Ausnutzung von Restriktionen zur Verbesserung des Deployment-Vorgangs des Verteilten Datenstromverarbeitungssystems NexusDS

Nazario Cipriani
Carlos Lübbe

Universität Stuttgart, Institute of Parallel and Distributed Systems,
Universitätsstraße 38, 70569 Stuttgart, Germany
{cipriano | luebbecs}@ipvs.uni-stuttgart.de

Abstract: Kontextsensitive Informationssysteme verarbeiten häufig Daten der näheren Umgebung, die mit Hilfe von Sensoren kontinuierlich erhoben werden. Für die Verarbeitung kontinuierlicher Datenströme können Datenstromverarbeitungssysteme eingesetzt werden. Jedoch müssen diese im Bereich der Kontextdatenverarbeitung mit einem heterogenen Umfeld zurechtkommen. Je nach technischer Ausstattung der physischen Umgebung sollten sie sich an die dort geltenden Bedingungen bzw. Restriktionen anpassen. Für verteilte Datenstromverarbeitungssysteme ist die Anfrageverteilung in einem solchen Umfeld eine besondere Herausforderung, denn das System muss Restriktionen auf verschiedenen Ebenen berücksichtigen. So könnte ein Teil der Anfrage gewisse Anforderungen an seine Ausführungsumgebung haben, wie spezialisierte Hardware, oder es könnte aus Sicherheitsgründen notwendig sein, die Anfrage oder einen Anfrageteil in einer sicheren Ausführungsumgebung auszuführen.

In diesem Papier klassifizieren wir Restriktionen, die auf verschiedenen Ebenen der Anfrageverarbeitung des Systems vorkommen können. Überdies stellen wir ein Konzept zur Modellierung der Restriktionsklassen vor und zeigen wie diese in der Anfrageverarbeitung des verteilten Datenstromverarbeitungssystems *NexusDS* berücksichtigt werden.

1 Einleitung

Kontextsensitive Informationssysteme beziehen häufig die physische Umgebung mit in die Anwendungslogik ein. Um Zustände der realen Welt erfassen zu können, verwenden sie Sensoren, die kontinuierliche Datenströme produzieren. Zur effizienten Verarbeitung kontinuierlicher Datenströme wurden verteilte Datenstromverarbeitungssysteme entwickelt, wie [KCC+03, XECA07, AAB+05, KSKR05, SLJR05, GAW+08]. Diese Systeme bieten dem Benutzer eine deklarative Anfragesprache und sehen Möglichkeiten vor, Anfragen in einem verteilten Umfeld auszuführen. Hierzu verteilt ein solches System in Operatoren gekapselte Verarbeitungslogik auf unterschiedlichen Rechenknoten, die miteinander kommunizieren und dadurch Zwischenergebnisse der Verarbeitung austauschen.

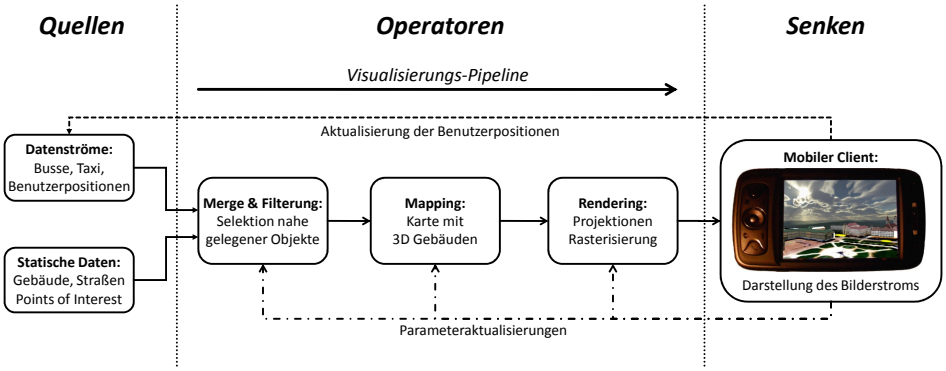

Abbildung 1: Ortsbezogene und interaktive Visualisierungs-Pipeline

Eine Anfrage eines Benutzers, könnte zum Beispiel wie folgt umgangssprachlich formuliert werden: *Ausgehend von meiner aktuellen Position, soll eine frei rotierbare 3D-Karte nahe gelegener Objekte wie Gebäude oder Straßen gezeichnet werden. Zusätzlich sollen alle in Reichweite liegenden Busse und Taxen mit ihrer aktuellen Position sowie besondere (meine Umgebung betreffende) "Points of Interest"(POIs) eingezeichnet werden*. Die Anfrage beschreibt einen Visualisierungsprozess, der abstrakte Informationen, wie Gebäude, Straßen, POIs, oder aktuelle Buspositionen, in visuell wahrnehmbare Bilder transformiert. Der Visualisierungsprozess selbst wird nach [HM90, ZWRI04] als eine Folge Schritten angesehen, nämlich das Filtering, das Mapping und das Rendering. Das Filtering betrifft die Auswahl der zu visualisierenden Daten, das Mapping beschreibt die Abbildung der Daten auf abstrakte visuelle Repräsentationen, wie Geometriedaten, und das Rendering transformiert diese Daten in visuelle Informationen. Die Andordnung dieser Schritte wird Visualisierungs-Pipeline genannt. Die einzelnen Schritte der in Abbildung 1 dargestellten Visualisierungs-Pipeline lassen sich hervorragend als Operatoren in einem Datenstromverarbeitungssystem realisieren. Daher ist es lohnenswert, die unterschiedlichen Datenströme mit einem Datenstromverarbeitungssystem zu verarbeiten, da ein solches System bereits die notwendige Grundfunktionalität mitbringt.

Das Ergebnis der in Abbildung 1 Visualisierungs-Pipeline ist ein Ausgabestrom in Form einer Sequenz von Bildern, die dann von der clientseitig ausgeführten Anwendung visualisiert wird. Diese Bilder können auch auf (aus Performance-Sicht) schwachen Endgeräten visualisiert werden, da die aufwendigen Berechnungsoperationen, wie das Rendern der Szene, auf einem entfernten Rechner durchgeführt werden und der Client lediglich das Ergebnis in Form einer Bildsequenz darstellen muss. Der Endanwender kann auch Einfluss auf die darzustellende Szene nehmen, indem er über die Parameteraktualisierungen mit den System interagiert (wie in Abbildung 1 dargestellt). Dadurch kann er beispielsweise beliebige Rotationen oder Translationen der aktuellen Szene vornehmen, die sich dann in der Folge auch in den erhaltenen Bildern widerspiegelt.

Das Ergebnis der Visualisierung könnte dem Benutzer abhängig von seinem aktuellen Aufenthaltsort auf verschieden Endgeräten präsentiert werden. Befindet sich ein größeres Anzeigegerät in der Nähe, könnte statt eines mobilen Endgerätes dieses verwendet werden. Die verschiedenen Geräte, die abhängig vom Kontext des Benutzers zur Verfügung stehen, unterscheiden sich in ihren technischen Eigenschaften und bringen somit gerätespezifische Restriktionen mit sich, die bei der Anfrageverarbeitung beachtet werden müssen. Möchte der Benutzer sichergehen, dass die Information seines derzeitigenn Aufenthaltsortes nicht in die falschen Hände gelangt, so muss das Datenstromverarbeitungssystem dafür sorgen, dass die Anfrage in einer sicheren Umgebung verarbeitet wird. Das Beispiel zeigt auf, dass Restriktionen auf verschiedenen Ebenen Anfrageverarbeitung auftreten können.

Bisherige Datenstromverarbeitungssysteme gehen von einer homogenen Systemtopologie aus, weshalb die Restriktionen, die in einem heterogenen Umfeld auftreten können, unberücksichtigt bleiben. Ferner wird in bisherigen Systemen keine nähere Unterscheidung der im System vorhandenen Rollen, wie beispielsweise die Rolle des Entwicklers oder des Benutzers, gemacht. Das Ziel dieser Arbeit ist es, die im heterogenen Umfeld auftretenden, rollenabhängigen, Restriktionen zu identifizieren und zu klassifizieren. Wir beleuchten die — gemäß den vorgestellten Restriktionsklassen — modifizierte Anfrageverarbeitung des verteilten Datenstromverarbeitungssystems, *NexusDS* [CNG+09, CEB+09], welches Restriktionen auf verschiedenen Ebenen der Anfrageverarbeitung beachtet. Diese Eigenschaft erhöht die Nutzbarkeit des Systems in einem heterogenen Umfeld und damit auch die Attraktivität für Anwendungen, die von ihrem Ausführungskontext abhängen.

Der Rest des Papiers gliedert sich wie folgt: In Kapitel 2 werden die Restriktionen klassifiziert, woraufhin in Kapitel 3 kurz auf die Architektur von NexusDS eingegangen wird. In Kapitel 4 stellen wir die Modellierung der Restriktionen und deren Integration in die Anfrageverarbeitung des Systems vor. Abgeschlossen wird das Papier mit Kapitel 5, in dem eine kurze Zusammenfassung und ein Ausblick auf zukünftige Forschungsthemen erfolgt.

2 Klassifizierung der Restriktionen

Das in Kapitel 1 vorgestellte, nicht triviale Beispiel zeigt deutlich, dass es unterschiedlichste Anforderungen an der Ausführung einer Anfrage gibt. Beispielsweise wird der *Render*-Operator typischerweise für eine GPU (Graphic Processing Unit) programmiert, da dieser sich auf einer solchen Hardware-Architektur am effizientesten umsetzen lässt. Alternativ gibt es bereits viele, ursprünglich für die CPU konzipierte, Algorithmen bereits als GPU-basierte Algorithmen [OLG+05, GWH05]. Genauso könnte man Operatoren auf einem FPGA (Field Programmable Gate Array) umsetzen, wie beispielsweise eine komplexe Funktion zur Interpolation von Werten.

Um diese exemplarisch aufgeführten Anforderungen berücksichtigen zu können, muss das System die Definition von Restriktionen unterstützen. Restriktionen sind also für die korrekte Verteilung und Ausführung der Anfrage von zentraler Bedeutung, da vom Datenstromverarbeitungssystem für einen Operator die geeignete Laufzeitumgebung gewählt werden muss.

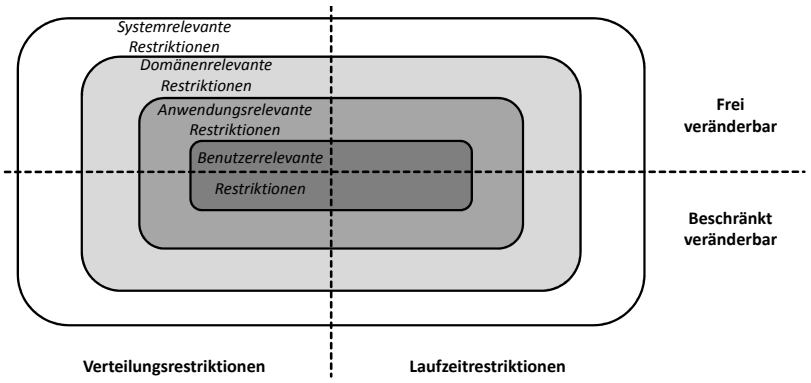

Abbildung 2: Aufteilung des Restriktionsraums

Wie in Abbildung 2 zu sehen ist, lässt sich der Restriktionsraum[1] in solchen Datenstromverarbeitungssystemen horizontal und vertikal partitionieren. Horizontal lässt sich der Raum in *Verteilungs-* und *Lauszeitrestriktionen* aufteilen, während es sich in der Vertikalen in *Frei* und *Beschränkt veränderbare Restriktionen* einteilen lässt. Im folgenden wird auf die jeweiligen Partitionen eingegangen:

1. **Verteilungsrestriktionen**: Verteilungsrestriktionen dienen dazu, das Verteilungverhalten des Systems zu beeinflussen. Mit diesem Mechanismus kann beeinflusst werden, welche Menge von Rechenknoten für die Ausführung eines Operators in Frage kommen. Eine mögliche Restriktion könnte beispielsweise sein, dass nur Knoten mit einer zertifizierte Umgebung für die Ausführung eines bestimmten Operators in Frage kommen.

2. **Laufzeitrestriktionen**: Die Laufzeitrestriktionen wirken sich auf das Laufzeitverhalten der Operatoren aus. Diese Art der Restriktion hat einen direkten Einfluss auf den Ressourcenverbrauch des Operators. So hat beispielsweise die Angabe der Zielauflösung, in der die Szenerie berechnet werden soll, direkten Einfluss auf den Ressourcenverbrauch des Render-Operators aus Abbildung 1.

3. **Frei veränderbare Restriktionen**: Frei veränderbare Restriktionen unterliegen keiner weiteren Einschränkung bezüglich möglicher erlaubter Werte. Beispielsweise könnte eine solche Restriktion lauten, dass ein Render-Operator mit einer bestimmten Auflösung arbeiten soll (Laufzeitrestriktion).

4. **Beschränkt veränderbare Restriktionen**: Beschränkt veränderbare Restriktionen charakterisieren sich dadurch, dass diese nur in einem zuvor festgelegten Rahmen verändern werden dürfen. Hier muss also zusätzlich zum Test der Verträglichkeit des erwarteten und der gesetzten Wertes überprüft werden, ob sich die Restriktionen im definierten Intervall befinden. Zum Beispiel könnte es eine Beschränkung

[1] Der Restriktionsraum beschreibt hier die Menge aller möglichen Restriktionen.

bzgl. der Auflösung eines Render-Operators geben, die in der Liste von möglichen Auflösungen enthalten sein muss. Ein weiteres Beispiel könnte lauten, dass sich nur Rechenknoten für die Ausführung dieses Operators qualifizieren, die in einer bestimmten Liste von Rechenknoten stehen.

Die Tatsache, ob eine Restriktion frei oder beschränkt Veränderbar ist, beziehen sich immer auf die Verteilungs- und Laufzeitrestriktionen. Verteilungs- und Laufzeitrestriktionen stellen somit die grundlegenden Restriktionsarten dar und werden feiner in frei und beschränkt veränderbare Restriktionen aufgeteilt.

Der bisher definierte Restriktionsraum lässt sich weiterhin bezüglich der verschiedenen Rollen in einem Datenstromverarbeitungssystem aufteilen (vergleiche Abbildung 2), nämlich aufgrund der Rolle des **Systems**, der **Domäne**, der **Anwendung** und des **Benutzers**. Im folgenden wird auf die unterschiedlichen Rollen näher eingegangen:

1. **Systemrelevante Restriktionen**: Systemrelevante Restriktionen werden von den jeweiligen Entwicklern bei der Realisierung der Operatoren und Dienste definiert und legen den Rahmen fest, in dem sich domänen, anwendungs oder benutzerrelevante Restriktionen abspielen können (siehe Abbildung 2). Ein Beispiel hierfür wäre ein Operator, der eine bestimmte GPU voraussetzt.

2. **Domänenrelevante Restriktionen**: Domänenrelevante Restriktionen dienen dazu, besonderes Wissen aus den Anwendungsdomänen in das System einzubringen. Das hat zur Folge, dass sich das System gezielter auf die besonderen Anforderungen bestimmter Anwendungsdomänen anpassen lässt. Wie in Abbildung 2 dargestellt, ist diese Restriktionsklasse komplett in der Restriktionsklasse der Systemrelevanten Restriktionen enthalten. Das bedeutet insbesondere, dass in dieser Klasse keine zuvor festgelegten systemrelevante Restriktionen umgangen werden können. Beispielsweise könnte eine Restriktion in dieser Klasse lauten, dass nur eine bestimmte Menge an Rechenknoten (mit einer GPU) die Daten verarbeiten sollen, die ebenfalls über einen Hauptspeicher einer bestimmten Größe verfügen, da die zu verarbeitenden Kontext-Daten ein hohes Datenvolumen aufweisen.

3. **Anwendungsrelevante Restriktionen**: In einer Anwendung selbst können wiederum Restriktionen definiert sein. Diese Restriktionsklasse ist wiederum in der Restriktionsklasse der domänenrelevanten Restriktionen enthalten. Eine Anwendung könnte beispielsweise fordern, dass der GPU-basierte Operator auf einem bestimmten Rechenknoten ausgeführt wird, sofern er nicht die system- und domänenrelevanten Restriktionen verletzt. Das könnte den Vorteil haben, dass sich die Latenzzeiten von der Eingabe einer Interaktion zur Umsetzung minimieren ließen, welche das Rendering der Szene betreffen, wie beispielsweise die Translation oder Rotation der Szenerie.

4. **Benutzerrelevante Restriktionen**: Diese bilden den kleinsten Teil der möglichen Restriktionen (siehe Abbildung 2). Die möglichen Restriktionen hier werden im wesentlichen von der Anwendung diktiert und können somit auch nur in diesem Rahmen verändert werden. Hierbei ist eine grafische Benutzeroberfläche vorstellbar, die

dem Benutzer über einem Eigenschaften-Fenster verschiedene Möglichkeiten bietet, die Anwendung (und somit auch indirekt die Anfrageverarbeitung) nach seinem Geschmack zu justieren. Beispielsweise könnte so der Benutzer Einfluss auf die mögliche Ausführungsumgebung haben, indem er lediglich Rechenknoten für die Verarbeitung der Kontext-Daten zulässt, die ihm eine sichere Umgebung, in Form einer kryptografischen Verschlüsselung der Kontext-Daten, bieten.

Systemrelevante Restriktionen können nicht übergangen werden, da diese das korrekte Funktionieren des Systems garantieren. Domänen-, anwendungs- oder benutzerrelevante Restriktionen sind Ergänzungen zu den systemrelevanten Restriktionen und haben untereinander, wie in Abbildung 2 dargestellt, eine Teilmengenbeziehung.

In Folgenden wird ein kurzer Überblick über NexusDS gegeben. Im Anschluss wird aufgezeigt, wie die oben beschriebene Klassifikation in das NexusDS System implementiert werden kann, um Einfluss auf die Verteilung und Ausführung der Anfragen in NexusDS zu nehmen.

3 NexusDS Überblick

NexusDS [CNG+09, CEB+09], wie in Abbildung 3 auf der linken Seite dargestellt, ist in einer Schichtenarchitektur aufgebaut. Innerhalb einer Schicht werden Operatoren oder Dienste definiert. Ein *Operator* verarbeitet Daten und integriert individuelle Verarbeitungslogik in das System. Der Operator arbeitet Push-basiert, er erhält also Daten in Form eines Datenstroms, verarbeitet diesen und leitet das Ergebnis weiter. Ein *Dienst* hingegen arbeitet Pull-basiert und wird benötigt, um eine Interaktion zwischen den Anwendungen und dem System zu ermöglichen. Ein Beispiel für einen solchen Dienst wäre ein dedizierter Visualisierungs-Dienst, der einen auf die Domäne der Visualisierung zugeschnittenen Funktionsumfang bietet.

In der untersten Schicht befinden sich die tatsächlichen physischen Rechenknoten, die an einer Verarbeitung teilnehmen. Diese entsprechen einem bestimmten Knoten in einer anderen Schicht. In der *Communication and Monitoring* Schicht werden Dienste zur Verfügung gestellt, die zur Kommunikation zwischen den Rechenknoten genutzt werden können, sowie ein Monitoring Dienst, der für die Überwachung der Laufzeiteigenschaften (CPU-Auslastung, Füllstand der Warteschlange usw.) zuständig ist. Die darüber liegende *Nexus Core* Schicht definiert die Kerndienste des NexusDS. Hier findet sich unter anderen der *Core Query Service*, der NPGM-Anfragen (siehe Kapitel 4.1.3) verarbeitet, der *Operator Repository Service*, das zur Integration von individuellen Operatoren benutzt werden kann, oder der *Operator Execution Service*, der eine Ausführungsumgebung für Operatoren auf den Rechenknoten anbietet. Die *Nexus Domain Extensions* Schicht enthält Dienste und Operatoren, die in logisch zusammenhängende Dömanen gruppiert werden. Die zugehörige Domäne für unser Anwendungsbeispiel ist die Visualisierungsdomäne, die den *Visualization Pipeline Service* und weitere zur Realisierung einer Visualisierungs-Pipeline benötigten Operatoren enthält. Die *Nexus Application* Schicht erlaubt es, anwendungsspezifische Logik in das NexusDS auszugliedern. Dies ermöglicht Szenarien, bei denen ein

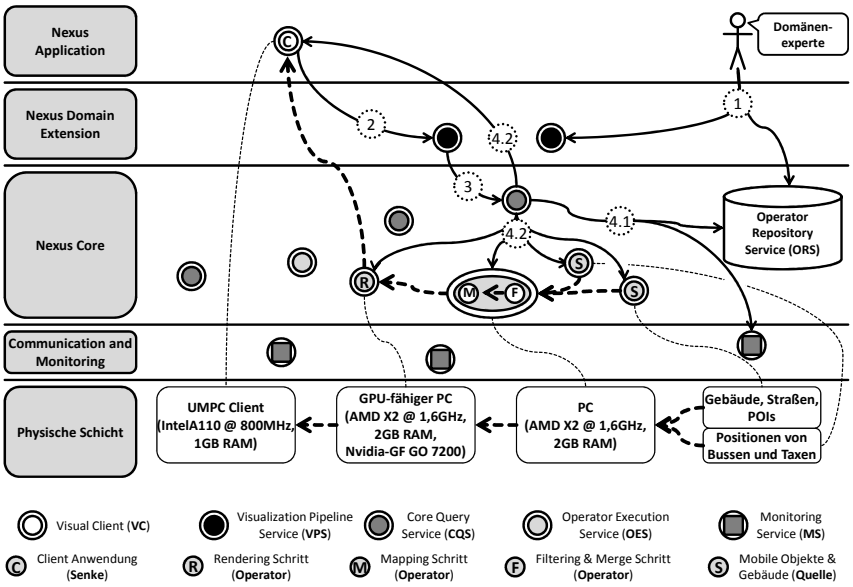

Abbildung 3: Ausführungsmodell für die Visualisierungs-Pipeline

ressourcenarmer Client komplexe Berechnungen von der NexusDS-Infrastruktur durchführen lässt.

Der Ablauf der Visualisierungsanfrage unseres Beispiels ist auf der rechten Seite von Abbildung 3 skizziert. Der interessierte Leser sei an dieser Stelle aus Platzgründen auf [CEB+09] verwiesen. Anhand des rechten Teils der Abbildung 3 wird in Kapitel 4.2 die modifizierte Anfrageverarbeitung in NexusDS erläutert.

4 Integration von Restriktionen in NexusDS

Für die korrekte Anfrageverarbeitung spielen Restriktionenin einem heterogenen Datenstromverarbeitungssystem eine entscheidende Rolle. Dies wird umso wichtiger, wenn das System flexibel im Bezug auf das Einsatzgebiet sein soll. Wie in Kapitel 2 aufgezeigt, können die unterschiedlichen Rollen unterschiedliche Anforderungen an die Ausführungsumgebung eines Datenstromverarbeitungssystems haben. Diese Tatsache macht es zu einer Notwendigkeit, dem System die notwendigen Anforderungen mitteilen zu können. Im Folgenden erläutern wir, wie diese Restriktionen in NexusDS Berücksichtigung finden.

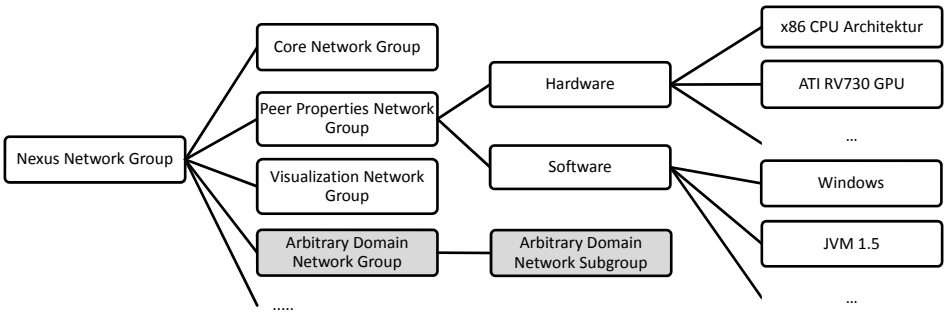

Abbildung 4: Netzwerkgruppen in NexusDS

4.1 Modellierung der Restriktionen

Die grundlegende Modellierung der Restriktionen wird in NexusDS durch zwei Konzepte bewerkstelligt: *Netzwerkgruppen* und *Operatorbeschreibungen*. Netzwerkgruppen dienen dazu, die Rechenknoten in Kompatibilitätsklassen einzuteilen. Operatorbeschreibungen dienen dazu, den Operator zu beschreiben. Die Kombination aus Anforderungen an die Ausführungsumgebung und die Einteilung der Rechenknoten in Kompatibilitätsklassen ermöglichen es, zu einem, in einer Anfrage definierten, Operator einen entsprechenden Rechenknoten zu finden.

4.1.1 Netzwerkgruppen

Netzwerkgruppen spannen in NexusDS den Restriktionsraum für die Verteilung auf (vgl. Abbildung 2). Netzwerkgruppen funktionieren analog zum JXTA-Konzept der Network Groups [TAA+03]. Netzwerkgruppen segmentieren das Netz in unterschiedliche, nicht zwingend disjunkte Regionen. Ein Rechenknoten kann mehreren Netzwerkgruppen gleichzeitig angehören. Die Semantik nach der gruppiert wird, ist dabei nicht vorgeschrieben. So könnte zum Beispiel eine Segmentierung nach geographischen Gesichtspunkten zur Latenzminimierung erfolgen, ähnlich der hierarchischen Struktur, die sich für das Internet herausgebildet hat. Darüber hinaus lässt sich das Konzept jedoch auch nutzen, um den Wirkungsbereich von Diensten oder Operatoren einzuschränken. Dies ist insbesondere dann wichtig, wenn sichergestellt werden muss, dass ein Dienst oder ein Operator in einer gewissen vertrauten Umgebung abläuft, um beispielsweise den Missbrauch schützenswerter Information zu verhindern. Weiterhin können Netzwerkgruppen helfen, eine initiale Anfrageverteilung zu finden, indem eine anfängliche Kandidatenliste von Rechenknoten auf Basis der Gruppen, der sie angehören, gebildet wird. Werden die Netzwerkgruppen auf der Basis von Hardware-Eigenschaften von Rechenknoten gebildet, so kann dies ebenfalls bei der Suche nach geeigneten Kandidatenknoten ausgenutzt werden.

Abbildung 4 zeigt die hierarchische Struktur von Netzwerkgruppen. Der Wurzelknoten wird durch die *Nexus Network Group* repräsentiert. Dieser Gruppe gehören alle Dienste und Operatoren und Anwendungen an, die in NexusDS verfügbar sind. Darüber hinaus gibt es unterschiedliche Untergruppen. Die Untergruppe, die sämtliche NexusDS-Kernfunktionalität bündelt, ist die *Core Network Group*. Zusätzlich gibt es eine *Peer Properties Network Group*, welche die einzelnen Rechenknoten entsprechend ihren Eigenschaften einteilt. Daneben kann es beliebige zusätzliche Netzwerkgruppen geben. Als Beispiel ist hier noch die *Visualization Network Group* aufgeführt. Die bisher genannten Netzwerkgruppen sind öffentlich, was bedeutet, dass prinzipiell jeder Rechenknoten diesen Netzwerkgruppen beitreten kann. Daneben gibt es auch die Möglichkeit, die Netzwerkgruppen zu sichern (in der Abbildung als graue Netzwerkgruppen dargestellt). Teilnehmer, die diesen Gruppen beitreten wollen, müssen sich dann authentifizieren bevor sie der Gruppe beitreten können und somit Zugriff auf die Ressourcen erhalten können.

Die Verteilungsrestriktionen werden durch die Netzwerkgruppen unterhalb der *Peer Properties Network Group* beschränkt. Hier werden die einzelnen Rechenknoten entsprechend ihren Fähigkeiten in unterschiedliche Netzwerkgruppen unterteilt. Beispielsweise werden alle Rechenknoten, die eine *x86 CPU* haben in die Netzwerkgruppe *x86 CPU Architecture* aufgenommen. Dadurch kann gezielt auf Rechenknoten zugegriffen werden, die das entsprechende Kriterium *x86 CPU* erfüllen. Genau so verhält es sich beispielsweise mit Angaben über den maximal verfügbaren Speicher. Die Netzwerkgruppen können jedoch nur statische Eigenschaften der Rechenknoten beschreiben, d.h. die aktuelle Speicherbelegung muss separat mithilfe eines Monitoring-Dienstes erfasst werden. Das Konzept der Netzwerkgruppen bietet folglich nur die Möglichkeit, eine Vorauswahl der Rechenknoten zu treffen, die einer näheren Untersuchung unterzogen werden müssen.

4.1.2 Operatorbeschreibungen

Operatorbeschreibungen, genauer die Angabe der Operatorparameter, spannen in NexusDS den Restriktionsraum für die Laufzeit auf (vgl. Abbildung 2). Operatorbeschreibungen dienen allgemein dazu, einen Operator zu charakterisieren und stellen dessen Metadaten dar. Zusätzlich kann durch dieses Konzept ein Entwickler dem System Randbedingungen für das korrekte Funktionieren des Operators bekannt machen. Die Operatorbeschreibung eines einzelnen Operators gliedert sich in drei Teile, die im Folgenden näher erläutert werden:

1. **Operatordeskriptor**: Der *Operatordeskriptor* dient dazu, den Operator selbst zu charakterisieren. Hier wird festgehalten, wie viele Ein- und Ausgänge der Operator hat und welche Parameter definiert sind. Ferner wird für jeden Ein- und Ausgang sowie für jeden Parameter der entsprechende Datentyp, der erwartet bzw. geliefert wird, angegeben. Das gewährleistet die Inter-Operator-Komptabilität, da nur Operatoren mit kompatiblen Datentypen an den Ein- und, Ausgängen miteinander verknüpft werden können. Der Entwickler kann für die Parameter zusätzlich noch ein Intervall oder eine Liste von erlaubten Werten definieren, die der jeweilige Parameter annehmen kann, was dann bedeutet es handelt sich um **beschränkt veränder-**

bare Laufzeitrestriktionen. Fehlt die Angabe der Intervalle oder Listen, sind die **Laufzeitrestriktionen frei veränderbar**.

2. **Operatoranforderungen**: *Operatoranforderungen* haben legen fest, welche Mindestanforderungen der Operator an die Ausführungsumgebung stellt, in der er ausgeführt werden soll. Es handelt sich hierbei im wesentlichen um eine Sammlung von Schlüssel-Wert-Paaren, die denen der Netzwerkgruppen entsprechen müssen, beispielsweise *Software="Windows"*. Diese Informationen werden von der Anfrageverarbeitung hergenommen, um eine Vorauswahl der Rechenknoten zu treffen, die an einer anschließenden Verteilung teilnehmen sollen. Durch Angabe der Operatoranforderungen wird der Resitiktionsraum der **beschränkt veränderbaren Verteilungsrestriktionen** definiert. Fehlt die Angabe der Operatoranforderungen, befinden wir uns in der Klasse der **frei veränderbaren Verteilungsrestriktionen**. Zudem kann hier die Ausführung des Operators auf bestimmte Rechenknoten beschränkt werden (ein sog. *angehefteter Operator*).

3. **Operatorvoreinstellungen**: Die *Operatorvoreinstellungen* werden dazu verwendet, um Wertekombinationen für die Parameter des Operators zu definieren. Hier kann man beispielsweise häufig gebrauchte Wertekombinationen ablegen und zu einem späteren Zeitpunkt wieder laden. Die Angabe von Standardwerten garantiert, dass der Operator auch korrekt funktioniert, falls vom Anfragesteller keine weiteren Angaben zu den Parametern des Operators gemacht werden.

4.1.3 Nexus Plan Graph Model

Das *Nexus Plan Graph Model (NPGM)* dient dazu, Restriktionen in die Anfrageverarbeitung mit einfließen zu lassen. NPGM ist ein Anfragemodell zur Orchestrierung von Datanflussgraphen. Es ähnelt dem *Boxes and Arrows* Ansatz, wie in [ACc+03] beschrieben. Im Gegensatz zu den bisherigen Ansätzen unterstützt NPGM zusätzlich den Anfragegraphen mit Annotationen anzureichern, womit sich die Verteilung und Ausführung beeinflussen lässt. NPGM ermöglicht somit logische Anfragepläne unter Berücksichtigung der in Kapitel 2 genannten Verteilungs- sowie Laufzeitrestriktionen zu modellieren.

Wie in Abbildung 5 zu sehen ist, besteht ein NPGM-Anfragegraph aus einer Menge von NPGM-Boxen, die untereinander verbunden sind und somit eine Verarbeitungspipeline bilden. Boxen sind hierbei entweder eine Quelle, Senke oder ein Operator. NPGM-Boxen haben eine beliebige Anzahl an Ein- und Ausgängen. Die genaue Anzahl wird durch die Operatordeskriptoren festgelegt. Diese legen auch fest, welche Datentypen von den NPGM-Boxen geliefert und erwartet werden. Das hat zur Folge, dass nur Ein- und Ausgänge der NPGM-Boxen untereinander verknüpft werden können, deren Datentyp kompatibel sind. Dieser Sachverhalt ist durch die in unterschiedlichen Grautönen gezeichneten Ein- und Ausgängen dargestellt.

Durch die Angabe von Restriktionen in einem NPGM-Graphen kann auf die Verteilung und auf die Laufzeit des Operators Einfluss genommen werden. Diese Zweiteilung ist in Abbildung 5 dargestellt.

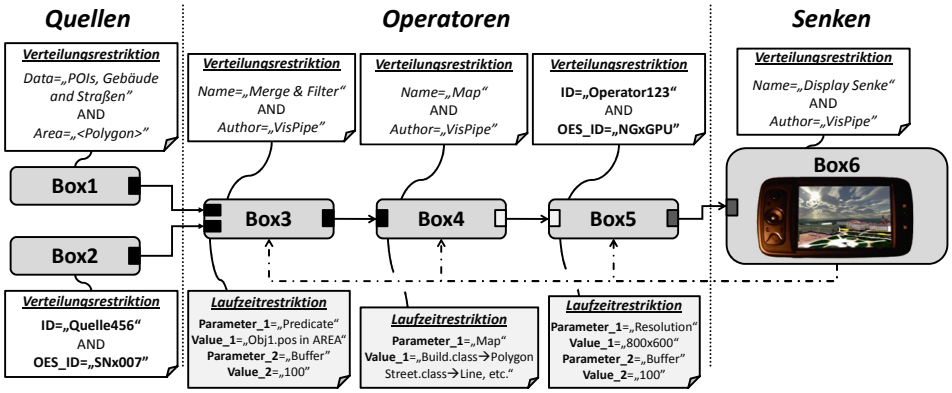

Abbildung 5: Eine NPGM Anfrage in NexusDS

Verteilungsrestriktionen können auf unterschiedlichen Ebenen gesetzt werden: Von einem Benutzer, einer Anwendung, einer Domäne oder dem System selbst. Selbiges gilt für Laufzeitrestriktionen. Somit bildet die Angabe von Restriktionen innerhalb des NPGM-Graphen für die verschiedenen Rollen (Benutzer, Anwendung, Domäne und System) eine universelle Möglichkeit, auf die Verteilung und Laufzeit der Anfrage Einfluss zu nehmen. Die Zweiteilung in Verteilungs- und Laufzeitrestriktionen ist in Abbildung 5 dargestellt.

Verteilungsrestriktionen dienen dazu, (1) zu einem logischen Operator den entsprechenden physischen Operator zu spezifizieren und (2) den Rechenknoten bzw. die Knotengruppe, auf dem bzw. der Operator ausgeführt werden soll, festlegen zu können. Beide Punkte können entweder spezifisch (in Abbildung 5 **fett** dargestellt) oder variabel (in Abbildung 5 *kursiv* dargestellt) angegeben werden (Punkt 1 ist obligatorisch und muss angegeben werden, wohingegen Punkt 2 optional ist). Im Beispiel oben wird gefordert, dass für die *Box2* die Quelle mit **ID="Quelle456"** auf dem Rechenknoten **OES_ID="SNx007"** ausgeführt wird, was einer spezifischen Angabe entspricht. Dagegen ist eine Angabe wie die für *Box3* variabel, in der gefordert wird, dass der Operator mit *Name="Merge & Filter"* und *Author="VisPipe"* gewählt werden soll. Diese variable Beschreibung muss anschließend von der Anfrageverarbeitung auf eine passende physische Implementierung abgebildet werden. Es kann vorkommen, dass zu einer variablen Verteilungsrestriktion mehrere passende physische Operatoren existieren. In einem solchen Fall muss die Anfrageverarbeitung den passenden finden. Eine nähere Betrachtung würde allerdings den Rahmen dieses Papiers sprengen.

Durch Laufzeitrestriktionen lassen sich Operatoren parametrisieren, womit Einfluss auf die tatsächliche Ausführung des Operators genommen werden kann. Laufzeitrestriktionen sind optional und müssen nicht angegeben werden. Falls diese Angabe fehlt, werden vordefinierte Parameterwerte geladen (siehe Operatorvoreinstellungen in Kapitel 4.1.2). Es gibt allgemeingültige Laufzeitrestriktionen für Operatoren, wie beispielsweise die Angabe *Parameter_2="Buffer"* mit dem zugehörigen Wert *Value_2="100"*. Diese Angabe bezieht sich auf die maximale Größe des Puffers und somit auf das Sichtfenster auf den

Datenstrom und sind bei allen Operatoren verfügbar. Laufzeitrestriktionen können aber auch sehr operatorspezifisch sein, wie bei *Box5* zu sehen ist. Hier gibt es ein Parameter *Parameter_1="Resolution"* mit den zugehörigen Wert *Value_1="800x600"*. Diese Angabe definiert die Viewportgröße des Render-Operators auf 800 mal 600 Bildpunkte.

4.2 Anfrageverarbeitung in NexusDS unter Berücksichtigung von Restriktionen

Die Anfrageverarbeitung in NexusDS muss leicht modifiziert werden, um die angegebenen Restriktionen berücksichtigen zu können. In folgenden wird Bezug auf Abbildung 3 genommen, um die Anfrageverarbeitung zu erläutern.

Zunächst definiert ein Domänenexperte eine Domänenerweiterung, in unserem Beispiel den Visualisierungsdienst und die entsprechenden Operatoren *Render*, *Map* und *Filter & Merge* (1). In einem weiteren Schritt wurde eine Anwendung von einem Entwickler (nicht zwangsläufig der Domänenexperte) erstellt, welche die oben beschriebenen domänenspezifischen Komponenten nutzt. Nun kann eine Anwendung die Anfrage in einer domänenspezifischen Anfragesprache an diesen Dienst senden (2). In dieser Anfrage können anwendungsrelevante und auch benutzerrelevante Restriktionen enthalten sein sofern es die Anwendung zulässt. Beispielsweise könnte hier angegeben sein, dass die Ausführung nur innerhalb einer bestimmten, vom Anwender als sicher erachteten Menge von Rechnenknoten, erfolgen darf. Als Antwort erhält die Anwendung eine Kennung der Anfrage in Form einer ID, mit der die Anfrage später verändert oder terminiert werden kann.

Der Visualisierungsdienst bildet die Anfrage auf die interne, aus logischen Operatoren zusammengesetzte Anfragesprache NPGM (siehe Kapitel 4.1.3) ab. Dabei wird zum Einen überprüft, ob die benutzer- oder anwendungsrelevanten Restriktionen angegeben wurden und ob diese gegen domänenrelevante Restriktionen verstoßen. Es können zusätzlich noch domänenrelevante Restriktionen hinzukommen, wenn eine solche Restriktion innerhalb der Domäne definiert wurde. Beispielsweise könnte eine solche Restriktion lauten, dass die Ausführung auf Knoten beschränkt sein soll, die über einen sehr großen (freien) Hauptspeicher verfügen. Das ist spezielles Wissen aus einer Domäne, beispielsweise der Domäne der Visualisierung, die besagt, dass das hier auftretende Datenvolumen sehr groß werden kann.

Die Anfrage wird nun an den Core Query Service gesendet (3). Mit Hilfe der NPGM-Repräsentation erzeugt der Core Query Service den physischen Anfragegraph, der die physischen Operatoren und Informationen über deren Verteilung enthält. Vor der Erzeugung des physischen Anfragegraphen muss der Core Query Service jedoch zunächst überprüfen, ob zu jedem logischen Operator des NPGM-Graphen eine physische Implementierung vorhanden ist (4.1). Anhand der Operatorbeschreibungen wird überprüft, ob die bisher angegebenen Restriktionen gegen systemrelevante Restriktionen verstoßen. Gegebenenfalls müssen noch fehlenden Restriktionen unter Zuhilfenahme des Operatordeskriptor (Laufzeitrestriktionen) und der Operatoranforderungen (Verteilungsrestriktionen) im NPGM-Graphen ergänzt werden.

Jetzt ist der NPGM-Graph vollständig und es wird in einem nächsten Schritt bestimmt, welche Rechenknoten welchen Teil der Anfrage ausführen sollen. Dies geschieht unter Zuhilfenahme des Monitoring Dienstes. Genauer gesagt werden hier die *Operatoranforderungen* (wie oben beschriebenen ein Teil der Operatorbeschreibungen) mit den Netzwerkgruppen abgeglichen. Hierbei fließt (zu Load-Balancing Zwecken) auch die aktuelle Auslastung der Rechenknoten ein, die vom Monitoring Dienst abgefragt wird.

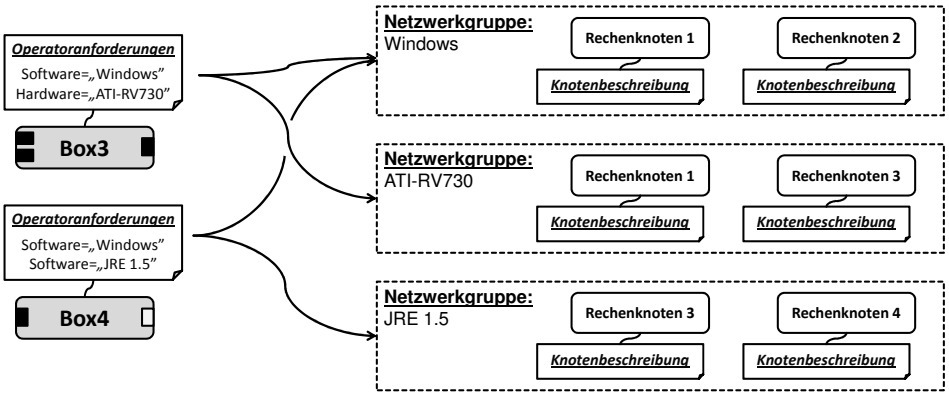

Abbildung 6: Zusammenführung der Konzepte der Netzwerkgruppen und Operatorbeschreibungen

Wie in Abbildung 6 zu sehen ist, wird, bevor die Anfrageverarbeitung starten kann, zunächst nach passenden Kandidaten für die Verteilung der Operatoren gesucht. Das ist exemplarisch für die Boxen 3 und 4 aus dem Beispiel in Abbildung 5 dargestellt. Dazu wird in den entsprechenden Netzwerkgruppen nach passenden Rechenknoten für einen bestimmten Operator gesucht, indem die Operatoranforderungen mit den Netzwerkgruppen abgeglichen werden. Für Box3 wäre das beispielsweise zum einen *Software="Windows"* und zum anderen *Hardware="ATI-RV730"*.

Wurde der Abgleich mit allen Operatoren vorgenommen, stehen ab diesem Zeitpunkt die passenden Rechenknoten für die Auswahl an Operatoren zur Verfügung, die Anfrage kann also verteilt werden. Die Operatoren werden auf den so ausgewählten Rechenknoten ausgeführt, bzw. eine Fehlermeldung zur Anwendung propagiert, falls keine passenden Rechenknoten gefunden wurden (4.2). Sind auf den Rechenknoten die ausgewählten physischen Implementierungen der Operatoren nicht verfügbar, so muss der betreffende Rechenknoten das Operator-Repository kontaktieren und den fehlenden Operator nachladen (dieser Vorgang wurde aus Übersichtsgründen in der Abbildung verschwiegen). Das Resultat der Visualisierungsanfrage ist ein kontinuierlicher Datenstrom, der von den Quellen (S) über die Operatoren Filtering (F), Mapping (M) und Rendering (R) solange zum Client (C) gesendet wird, bis die Anfrage von der Anwendung terminiert wird, die vorab festgelegte Lebensdauer der Anfrage abgelaufen ist, oder über einen bestimmten Zeitraum keine Daten mehr empfangen werden.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Ansatz zur Klassifizierung von Restriktionen vorgestellt, mit dem die Arbeitsweise des Anfrageverarbeitungssystems von NexusDS in verschiedenen Granularitätsabstufungen gesteuert werden kann. Hierbei wurde die Aufteilung des Restriktionsraums in vier Partitionen vorgeschlagen: *Frei veränderbare Verteilungsrestriktionen*, *beschränkt veränderbare Verteilungsrestriktionen*, *frei veränderbare Laufzeitrestriktionen* (und *beschränkt veränderbare Laufzeitrestriktionen*. Jede dieser Partition kann nun systemspezifisch, domänenspezifisch, anwendungsspezifisch oder benutzerspezifisch manipuliert werden. In einem hochdynamischen und heterogenen Umfeld ist es von Nutzen, auf die Anfrageverteilung in verschiedenen Granularitätsabstufungen Einfluss nehmen zu können. Zum einen kann dadurch die Menge zu untersuchenden Rechenknoten verringert werden, so dass die Komplexität des Verteilungsproblems verringert werden kann. Zum anderen kann so externe Information bei der Verteilung berücksichtigt werden. Damit lassen sich Relationen modellieren, die in einem herkömmlichen Anfrageverarbeitungssystem unberücksichtigt bleiben. Beispiele hierfür sind unter anderem die Berücksichtigung von Vertrauen, oder Modellierung von Inkompatibilitäten zwischen Programmcode und ausführender Hardware-Umgebung.

Mit dem hier beschriebenen Konzept lassen sich statische Restriktionen in die Anfrageverarbeitung integrieren. Während der Laufzeit einer Anfrage könnten diese aber überschritten werden. Definiert ein Operator zum Beispiel eine Mindestmenge an verfügbarem Speicher, so kann die aktuelle Knotenauslastung dazu führen, das diese Menge nicht zur Verfügung gestellt werden kann. Der Fokus aktueller Forschung ist es, die dynamischen Laufzeiteigenschaften der Rechenknoten mit in die Anfrageverarbeitung einzubeziehen, um zu verhindern, dass definierte Restriktionen, während der Laufzeit einer Anfrage überschritten werden.

Danksagung

Diese Arbeit wurde vom Sonderforschungsbereich 627 “Umgebungsmodelle für mobile kontextbezogene Systeme” (Nexus) unterstützt.

Literatur

[AAB+05] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing und Stan Zdonik. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[ACc+03] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul und Stan Zdonik. Aurora: a new

model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[CEB+09] Nazario Cipriani, Mike Eissele, Andreas Brodt, Matthias Grossmann und Bernhard Mitschang. NexusDS: A Flexible and Extensible Middleware for Distributed Stream Processing. To Appear In: *IDEAS '09: Proceedings of the 2008 international symposium on Database engineering & applications*, New York, NY, USA, 2009. ACM.

[CNG+09] Nazario Cipriani, Daniela Nicklas, Matthias Grossmann, Nicola Hönle, Carlos Lübbe und Bernhard Mitschang. Verteilte Datenstromverarbeitung von Sensordaten. *Datenbank-Spektrum*, 9(28):37–43, 2009.

[GAW+08] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu und Myungcheol Doo. SPADE: the system s declarative stream processing engine. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, Seiten 1123–1134, New York, NY, USA, 2008. ACM.

[GWH05] Nolan Goodnight, Rui Wang und Greg Humphreys. Computation on Programmable Graphics Hardware. *IEEE Computer Graphics and Applications*, 25(5):12–15, 2005.

[HM90] R. B. Haber und D. A. McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In B. Schriver, G. M. Nielson und L. J. Rosenblum, Hrsg., *Visualization in Scientific Computing*, Seiten 74–93. IEEE Computer Society Press, 1990.

[KCC+03] Sailesh Krishnamurthy, Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Frederick Reiss und Mehul A. Shah. TelegraphCQ: An Architectural Status Report. *IEEE Data Eng. Bull.*, 26(1):11–18, 2003.

[KSKR05] Richard Kuntschke, Bernhard Stegmaier, Alfons Kemper und Angelika Reiser. StreamGlobe: processing and sharing data streams in grid-based P2P infrastructures. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, Seiten 1259–1262. VLDB Endowment, 2005.

[OLG+05] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn und Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. In *Eurographics 2005, State of the Art Reports*, Seiten 21–51, August 2005.

[SLJR05] Timothy M. Sutherland, Bin Liu, Mariana Jbantova und Elke A. Rundensteiner. D-CAPE: distributed and self-tuned continuous query processing. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, Seiten 217–218, New York, NY, USA, 2005. ACM.

[TAA+03] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean christophe Hugly, Eric Pouyoul und Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Bericht, 2003.

[XECA07] Xiaopeng Xiong, H. G. Elmongui, Xiaoyong Chai und W. G. Aref. Place: A Distributed Spatio-Temporal Data Stream Management System for Moving Objects. In *Mobile Data Management, 2007 International Conference on*, Seiten 44–51, 2007.

[ZWRI04] Mengxia Zhu, Qishi Wu, N.S.V. Rao und S. Iyengar. Adaptive visualization pipeline decomposition and mapping onto computer networks. *Image and Graphics, 2004. Proceedings. Third International Conference on*, Seiten 402–405, Dec. 2004.