen Integrationsprozesses eingesetzt werden können. Hierzu sind die funktionalen Anforderungen aus dem Integrationsprozess abzuleiten und mit den Merkmalsvektoren [BF07] existierender Integrationssysteme zu vergleichen. Auf Grundlage der Kandidatenmenge kann in einem zweiten Schritt (*Schritt 2: Auswahl Optimales Integrationssystem*) eine kostenbasierte Optimalitätsentscheidung – ausschließlich basierend auf nichtfunktionalen Eigenschaften – getroffen werden. Einen Teilschritt zum Erreichen dieser Vision könnte ein *Advisor* zur Unterstützung von Auswahlentscheidungen darstellen. Dabei werden normalisierte Kosten und das definierte plattforminvariante Kostenmodell dazu verwendet dem Nutzer eine *What-If*-Schnittstelle – im Sinne von Plan- und Kostendiagrammen [RH05] der unterschiedlichen Systeme für veränderliche Workload-Charakteristika – bereit zu stellen.

2.3 Problemstellungen der Kostenmodellierung und Kostennormalisierung

Zur Realisierung des Invisible Deployment ist die kostenbasierte Auswahl von Integrationssystemen eine zwingende Voraussetzung. Derartige kostenbasierte Entscheidungen erfordern im Wesentlichen zwei weitere Grundvoraussetzungen. Dies ist zum einen ein plattforminvariantes Kostenmodell, welches für Entscheidungen über Integrationssysteme hinweg zum Einsatz kommen kann. Zum anderen ist die Kostennormalisierung (Kostentransformation und Kostenvervollständigung) von eminenter Bedeutung. Im Rahmen dieser Normalisierung sind folglich Verarbeitungsstatistiken der einzelnen Systeme zu erheben und entsprechend zu transformieren und zu vervollständigen.

Als Konsequenz stellen also die Kostenmodellierung und Kostennormalisierung ein grundlegendes Problem in diesem Kontext dar. Dabei ist das Kostenmodell ausschließlich im Rahmen von Integrationsprozessen einsetzbar, während die Problematik der systemübergreifenden Kostennormalisierung auch in anderen Domänen Anwendung finden kann. Da im Hinblick auf diese Grundvoraussetzungen jedoch keinerlei Ergebnisse existieren, ist hier grundlegende Forschungsarbeit zu leisten. Gerade bei der Kostennormalisierung offenbaren sich hier folgende, grundlegende Problemstellungen:

- *Parallelität:* Im Rahmen der Normalisierung ist die Parallelität sowohl von Prozessinstanzen als auch Operatorinstanzen zu beachten. Darin ist die generelle Arbeitslast der gesamten Plattform einzubeziehen.

- *Ressourcenverwendung:* Neben der Parallelität ist ebenfalls die Ressourcenverwendung der einzelnen Instanzen und Operatoren zu analysieren. Hierbei ist die effektive Ressourcenverwendung $R_e(o_i)$ als auch die maximal sinnvolle Ressourcenverwendung $R_o(o_i)$ eines Operators o_i zu beachten.
- *Unterschiedliche Hardware:* Ein wesentliches Problem ist die unterschiedliche Hardware, auf der die jeweiligen Integrationssysteme ausgeführt werden. Im Rahmen der Normalisierung muss dieser Unterschied ausgeglichen werden.
- *Unterschiedliche Verarbeitungsmodelle:* Die Spezifika des jeweiligen Verarbeitungsmodells müssen in die Normalisierung einbezogen werden. Beispiele hierfür stellen die synchrone und asynchrone Verarbeitung als auch die Unterscheidung in das instanzbasierte und das vektorisierte (pipes and filter) Verarbeitungsmodell dar.
- *Semantik erhobener Statistiken:* Die Semantik der erhobenen Statistiken muss vereinheitlicht werden. Dies bedeutet, dass die Verarbeitungszeit die gleichen Aufgaben (Operatoren, Kardinalitäten) umfasst.
- *Fehlende Statistiken:* auf Grund der unterschiedlichen Verarbeitungsmodelle und primären Anwendungsgebiete der einzelnen Integrationssysteme müssen fehlende Statistiken (Statistiken, welche nicht erhoben werden können) auf Grundlage anderer Statistiken interpoliert oder geschätzt werden.
- *Inkonsistente Statistiken:* Auf Grund der Erhebung von sehr feingranularen Statistiken kann die Inkonsistenz zwischen Statistiken nicht ausgeschlossen werden und ist folglich im Rahmen der Normalisierung zu beseitigen.
- *Statistischer Fehler:* Im Rahmen von mehrmaligen Schätzungen von systemübergreifenden Kosten sollte der statistische Fehler (Differenz zwischen Erwartungswert und erhobenen Statistiken) reduziert und adaptiv minimiert werden.

Die aufgezeigten Probleme werden (mit Einschränkungen bei unterschiedlichen Verarbeitungsmodellen und der Minimierung des statistischen Fehlers) im Rahmen dieses Papiers mit einem plattforminvarianten Kostenmodell und geeigneten Algorithmen zur Kostennormalisierung adressiert.

Annahme 3 *Der im Rahmen dieser Arbeit vorgestellte Ansatz zur Kostennormalisierung ermöglicht die Vergleichbarkeit der Kosten (im Sinne der Verarbeitungseffizienz) von Integrationsprozessen über Integrationssysteme und Plattformen hinweg.*

An dieser Stelle sei weiterhin auf das Hauptproblem bei der Kostennormalisierung verwiesen. So ist die Kostennormalisierung per Definition unidirektional. Dies hat Auswirkungen auf unsere Vision des *Invisible Deployment*. Es kann ausschließlich eine Normalisierung der Verarbeitungsstatistiken konkreter Integrationssysteme in eine plattforminvariante (damit normalisierte) Form erfolgen, da eine denormalisierte Repräsentation auf exakt eine normalisierte Form abgebildet wird. Hingegen ist eine Denormalisierung abstrakter Kosten in Verarbeitungsstatistiken spezifischer Integrationssysteme nicht sinnvoll, da eine Normalform in eine unendliche Menge denormalisierter Formen transformiert werden kann.

3 Plattforminvariante Modellbildung

Im Rahmen dieses Abschnitts präsentieren wir ein plattformunabhängiges Kostenmodell für Integrationsprozesse. Weiterhin erläutern wir das Verfahren der Erhebung von Statistiken sowie deren Transformation und Vervollständigung.

Tabelle 1: Abstrakte und normalisierte Kosten der Operatoren des MTM [BBW+07]

Operatorname	**Abstrakte Kosten** $C(o_l)$	**Normalisierte Kosten** $NC(o_l)$
Receive	$\|ds_{out}\|$	$\frac{t_e(o_l)}{\|ds_{out}\|}$
Invoke	$\|ds_{in}\| + \|ds_{out}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$
Reply	$\|ds_{in}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|}$
Switch	-	$\sum_{i=1}^{n} P(pa_i)\left(\sum_{j=1}^{i} t_e(exppa_j) + \sum_{k=1}^{m_i} t_e(o_{i,k})\right)$
Fork	-	$\max_{i=1}^{n}\left(\sum_{j=1}^{m_i} t_e\left(o_{i,j}\right) + i \cdot C\left(thread\right)\right)$
Iteration	-	$m \cdot \sum_{i=1}^{n} t_e\left(o_i\right)$
Delay	-	$t_e(o_l)$
Signal	-	0
Assign	$\|ds_{in}\| + \|ds_{out}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$
Translation	$\|ds_{in}\| + \|ds_{out}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$
Selection	$\sigma = \|ds_{in}\|$ OR $\sigma_{sort} = \frac{\|ds_{in}\|}{2}$	$\frac{t_e(o_l)}{\|ds_{in}\|}$ OR $\frac{t_e(o_l)}{\frac{\|ds_{in}\|}{2}}$
Projection	$\|ds_{in}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|}$
Join	$\bowtie_{NL} = \|ds_{in1}\| + \|ds_{in1}\| \cdot \|ds_{in2}\|$	$\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in1}\|\cdot\|ds_{in2}\|}$
	$\bowtie_{SM} = \|ds_{in1}\| + \|ds_{in2}\|$	$\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in2}\|}$
Setoperation	$\cup_{all} = \|ds_{in1}\| + \|ds_{in2}\|$	$\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in2}\|}$
	$\cup, \cap, - = \|ds_{in1}\| + \|ds_{in1}\| \cdot \|ds_{in2}\|$	$\frac{t_e(o_l)}{\|ds_{in1}\|+\|ds_{in1}\|\cdot\|ds_{in2}\|}$
Split	$\|ds_{out}\|$	$\frac{t_e(o_l)}{\|ds_{out}\|}$
Orderby	$\|ds_{in}\| \cdot \log(\|ds_{in}\|)$	$\frac{t_e(o_l)}{\|ds_{in}\|\cdot\log(\|ds_{in}\|)}$
Groupby	$\|ds_{in}\| + \frac{\|ds_{in}\|^2}{2}$	$\frac{t_e(o_l)}{\|ds_{in}\|+\frac{\|ds_{in}\|^2}{2}}$
Window	$\frac{\|ds_{in}\|^2}{2}$	$\frac{t_e(o_l)}{\frac{\|ds_{in}\|^2}{2}}$
Validate	$\|ds_{in}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|}$
Savepoint	$\|ds_{in}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|}$
Action	$\|ds_{in}\| + \|ds_{out}\|$	$\frac{t_e(o_l)}{\|ds_{in}\|+\|ds_{out}\|}$

3.1 Kostenmodell

Das plattforminvariante Kostenmodell definiert Kosten für die einzelnen Operatoren des zentralen Message Transformation Models (MTM) [BBW+07]. Das Hauptproblem bei der Definition dieses Kostenmodells ist der systemübergreifende Charakter des Anwendungsgebietes. So genügt es auf der einen Seite eben *nicht*, lediglich abstrakte Kosten (beispielsweise auf Ebene von Kardinalitäten, wie im Datenbankbereich) miteinander zu vergleichen, da man hier nicht nur alternative Pläne für ein und dasselbe System miteinander vergleicht. Vielmehr müssen hier die normalisierten Verarbeitungszeiten der einzelnen Systeme einbezogen werden, um eine absolute Vergleichbarkeit zu schaffen. Auf der anderen Seite genügt aber auch der Vergleich der normalisierten Verarbeitungszeiten *nicht*, da dies kein Indiz über die Arbeitslast (im Sinne von Kardinalitäten) ist.

Folglich definieren wir ein zweistufiges Kostenmodell für die einzelnen Operatoren des MTM. Dieses enthält abstrakte Kosten $C(o_l)$ und normalisierte Kosten $NC(o_l)$ (siehe Ta-

belle 1). Dabei basieren die abstrakten Kosten auf Kardinalitäten, während die normalisierten Kosten mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$ aus den normalisierten (effektiven) Verarbeitungszeiten $t_e(o_l)$ und den abstrakten Kosten $C(o_l)$ berechnet werden. Damit kann dem systemübergreifenden Charakter Rechnung getragen werden. Im Minimum werden Kardinalitäten und Verarbeitungszeiten mit der Granularität einzelner Operatoren benötigt (Anforderungen an die Erhebung). Daraus können weitere Statistiken (wie zum Beispiel die relativen Häufigkeiten der Nutzung alternativer Prozesspfade) abgeleitet werden.

Im Detail sind die Operatoren des MTM in interaktions-, kontrollfluss- und datenflussorientierte Operatoren zu unterscheiden. Die abstrakten Kosten der Gruppe der interaktionsorientierten Operatoren sind aus der Semantik dieser abgeleitet. So kann `Receive` lediglich Daten empfangen und `Reply` lediglich Daten zurückgeben, während ein `Invoke` sowohl Input- als auch Output- Daten aufweisen kann. Die normalisierten Kosten ergeben sich hier mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$. Im Gegensatz dazu weisen die kontrollflussorientierten Operatoren keine abstrakten Kosten auf, da hier keine Datenoperationen vorgenommen werden. Die normalisierten Kosten ergeben sich hier wiederum aus der spezifischen Semantik der einzelnen Operatoren, wobei `Switch`, `Fork` und `Iteration` komplexe Operatoren darstellen und somit ebenfalls Kosten der enthaltenen Operatoren o_i einschließen. Für die Menge an datenflussorientierten Operatoren adaptierten wir das in [Mak07] dargestellte Kostenmodell eines relationalen Anfrageoptimierers und erweiterten es hinsichtlich der Operatoren des MTM zu dem abstrakten Kostenmodell. Die normalisierten Kosten ergeben sich mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$.

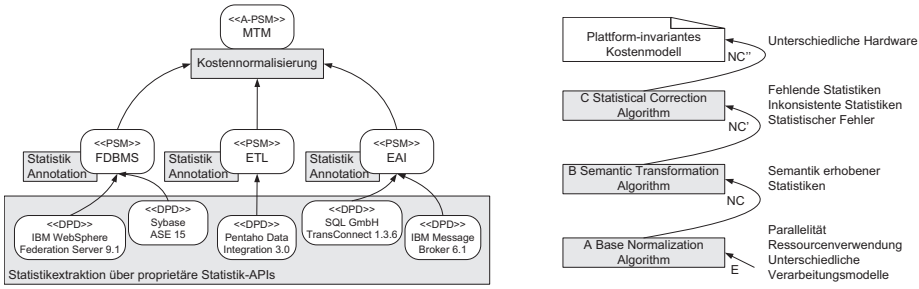
Um die Notwendigkeit von normalisierten Kosten für die systemübergreifende Kostenmodellierung zu veranschaulichen, nutzen wir das folgende illustrative Beispiel.

Beispiel 2 Systemübergreifender Kostenvergleich: *Nehmen wir an, ein Integrationsprozess P (einfache Sequenz von Operatoren: `Receive` o_1, `Translation` o_2 und `Invoke` o_3) wird sowohl von System $s1$ als auch von System $s2$ ausgeführt.*

Tabelle 2: Beispiel-Kostenvergleich auf der Basis normalisierter Kosten

Plan / System	**Statistik (Aggregat)**	o_1	o_2	o_3	$\sum$
P_{s1}	$t_e(o_l)$	10 ms	150 ms	70 ms	230 ms
	$\|ds_{in}\|$	-	1573	1345	
	$\|ds_{out}\|$	1573	1345	0	
	$C(o_l)$	1573	2918	1345	
	$NC(o_l)$	0.00636 ms	0.05141 ms	0.05204 ms	0.10981 ms
P_{s2}	$t_e(o_l)$	10 ms	140 ms	61 ms	211 ms
	$\|ds_{in}\|$	-	1105	1017	
	$\|ds_{out}\|$	1105	1017	0	
	$C(o_l)$	1105	2122	1017	
	$NC(o_l)$	0.00905 ms	0.06598 ms	0.05998 ms	0.13501 ms

Tabelle 2 stellt beispielhaft Statistiken für diesen Fall dar. Sofern wir ausschließlich die Kosten $C(o_l)$ respektive die Verarbeitungszeiten $t_e(o_l)$ betrachten würden, kämen wir zu der Fehlannahme, dass $s2$ das optimale System darstellen würde. Indem wir jedoch die gewichteten Kosten $NC(o_l)$ mit $NC(o_l) = \frac{t_e(o_l)}{C(o_l)}$ in die Betrachtung einschließen (und damit die relative Verarbeitungszeit pro Tupel berechnen), können wir die eigentliche systemübergreifende Optimalitätsentscheidung zu Gunsten von $s1$ fällen.



(a) Statistikextraktion (b) Kostennormalisierung

Abbildung 2: Überblick des Ablaufs der Statistikextraktion und Kostennormalisierung

Mit dieser Vorgehensweise kann also dem Problem der *unterschiedlichen Hardware* auf modellinhärente Weise begegnet werden. Weiterhin wird die Auswahlmöglichkeit über Systemgrenzen hinweg (bei disjunkten Prozessinstanzen) erst ermöglicht.

3.2 Erhebung von Verarbeitungsstatistiken

In diesem Abschnitt möchten wir, aufbauend auf dem eingeführten Kostenmodell, einen Überblick über die Erhebung von Verarbeitungsstatistiken und deren Normalisierung geben. In der Abbildung 2(a) wird die systemübergreifende Statistikextraktion und die Kostennormalisierung im Detail aufgezeigt.

Die Statistikerhebung erfolgt dabei in vier Schritten. Zunächst sind die Monitoring-Funktionalitäten der einzelnen Integrationssysteme zu nutzen, um Statistiken aufzunehmen. Diese können in einem zweiten Schritt über die jeweiligen proprietären Statistik-APIs extrahiert werden. Wir modellieren die erhobenen Basisstatistiken als eine Menge E von Statistiktupeln e_i mit $E = (e_1, e_2, \ldots, e_{n(m+1)})$. Dabei ergibt sich die Anzahl an Statistiktupeln N mit $N = |E| = n(m+1)$ aus der Menge der verarbeiteten Prozessinstanzen n und der Anzahl an Operatoren eines Prozesstyps m (plus 1, da Statistik des Gesamtprozesses enthalten ist). Weiterhin modellieren wir ein Statistiktupel e mit $e = (id, t0, t1, ds1, ds2, ds3)$, wobei $id \in \mathbb{N}$ einen Identifikator des Operators, $t0 \in \mathbb{R}$ die absolute Startzeit und $t1 \in \mathbb{R}$ die absolute Endzeit angibt. Darüber hinaus repräsentieren $ds1 \in \mathbb{N}$ und $ds2 \in \mathbb{N}$ die Kardinalitäten von Input-Datenmengen, während $ds3 \in \mathbb{N}$ die Kardinalität der Output-Datenmenge angibt. Dabei sind id, $t0$ und $t1$ obligatorisch für jeden Operator, während $ds1$, $ds2$ und $ds3$ in Abhängigkeit der Semantik der einzelnen Operatoren anzugeben sind. Auf systemtypspezifischer Ebene werden diese Basisstatistiken, in einem dritten Schritt, entsprechend den Systemtypen und Systemen mit Informationen annotiert. Bevor die Statistiken in dem vierten Schritt in das plattforminvariante Modell überführt werden können, muss die eigentliche Kostennormalisierung erfolgen.

Die Kostennormalisierung selbst (Abbildung 2(b)) erfolgt durch drei Algorithmen. Im ersten Algorithmus werden die Basisstatistiken normalisiert und aggregiert. Mit diesem Algorithmus können die drei Probleme *Parallelität*, *Ressourcenverwendung* und (mit Einschränkungen) *Unterschiedliche Verarbeitungsmodelle* behoben werden. Im Rahmen des zweiten Algorithmus werden die operatorspezifischen Statistiken von den plattformspezifischen Modellen in das MTM überführt. Dabei erfolgt die inverse Zuordnung zur Generie-

rung von Integrationsprozessen. Damit kann das Problem *Unterschiedliche Semantik der erhobenen Statistiken* beseitigt werden. Der dritte Algorithmus bereinigt die plattforminvarianten Statistiken, indem fehlende Werte geschätzt und korrigiert werden. Dies beseitigt die Probleme *Fehlende Statistiken*, *Inkonsistente Statistiken* und *Statistischer Fehler*.

4 Kostennormalisierung

Wie bereits angedeutet, umfasst die Kostennormalisierung drei aufeinander aufbauende Algorithmen, welche in diesem Abschnitt im Detail vorgestellt werden.

4.1 Parallelität, Ressourcenverwendung und Verarbeitungsmodelle

Der erste Algorithmus adressiert die Beseitigung von Parallelität, unterschiedliche Ressourcenverwendung und die Nutzung verschiedener Verarbeitungsmodelle. Das Grundkonzept soll zunächst an einem Beispiel veranschaulicht werden.

Beispiel 3 Grundnormalisierung: *Die Parallelität von Prozessinstanzen, und damit auch von Operatorinstanzen (Operatoren einer Prozessinstanz), beeinflusst die effektive Ressourcenverwendung und somit letztendlich auch die Verarbeitungszeit. Die Kernidee ist die Einteilung der Zeitskala in Abschnitte (bestimmt durch den Beginn und das Ende von Instanzen) und die Normalisierung der Abschnittszeit entsprechend den parallel genutzten Ressourcen (und entsprechend der Anzahl paralleler Instanzen num). Die Abbildung 3 zeigt ein Beispiel dieser Grundnormalisierung.*

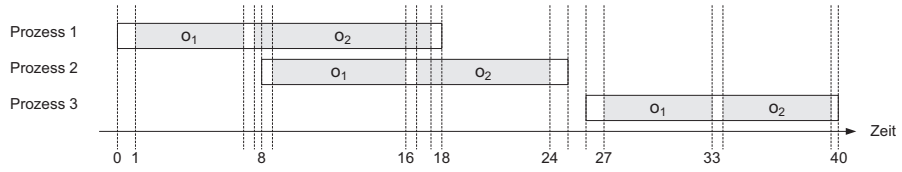


Abbildung 3: Beispiel einer Grundnormalisierung

Entsprechend der Normalisierungsvorschrift $t_e(o_l) = \frac{R_e(o)}{R_o(o)} \cdot (t1_{part} - t0_{part})$ *mit* $Re(o) = \min(\frac{1}{num}, Ro(o, e.id))$ *für die einzelnen Abschnitte, vergleicht Tabelle 3 die erhobenen Statistiken mit den normalisierten Statistiken. Hierbei nehmen wir* $R_o(P_i) = 0.1$, $R_o(o_1) = 0.6$ *und* $R_o(o_2) = 0.85$ *als maximale Ressourcenverwendung an (kann festgelegt oder aus den erhobenen Statistiken berechnet werden). Damit ergeben sich die normalisierten Kosten der einzelnen Abschnitte. Beispielsweise errechnen sich die Kosten des Operators* o_2 *der Prozessinstanz* $P2$ *mit* $t_e(P_2o_2) = \frac{0.5}{0.85} \cdot 1.0 + \frac{0.5}{0.85} \cdot 0.5 + \frac{0.85}{0.85} \cdot 6.0 = 6.88$. *Im Kostenvergleich der parallel ausgeführten Prozessinstanzen* P_1 *und* P_2 *mit der Prozessinstanz* P_3 *(ohne Parallelität) ist der Einfluss der Kostennormalisierung zu erkennen.*

Tabelle 3: Beispiel-Kostenvergleich auf der Basis gewichteter Kosten

	$t_e(o_1)$	**Normalisiert**	$t_e(o_2)$	**Normalisiert**	$t_e(P_i)$	**Normalisiert**
P_1	6 ms	6 ms	10 ms	6.09 ms	18 ms	14.09 ms
P_2	7.5 ms	6.25 ms	7.5 ms	6.88 ms	17 ms	15.13 ms
P_3	6 ms	6 ms	6 ms	6 ms	14 ms	14 ms
AVG	6.5 ms	6.08 ms	7.73 ms	6.32 ms	16.33 ms	14.41 ms

Der Algorithmus 1 zeigt eine schematische Realisierung dieses Konzeptes. Im Wesentlichen besteht dieser Algorithmus aus den drei Teilen Sortierung, Normalisierung und Aggregation. Die Sortierung erfolgt zum Zwecke einer verbesserten Auswertbarkeit der Minima während der Normalisierung. Innerhalb des zweiten Schritts wird über die Menge an Statistiken iteriert und jede Prozessinstanz in ihre logischen Abschnitte untergliedert (kann durch $t0$ und $t1$ von Prozessinstanzen und Operatorinstanzen bestimmt sein). Die normalisierte Abschnittszeit ergibt sich mit $ne.t_e = \frac{R_e(o)}{R_o(o)} \cdot (t1_{part} - t0_{part})$ aus der gemessenen Abschnittszeit gewichtet mit dem effektiven und maximalen Ressourcenverbrauch. Dabei

Algorithm 1 Base Normalization

Require: base statistic E, process plan P

1: $NE \leftarrow \oslash, NC \leftarrow \oslash$
2: *// P*art 1: sort base statistics
3: $E \leftarrow$ sort $E[t0_i]$ asc
4: *// P*art 2: normalize base statistics
5: **for** $i = 1$ to $|E|$ **do** *// foreach instance*
6: $ne \leftarrow$ create $ne(e.id, 0, ed.s1, e.ds2, e.ds3)$
7: $t0_{part} \leftarrow e.t0$
8: $t1_{part} \leftarrow 0$
9: **while** `TRUE` **do** *// foreach instance part*
10: *// determine the next part border*
11: **if** $t1_{part} > 0$ **then**
12: $t0_{part} \leftarrow t1_{part}$
13: **end if**
14: $t1_{part} \leftarrow \min(\min(\pi_{t0}(\sigma_{t0>t0_{part} \wedge t0 \leq e.t1} E)), \min(\pi_{t1}(\sigma_{t1>t0_{part} \wedge t1 \leq e.t1} E)))$
15: **if** $t1_{part}$ is `NULL` **then** *// no next part border*
16: **break**
17: **end if**
18: *// compute the normalized part time*
19: $num \leftarrow count\,(\texttt{distinct}\ \ e \in [t0_{part}, t1_{part}]\ E)$
20: $R_e(o) \leftarrow \min(\frac{1}{num}, R_o(o, e.id))$
21: $ne.t_e \leftarrow ne.t_e + \frac{R_e(o)}{R_o(o)} \cdot (t1_{part} - t0_{part})$
22: **end while**
23: $NE \leftarrow NE \cup ne$
24: **end for**
25: *// P*art 3: aggregate statistics
26: **for** $i = 1$ to $|NE|$ **do** *// foreach statistic tuple*
27: **if** $id \in NC$ **then**
28: $nc \leftarrow \sigma_{id=ne.id} NC$
29: **else**
30: $NC \leftarrow NC \cup nc$ where $nc \leftarrow$ create $nc(ne.id, 0, 0, 0, 0, 0)$
31: **end if**
32: $n \leftarrow nc.n, \quad nc.n \leftarrow n + 1, \quad nc.t_e \leftarrow \frac{n \cdot nc.t_e + ne.t_e}{n+1}$
33: $\forall k\ with\ 1 \leq k \leq 3 : nc.ds_k \leftarrow \frac{n \cdot nc.ds_k + ne.ds_k}{n+1}$
34: **end for**
35: **return** NC

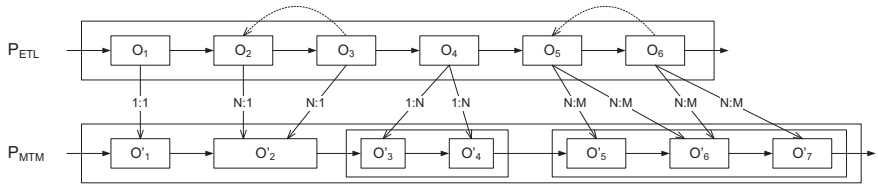


Abbildung 4: Beispiel-Transformation der Semantik von Operatoren

wird der effektive Ressourcenverbrauch mit $R_e(o) = \min(\frac{1}{num}, R_o(o, e.id))$ errechnet. Die Anzahl paralleler Abschnitte num wird hierbei als Menge disjunkter Instanzen genutzt. Somit wird beispielsweise eine Operatorinstanz o_1, jedoch nicht deren umschließende Prozessinstanz p_x betrachtet. Auf Grund des instanzbasierten Charakters nimmt eine Prozessinstanz bei ihrer Normalisierung stets die maximale Ressourcenverwendung $R_o(o)$ des aktuell ausgeführten Operators an. Im Rahmen der Aggregation wird das arithmetische Mittel über alle Instanzen mit gleicher id (eindeutig pro Prozesstyp) berechnet.

Theorem 1. *Der Algorithmus* Base Normalization *zur Beseitigung des Einflusses von Parallelität, Ressourcenverwendung und unterschiedlichen Verarbeitungsmodellen weist eine quadratische Komplexität von $O(n^2m^2)$ auf.*

Proof. Zum Beweis von Theorem 1 sind die Einzelkomplexitäten der drei Algorithmusteile zu beweisen. Nehmen wir an, dass die Anzahl der Statistiktupel N mit $N = n \cdot (m+1)$ bestimmt ist, wobei n die Anzahl der Prozessinstanzen und m die Anzahl der Operatoren angibt (plus 1, da für die gesamte Prozessinstanz). Teil 1 des Algorithmus sortiert die erhobenen Basisstatistiken. Es ist bekannt, dass die Sortierung von E mit $O(N \log(N))$ möglich ist. Der zweite Teil des Algorithmus normalisiert die Basisstatistiken. Dies wird auf Ebene der Instanzabschnitte realisiert. Dabei kann die i-te Instanz im schlimmsten Fall $2i-1$ Abschnitte aufweisen (im Fall, dass jede i-te Instanz die $i+1$-te Instanz vollständig überdeckt). Insgesamt sind somit also maximal

$$\sum_{i=1}^{n(m+1)} (2i-1) = (2 \sum_{i=1}^{n(m+1)} i) - n(m+1) = n^2m^2 + 2n^2m + n^2$$

Abschnitte zu evaluieren. Auf Grund der sortierten Reihenfolge der Statistiken, nehmen wir konstante Komplexität mit $O(1)$ für die Berechnung der folgenden Abschnittsgrenzen an. Somit weist der zweite Teil des Algorithmus eine quadratische Komplexität mit $O(n^2m^2) = O(n^2m^2 + 2n^2m + n^2) \cdot O(1)$ auf. Der dritte Teil des Algorithmus realisiert die Aggregation der Statistiken ($n \cdot (m+1)$) mit linearer Komplexität von $O(n \cdot m)$). Zusammenfassend weist der Algorithmus entsprechend des Theorems eine quadratische Komplexität mit $O(n^2m^2) = O(nm \log(nm)) + O(n^2m^2) + O(nm)$ auf.

4.2 Transformation der Semantik von Operatoren

Aufbauend auf dem Ergebnis der Grundnormalisierung ist im zweiten Schritt die Normalisierung der Semantik von Operatorstatistiken – invers zur Generierung – zu realisieren. Das folgende Beispiel veranschaulicht dieses Problem.

Beispiel 4 Semantiktransformation: *In Abbildung 4 ist (auf schematische Weise) das Problem der Überführung von Statistiken aufgezeigt. Hier wurden Statistiken für einen ETL-Job erhoben und müssen nun in die plattforminvariante Form überführt werden. Dieses*

Problem adressiert unterschiedliche Operatoren beider Modellebenen. Sofern eine einfache `1:1` Abbildung besteht, können die Statistiken direkt übernommen werden. Auch eine `N:1`-Abbildung ist unproblematisch, da hier lediglich die Einzelstatistiken der N-Seite zu aggregieren sind. Bei `1:N`- oder `N:M`-Abbildungen ist jedoch der komplexe Operator `Process` zur Repräsentation der Statistiken zu nutzen. Im Beispiel würden die Statistiken des Operators o_4 also zwar übernommen, jedoch könnten keine Rückschlüsse auf die Einzeloperatoren o'_3 und o'_4 gezogen werden.

Die Realisierung der Semantiktransformation erfolgt mit Hilfe des Algorithmus *Semantik Transformation* (Algorithmus 2). Zunächst werden die Operatorsequenzen des plattformspezifischen Prozessplans P und des plattformunabhängigen Prozessplans P' ermittelt. Anschließend arbeitet der Algorithmus linear über alle Operatoren $o_i \in P$. Sofern o_i ein komplexer Operatortyp ist, wird der Algorithmus rekursiv aufgerufen. Weiterhin wird

Algorithm 2 Semantic Transformation

Require: normalized stats NC, node P, node P' *// a process plan is a specialized node*
1: $NC' \leftarrow \emptyset, \quad o \leftarrow o \in P, \quad o' \leftarrow o' \in P'$
2: **for** $i = 1$ to $|o|$ **do** *// foreach operator in o*
3: **if** $o_i \in$ `Process`, `Switch`, `Fork`, `Iteration` **then**
4: $NC' \leftarrow NC' \cup$ execute *Semantic Transformation*(NC, o_i, P')
5: **end if**
6: $struct \leftarrow$ execute *evalMappingRule*(o_i, o')
7: **if** $struct \in$ `1:1` **then**
8: *// copy operator statistics*
9: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(\pi_{struct.id,n,t_e,ds1,ds2,ds3}(\sigma_{id=id(o_i)}NC))$
10: **else if** $struct \in$ `1:N` **then**
11: *// copy operator statistics to parent node*
12: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(\pi_{struct.id,n,t_e,ds1,ds2,ds3}(\sigma_{id=id(o_i)}NC))$
13: **for** $j = 1$ to $|o' \in struct|$ **do** *// foreach operator $o'_j \in struct$*
14: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(id(o'_j), -1, -1, -1, -1, -1)$
15: **end for**
16: **else if** $struct \in$ `N:1` **then**
17: *// sum up and copy statistics*
18: $t_e \leftarrow \pi_{\sum_{j=1 \wedge o_j \in struct}^{i}}(\sigma_{id=id(o'_j)}NC)$
19: $n, ds1, ds2, ds3 \leftarrow$ operator specific copy of statistics
20: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(struct.id, n, t_e, ds1, ds2, ds3)$
21: **else if** $struct \in$ `N:M` **then**
22: *// sum up statistics and copy to parent node*
23: $t_e \leftarrow \sum_{j=1 \wedge o_j \in struct}^{i} \pi_{t_e}(\sigma_{id=id(o_j)}NC)$
24: $n, ds1, ds2, ds3 \leftarrow$ operator specific copy of statistics
25: **for** $j = 1$ to $|struct|$ **do** *// foreach operator in struct*
26: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(id(o_j), -1, -1, -1, -1, -1)$
27: **end for**
28: $NC' \leftarrow NC' \cup nc$ where $nc \leftarrow$ create $nc(struct.id, n, t_e, ds1, ds2, ds3)$
29: **end if**
30: **end for**
31: **return** NC'

im Wesentlichen die inverse Abbildungsregel zwischen dem Operator o_i und der Operatormenge o' bestimmt. Die resultierende Datenstruktur wird nun hinsichtlich der vier konzeptuellen Abbildungsmöglichkeiten (`1:1`, `1:N`, `N:1` und `N:M`) unterschieden. Im Falle der `1:1`-Abbildung erfolgt eine einfache Kopie der Statistiken. Schwieriger stellt sich die `1:N`-Abbildung dar. Hier werden die Statistiken für den kompletten Teilgraphen (welcher mehrere Operatoren enthält) abgebildet, während für die Einzeloperatoren fehlende Statistiken vermerkt werden, so dass diese im nachfolgenden Algorithmus zu schätzen sind. Bei der `N:1` Abbildung kann hingegen einfach die Summe über die Verarbeitungszeiten gebildet und die Start- und Endkardinalitäten können schlicht übernommen werden. Die `N:M`-Abbildung ist eine Kombination aus `1:N`- und `N:1`-Abbildungen. So werden die Statistiken zunächst (ähnlich zu `N:1`) zusammengefasst und anschließend (ähnlich zu `1:N`) auf einen Teilgraphen abgebildet.

Theorem 2. *Der Algorithmus* Semantic Transformation *zur Beseitigung des Einflusses der Semantik plattformspezifischer Operatoren weist eine quadratische Komplexität von* $O(m^2 + m')$ *auf.*

Proof. Generell arbeitet der Algorithmus linear über alle Operatoren des plattformspezifischen Prozessplans. Im schlimmsten Fall (N:1 oder N:M) müssen alle Statistiken des plattformspezifischen Prozesses aggregiert werden. Hierfür ergeben sich

$$\sum_{i=1}^{m}(m-i)+m' = m(m+1) - \frac{(m+1)^2 - m + 1}{2} + m' = \frac{m^2 - m}{2} + m'$$

notwendige Operatorevaluierungen. Folglich weist der Algorithmus eine quadratische Komplexität mit $O(m^2 + m') = O(\frac{m^2-m}{2} + m')$ auf.

4.3 Statistische Korrektur

Anschließend erfolgt die statistische Korrektur im Sinne des Erkennens von Inkonsistenzen zwischen Statistiken sowie deren Vervollständigung. Dabei können Inkonsistenzen ausschließlich durch die Integrationssysteme verursacht worden sein, während fehlende Statistiken durch die Integrationssysteme und durch die zuvor vorgenommene Transformation der Semantik (`1:N`- und `N:M`-Abbildungen) begründet sein können.

Beispiel 5 Statistikkorrektur: *Nehmen wir einen Prozess P an, welcher eine Sequenz von vier Operatoren o_1 bis o_4 umfasst. Für o_1 und o_4 liegen jeweils Statistikvektoren $NC(o_i)$ vor, worin Input-Datenmengen $|ds_{in}|$, Output-Datenmengen $|ds_{out}|$ sowie die effektive Verarbeitungszeit $t_e(o_i)$ enthalten sind. Für die Operatoren o_2 und o_3 existieren keine Einzelstatistiken sondern lediglich Aggregate über den Teilprozess sp (o_2, o_3). Um die fehlenden Statistiken zu berechnen, können wir zunächst $|ds_{in}(o_2)| = |ds_{in}(sp)|$ sowie $|ds_{out}(o_3)| = |ds_{out}(sp)|$ setzen. Nun besteht das Problem der Schätzung der Kardinalitäten der Zwischenresultate. Hierzu wenden wir mit $|ds_{out}(o_2)| = |ds_{in}(sp)| -$*

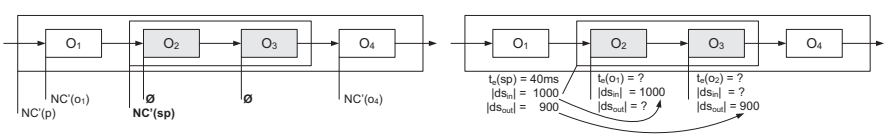


Abbildung 5: Beispielkorrektur der normalisierten Statistiken

(2) $\frac{|ds_{in}(sp)|-|ds_{out}(sp)|}{2}$ *und* $|ds_{in}(o_3)| = |ds_{in}(sp)| - (3-1)\frac{|ds_{in}(sp)|-|ds_{out}(sp)|}{2}$ *die lineare Extrapolation an und kommen zum Ergebnis* $|ds_{out}(o_2)| = |ds_{in}(o_3)| = 950$*. Nun können wir mit Hilfe der abstrakten Kosten* $C(o_i)$ *und der effektiven Verarbeitungszeit* $t_e(sp)$ *die fehlenden Verarbeitungszeiten mit* $t_e(o_i) = t_e(sp) \cdot \frac{C(o_i)}{\sum_{j=1}^{|o|} C(o_j)}$ *berechnen. Sofern wir von den Operatoren* `Translation` *und* `Invoke` *ausgehen, ergibt sich hier:* $t_e(o_2) = 40ms \cdot \frac{1950}{3800} = 20.53ms$ *und* $t_e(o_3) = 40ms \cdot \frac{1850}{3800} = 19.47ms$*.*

Der Algorithmus 3 (*Statistical Correction*) zeigt diese statistische Korrektur. Zunächst wird versucht, Inkonsistenzen zwischen Statistiken aufzudecken. Hierzu wird für alle Operatoren die Bedingung $|ds_{out}(o_i)| \neq |ds_{in}(o_{i+1})|$ überprüft. Wir können hier Gleichheit fordern, da wir zur Aggregation das arithmetische Mittel und nicht die exponentielle

Algorithm 3 Statistical Correction

Require: operator sequence o, total costs $NC(p)$, operator costs $NC(o)$
execute *assignCosts*$(o, NC(p), NC(o))$ *//assign costs to nodes with* $id(o) = nc.id$
$flag \leftarrow$ **false**, $sum \leftarrow 0$
// check for inconsistencies between statistics
for $i = 1$ to $|o| - 1$ **do** *// foreach operator in o*
  if $|ds_{out}(o_i)| \neq |ds_{in}(o_{i+1})|$ **then**
    $flag \leftarrow$ **true**; **break**
  end if
  $sum \leftarrow sum + t_e(o_i)$
end for
$sum \leftarrow sum + t_e(o_m)$
if $|ds_{in}(P)| \neq |ds_{in}(o_1)|$ `OR` $|ds_{out}(P)| \neq |ds_{out}(o_m)|$ `OR` $sum > t_e(P)$ **then**
  $flag \leftarrow$ **true**
end if
if $flag$ **then**
  $\forall i$: clear statistics $NC(o_i) \leftarrow -1$
end if
// check for missing statistics
if $flag$ `OR` $\forall i : NC(o_i) = \oslash$ **then**
  // recompute missing statistics (linear extrapolation)
  for $i = 1$ to $|o|$ **do** *// foreach operator in o*
    $ds_{in}(o_i) \leftarrow ds_{in}(P) - (i-1)\frac{ds_{in}(P)-ds_{out}(P)}{|o|}$
    $ds_{out}(o_i) \leftarrow ds_{in}(P) - (i)\frac{ds_{in}(P)-ds_{out}(P)}{|o|}$
    $C(o_i) \leftarrow C(ds_{in}(o_i), ds_{out}(o_i))$
  end for
  for $i = 1$ to $|o|$ **do** *// foreach operator in o*
    $t_e(o_i) \leftarrow t_e(P) \cdot \frac{C(o_i)}{\sum_{j=1}^{|o|} C(o_j)}$
    if $o_i \in$ `Process`, `Switch`, `Fork`, `Iteration` **then**
      execute *Statistical Correction* for o_i
    end if
  end for
end if
return NC'

Glättung verwenden. Sofern in diesem Schritt Ungleichheit festgestellt wurde, die Summe der Verarbeitungszeiten nicht mit dem hierarchisch übergeordneten Knoten übereinstimmt oder die Input- und Output-Kardinalitäten des Prozesses nicht mit dem ersten respektive letzten Operator übereinstimmen, werden die Statistiken entsprechend gelöscht. Im zweiten Hauptschritt des Algorithmus werden die fehlenden Statistiken mit Hilfe linearer Extrapolation geschätzt. Hierzu sind in einem ersten Schleifendurchlauf die Input- und Output-Kardinalitäten $ds_{in}(o_i)$ und $ds_{out}(o_i)$ in Abhängigkeit der Statistiken des übergeordneten Knotens zu schätzen, sowie die abstrakten Kosten $C(o_i)$ (siehe Tabelle 1) zu summieren. In einem zweiten Schleifendurchlauf (die Summe wird hier benötigt) wird dann letztendlich die Verarbeitungszeit $t_e(o_i)$ mit $t_e(o_i) = t_e(P) \cdot \frac{C(o_i)}{\sum_{j=1}^{|o|} C(o_j)}$ geschätzt. Sofern ein komplexer Operatortyp vorliegt, wird auch hier rekursiv der Algorithmus für Teilgraphen ausgeführt. Dies erfolgt aber bewusst nachdem die Statistiken für diesen Knoten geschätzt wurden. Somit kann die Schätzung des Teilgraphs auf Schätzungen des Elternknotens beruhen.

Theorem 3. *Der Algorithmus* Statstical Correction *zur Korrektur und Vervollständigung von Statistiken weist eine lineare Komplexität von $O(m)$ auf.*

Proof. Im schlimmsten Falle (fehlende Statistiken für den letzten Operator) werden vier Schleifendurchläufe über die Menge an Operatoren zur statistischen Berichtigung benötigt. Somit ergibt sich eine lineare Komplexität von $O(m) = O(4m)$.

5 Experimentelle Evaluierung

Aufbauend auf den theoretischen und konzeptionellen Ergebnissen dieser Arbeit wurde die praktische Anwendbarkeit des Ansatzes an Hand von Experimenten evaluiert. Hierzu realisierten wir die Algorithmen im Rahmen unserer `WFPE` (*Workflow Process Engine*, in Java 1.6 implementiert). Im Wesentlichen evaluieren wir funktionale (Qualität der Normalisierung) und nichtfunktionale (Performanz) Aspekte der Kostennormalisierung. Die Experimente wurden auf einem standard Balde (OS Suse Linux) mit zwei Prozessoren (jeder davon ein Dual Core AMD Opteron Processor 270 mit 1994 MHz) und 8.9 GB RAM ausgeführt. Zur Erhebung von Statistikinformationen führten wir Integrationsprozesse des DIPBench [BHLW08a] (und Derivate dieser Prozesse) mit der `WFPE` und unterschiedlichen Workload-Charakteristika aus. Im Rahmen dieser experimentellen Evaluation analysierten wir systematisch die folgenden Einflussfaktoren: Anzahl der Prozessinstanzen n, Kardinalität der Input-Daten $d = |ds_{in}(P)|$, Verarbeitungsmodell v, Anzahl der Operatoren m sowie die Menge der Statistik-Tupel N.

5.1 Funktionale Evaluierung der Kostennormalisierung

Zum Zwecke der Evaluierung der Qualität der Kostennormalisierung nutzen wir zwei unterschiedliche Prozesstypen. Der Prozesstyp $P13$ wurde innerhalb des DIPBench spezifiziert und adressiert die Extraktion von Bestellungen und Bestellpositionen aus einer konsolidierten Datenbank und das Laden dieser Daten in ein Data-Warehouse. Dabei impliziert der Skalierungsfaktor Datenmenge d, mit $d = 1.0$, 70.000 Bestellungen und 350.000 Bestellpositionen. Der Prozesstyp P_{m5} besteht aus einer Sequenz von fünf Operatoren

Tabelle 4: Workload-Skalierung im Rahmen der funktionalen Evaluierung

Prozessinstanz n_i	**Skalierung p in Experiment 1**	**Skalierung d in Experiment 2**
$0 < n_i \leq 30.000$	$p = 1$	$d = 0.0001$
$30.000 < n_i \leq 60.000$	$p = 2$	$d = 0.0002$
$60.000 < n_i \leq 90.000$	$p = 3$	$d = 0.0003$
$90.000 < n_i \leq 120.000$	$p = 4$	$d = 0.0004$
$120.000 < n_i \leq 150.000$	$p = 3$	$d = 0.0003$
$150.000 < n_i \leq 180.000$	$p = 2$	$d = 0.0002$
$180.000 < n_i \leq 210.000$	$p = 1$	$d = 0.0001$

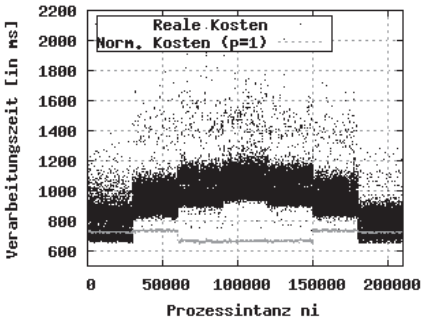
($m = 5$), welche (1) eine Anfrage vorbereiten, (2) ein $100kb$ großes XML-Dokument laden, (3) ein Schema-Mapping des Dokumentes durchführen, (4) eine weitere Anfrage vorbereiten und letztendlich (5) das Ergebnis versenden. Zur Skalierung der Operatoranzahl m werden diese Operatoren periodisch wiederholt.

Die Abbildung 6 zeigt die Qualität der Kostennormalisierung für variierende Workload-Charakteristika (Grad der Parallelität p / Datenmenge d) sowie die Qualität im Hinblick auf unterschiedliche Verarbeitungsmodelle.

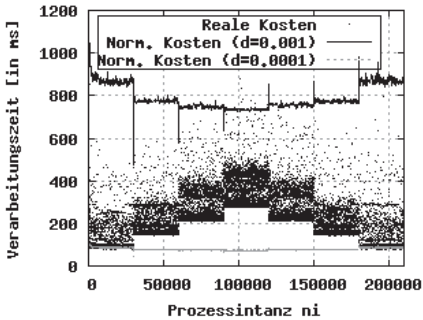
Experiment 1: In diesem Experiment (Abbildung 6(a)) wurden $n = 210.000$ Prozessinstanzen von $P13$ mit einer Datenmenge von $d = 0.001$ mit der `WFPE` ausgeführt. Hierbei wurde der Grad der Parallelität p – im Sinne der gleichzeitigen Initiierung von p Prozessinstanzen – variiert (siehe Tabelle 4, Spalte 2). Anschließend wurden die Erhobenen Statistikdaten mit Hilfe der Normalisierungsalgorithmen auf $p = 1$ normalisiert (jeweils in einem Intervall von 210 Instanzen). Dabei war die maximale Ressourcenauslastung $R_o = 0.75$. Es ist zu erkennen, dass die Normalisierung für $p = 2$ sehr gute Ergebnisse liefert, während es für $p > 2$ zu einer Unterschätzung kommt. Dies ist mit der sehr guten Skalierung der Hardware bei steigender Parallelität zu erklären.

Experiment 2: Weiterhin wurden in einem zweiten Experiment (Abbildung 6(b)) abermals $n = 210.000$ Prozessinstanzen von $P13$ (mit konstantem $p = 1$) ausgeführt. Dabei variierten wir die Datenmenge d entsprechend Tabelle 4, Spalte 3. Die Normalisierung (auf $d = 0.001$ und $d = 0.0001$) erfolgte ähnlich zu Experiment 1. Offensichtlich ist eine hohe Qualität der Normalisierung zu erkennen, wobei lediglich geringe Ausreißer an den Grenzen des Workload-Wechsels auftraten.

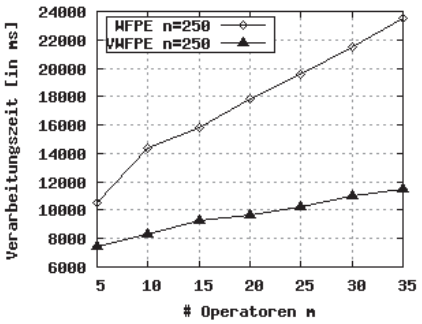
Experiment 3: Im dritten Experiment hinterfragten wir die Normalisierung unterschiedlicher Verarbeitungsmodelle. Während die `WFPE` standardmäßig ein instanzbasiertes Verarbeitungsmodell (jede Instanz ein Thread, kein Pipelining) nutzt, ermöglicht sie auch die vektorisierte Prozessausführung `VWFPE` (Verarbeitungsmodell *pipes and filter*), wobei jeder Operator als Thread ausgeführt wird und Pipelining von Nachrichten zwischen Operatoren erfolgt. Die Abbildung 6(c) zeigt den Durchsatzvergleich (Ausführung von 250 Prozessinstanzen) der beiden Ausführungsmodelle für den Prozesstyp P_{mX} (mit $m = X$), wobei wir X von 5 bis 35 Operatoren skalierten und das Experiment 20 mal wiederholten. Der bessere Durchsatz wird durch die höhere Parallelität erzielt. Nun war zu untersuchen, ob die Normalisierung auch über Verarbeitungsmodelle hinweg gute Ergebnisse liefert. Hierbei erwarteten wir, dass die Einzelverarbeitungszeiten der `VWFPE` höher als die der `WFPE` wären und die Normalisierung der `VWFPE`-Zeiten ähnlich der `WFPE`-Zeiten wären. Die Abbildung 6(d) zeigt das Resultat dieses Experiments. Offensichtlich kann die



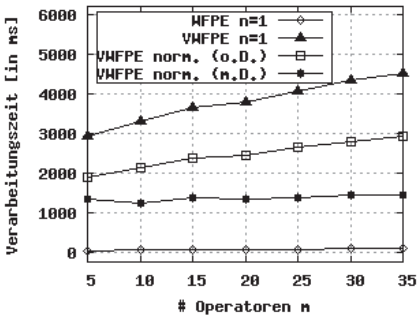
(a) Kostennormalisierung Parallelität p



(b) Kostennormalisierung Datenmenge d



(c) Durchsatzvergleich Verarbeitungsmodelle



(d) Kostennormalisierung Verarbeitungsmodell

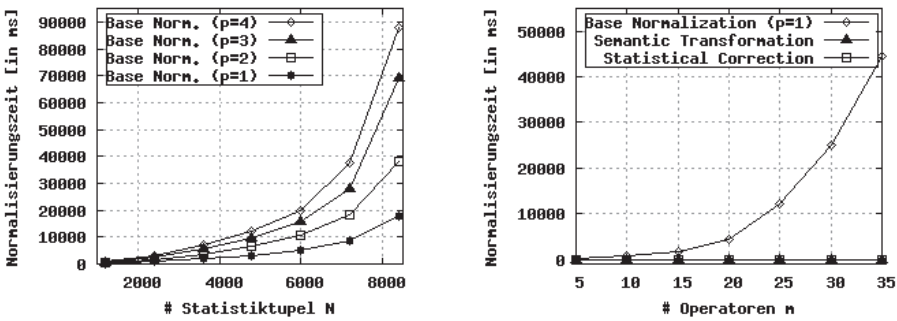
Abbildung 6: Qualität der Kostennormalisierung bei unterschiedlichen Einflussfaktoren

Normalisierung nicht das Problem unterschiedlicher Verarbeitungsmodelle beheben und neigt hier zu extremen Überschätzungen. Dieses negative Ergebnis ist auch plausibel zu erläutern. So funktioniert die Normalisierung der Verarbeitungszeiten einzelner Operatoren auch für unterschiedliche Verarbeitungsmodelle. Die Normalisierung von Prozessplänen ist aber viel komplexer, da hier die Verarbeitungszeit extrem vom Füllstand der Warteschlangen (jede Kante im Operatorgraph wird durch eine Queue repräsentiert) zwischen den Operatoren abhängt. Folglich kann es bei hohem Füllstand zu sehr langen Gesamtzeiten und damit zu starken Überschätzungen kommen. Als Schlussfolgerung dieses Ergebnisses bleibt zu erwähnen, dass die Normalisierung unterschiedlicher Verarbeitungsmodelle eine offene Forschungsfrage darstellt, der möglicherweise mit einer durchsatzbasierten oder operatorbasierten Normalisierung begegnet werden kann.

5.2 Nichtfunktionale Evaluierung der Kostennormalisierung

Neben dem Aspekt der Qualität der Kostennormalisierung stellt sich im Kontext der kontinuierlichen Kostennormalisierung natürlich auch die Frage der Effizienz der Algorithmen. Diese wird mit der nichtfunktionalen Evaluierung untersucht.

Experiment 4: Im vierten Experiment wird zunächst der Algorithmus *Base Normalization* hinsichtlich der Skalierung mit zunehmender Anzahl an Statistiktupeln N (mit $N = m \cdot n$)



(a) Performance unter Skalierung n (b) Performance unter Skalierung m

Abbildung 7: Effizienz der Kostennormalisierung

evaluiert. Dabei wurden die Statistikdaten aus Experiment 1 wiederverwendet. Da die Anzahl der parallelen Instanzen einen Einfluss auf die Laufzeit des Algorithmus hat, haben wir den Grad der Parallelität p in das Experiment eingeschlossen. Die Abbildung 7(a) visualisiert das Ergebnis und bestätigt die quadratische Komplexität von $O(m^2n^2)$. Weiterhin zeigt das Ergebnis eine steigende Laufzeit mit steigendem Grad der Parallelität, da hier mehr Teilabschnitte zu normalisieren sind. Die beiden anderen Algorithmen verarbeiten aggregierte Daten und hängen damit nur von m (linear von N) ab.

Experiment 5: In einem fünften und letzten Experiment verwendeten wir die Statistikdaten aus Experiment 3 wieder, um den Einfluss der Anzahl an Operatoren m auf die Laufzeit der Algorithmen zu analysieren. Die Abbildung 7(b) zeigt das Ergebnis, wobei klar zu erkennen ist, dass der Algorithmus *Base Normalization* die Gesamtlaufzeit zweifelsohne dominiert. Dabei ist wiederum die quadratische Komplexität dieses Algorithmus zu erkennen. Da die beiden anderen Algorithmen ausschließlich auf aggregierten Daten N' mit $N' \ll N$ arbeiten, hängen diese nur von m ab. Die Laufzeit dieser Algorithmen steigt weiterhin linear mit der Anzahl an Operatoren m.

6 Verwandte Arbeiten

Im Rahmen verwandter Arbeiten wollen wir zum einen Anwendungsgebiete aufzeigen, in welchen die systemübergreifende Kostennormalisierung zum Einsatz kommen kann. Darüber hinaus wollen wir angrenzende Arbeiten darstellen, welche ebenfalls die Notwendigkeit der Kostennormalisierung respektive die Normalisierung der Arbeitslast zum Gegenstand haben. Generell lässt sich hierbei einschätzen, dass (1) Vorarbeiten (unsere eigenen eingeschlossen) im Bereich der Generierung von Integrationsprozessen ein Anwendungsgebiet darstellt und, dass (2) bislang noch kein Ansatz existiert der die *systemübergreifende* Kostennormalisierung adressiert.

Generierung und Deployment von Integrationsprozessen. Im Kontext der Generierung und des Deployments von Integrationsprozessen sind im Wesentlichen drei Ansätze hervorzuheben. Das Orchid-Projekt [DHW+08] befasst sich mit der Generierung von ETL-Jobs auf Grundlage von deklarativen Mapping-Spezifikationen. Dabei wird hier ein sogenanntes Operator Hub Model (OHM) verwendet, um die Verarbeitungssemantik von ETL-

Jobs plattformunabhängig abzubilden. Das Orchid-Projekt beschränkt sich momentan ausschließlich auf die Generierung von ETL-Prozessen für IBM-ETL-Werkzeuge. Konzeptionell scheint die Erweiterung auf den gesamten (herstellerunabhängigen) ETL-Bereich problemlos möglich. Im Gegensatz dazu, adressiert eine andere Arbeit [AN08] die Verbindung der Generierung von ETL-Prozessen und des Model Managements, indem plattformunabhängige Operatoren für das Deployment von ETL-Prozessen vorgestellt wurden. Unser GCIP Framework (Generation of Complex Integration Processes) hingegen adressiert die Modellierung von plattformunabhängigen Integrationsprozessen [BWHL08], die Generierung für *unterschiedliche* Integrationssystemtypen (wie zum Beispiel für FDBMS, EAI und ETL) sowie die Anwendung von Optimierungstechniken [BWHL08,BHLW08b] im Rahmen der modellgetriebenen Generierung. Die Gemeinsamkeit der drei Projekte liegt in der Auswahlmöglichkeit des optimalen Integrationssystems (Bestimmung des Ziels der Generierung). Als Grundlage für eine derartige Optimalitätsentscheidung muss es eine systemübergreifende Kostennormalisierung geben. Folglich stellen alle drei genannten Projekte mögliche Anwendungsgebiete des vorgestellten Ansatzes dar.

Kostennormalisierung in verteilten Datenbanksystemen. Während die Statistiknormalisierung in der Mathematik und des Operations Research ein oft verwendetes Mittel zur Herstellung der Vergleichbarkeit ist (beispielsweise im Rahmen der Data Envelopment Analysis [SCPP07]), existieren im Bereich der Normalisierung von Verarbeitungsstatistiken sehr wenige Arbeiten auf angrenzenden Gebieten. Ähnliche Probleme existieren im Bereich der Anfrageverarbeitung in verteilten Datenbanksystemen. Im Wesentlichen sind an dieser Stelle zwei Gruppen von Ansätzen zur Kostennormalisierung zu unterscheiden. Die Gruppe der *hierarchischen Ansätze* versucht eine Hierarchie von Kostenbewertungen über die Quellsysteme in statischer Weise aufzubauen. So werden systemübergreifend logische Statistiken zum Vergleich verwendet, während die einzelnen Wrapper und Quellsysteme für die Überführung der physischen Statistiken in logische Statistiken verantwortlich sind. Ein interessanter Ansatz dieser Kategorie [NGT98] definiert einen sogenannten hierarchic cost formula tree, wobei die Kostenbewertung einer Anfrage in unterschiedliche Aspekte untergliedert wird. Ein anderes Beispiel für die Kategorie ist das Garlic-Projekt [ROH99,JSHL02] wobei relationale und nicht-relationale Datenquellen Statistikinformationen bereitstellen und nur bei Fehlen dieser Statistiken Schätzungen an zentraler Stelle vorzunehmen sind. Im Gegensatz zu diesen statischen Ansätzen adressiert die Gruppe der *dynamischen Ansätze* die adaptive Kostenbewertung der einzelnen Datenquellen, wobei die Kostenmodelle dynamisch zur Laufzeit, basierend auf dem Vergleich von geschätzten und realen Verarbeitungsstatistiken, verändert werden. Als Beispiele sind hier die Realisierung von verteiltem Caching [SW97], die Kalibrierung der Koeffizienten eines Kostenmodells für IRO-DB [GST96] und die evolutionäre Anpassung des Anfragekostenmodells [RZL04] zu nennen. Diese Ansätze der verteilten Datenbanksysteme können nicht alle identifizierten Probleme der systemübergreifenden Kostennormalisierung beheben, da hier lediglich relative Kosten innerhalb der einzelnen Systeme miteinander verglichen werden und damit keine systemübergreifende Vergleichbarkeit von Statistiken hergestellt wird.

7 Zusammenfassung und Ausblick

Ausgehend von der Generierungsmöglichkeit von Integrationsprozessen für unterschiedliche Integrationssysteme wurden in dieser Arbeit die wesentlichen Probleme der sys-

temübergreifenden Kostennormalisierung identifiziert. Im Detail wurde eine generelle Methodik zur Extraktion und Normalisierung von Statistiken aus konkreten Integrationssystemen aufgezeigt. Hierzu diskutierten wir zunächst ein plattforminvariantes Kostenmodell und stellten drei aufeinander aufbauende Algorithmen zur Kostennormalisierung vor. Die experimentelle Evaluierung hat gezeigt, dass die Normalisierung (mit Einschränkungen) qualitativ gute Ergebnisse liefert und damit die Vergleichbarkeit von Integrationssystemen sichert (Annahme 3). Dies ist in unterschiedlichen Anwendungsgebieten einsetzbar, wobei jedoch noch offene Probleme existieren. So sind die Minimierung von statistischen Fehlern im Rahmen der Kostenvervollständigung, die Einbeziehung von Hintergrundlast andere Systeme, inkrementelle Normalisierungsalgorithmen sowie unterschiedliche Verarbeitungsmodelle genauer zu untersuchen.

Literatur

[AN08] Alexander Albrecht and Felix Naumann. Managing ETL Processes. In *NTII*, 2008.

[BBW+07] Matthias Böhm, Jürgen Bittner, Uwe Wloka, Dirk Habich, and Wolfgang Lehner. Ein Nachrichtentransformationsmodell für komplexe Transformationsprozesse in datenzentrischen Anwendungsszenarien. In *BTW*, 2007.

[BF07] Susanne Busse and Johann Christoph Freytag. Entwurf von Informationsintegrationssystemen auf der Basis der Merkmalsmodellierung. In *BTW*, 2007.

[BHLW08a] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. DIPBench Toolsuite: A Framework for Benchmarking Integration Systems. In *ICDE*, 2008.

[BHLW08b] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. Workload-Based Optimization of Integration Processes. In *CIKM*, 2008.

[BWHL08] Matthias Böhm, Uwe Wloka, Dirk Habich, and Wolfgang Lehner. Model-Driven Generation and Optimization of Complex Integration Processes. In *ICEIS (1)*, 2008.

[DHW+08] Stefan Dessloch, Mauricio A. Hernández, Ryan Wisnesky, Ahmed Radwan, and Jindan Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, 2008.

[GST96] Georges Gardarin, Fei Sha, and Z.-H. Tang. Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System. In *VLDB*, 1996.

[JSHL02] Vanja Josifovski, Peter M. Schwarz, Laura M. Haas, and Eileen Tien Lin. Garlic: a new flavor of federated query processing for DB2. In *SIGMOD*, 2002.

[Mak07] Mazeyar E. Makoui. *Anfrageoptimierung in objektrelationalen Datenbanken durch kostenbedingte Termersetzungen*. PhD thesis, Universität Hannover, 2007.

[NGT98] Hubert Naacke, Georges Gardarin, and Anthony Tomasic. Leveraging Mediator Cost Models with Heterogeneous Data Sources. In *ICDE*, 1998.

[RH05] Naveen Reddy and Jayant R. Haritsa. Analyzing Plan Diagrams of Database Query Optimizers. In *VLDB*, 2005.

[ROH99] Mary Tork Roth, Fatma Ozcan, and Laura M. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *VLDB*, 1999.

[RZL04] Amira Rahal, Qiang Zhu, and Per-Åke Larson. Evolutionary techniques for updating query cost models in a dynamic multidatabase environment. *VLDB J.*, 13, 2004.

[SCPP07] Hyeonju Seol, Jeewon Choi, Gwangman Park, and Yongtae Park. A framework for benchmarking service process using data envelopment analysis and decision tree. *Expert Syst. Appl.*, 32, 2007.

[Sto02] Michael Stonebraker. Too Much Middleware. *SIGMOD Record*, 31(1), 2002.

[SW97] Markus Sinnwell and Gerhard Weikum. A Cost-Model-Based Online Method for Distributed Caching. In *ICDE*, 1997.