Modulares Verteilungskonzept für Datenstrommanagementsysteme

Timo Michelsen, Michael Brand, H.-Jürgen Appelrath

Universität Oldenburg, Department für Informatik
Escherweg 2, 26129 Oldenburg
{timo.michelsen, michael.brand, appelrath}@uni-oldenburg.de

Abstract: Für die Verteilung kontinuierlicher Anfragen in verteilten Datenstrommanagementsytemen (DSMS) gibt es je nach Netzwerk-Architektur und Anwendungsfall unterschiedliche Strategien. Die Festlegung auf eine Strategie ist u. U. nachteilig, besonders wenn sich Netzwerk-Architektur oder Anwendungsfall ändern. In dieser Arbeit wird ein Ansatz für eine flexible und erweiterbare Anfrageverteilung in verteilten DSMSs vorgestellt. Der Ansatz umfasst drei Schritte: (1) Partitionierung, (2) Modifikation und (3) Allokation. Bei der Partitionierung wird eine kontinuierliche Anfrage in disjunkte Teilanfragen zerlegt. Die optionale Modifikation erlaubt es, Mechanismen wie Fragmentierung oder Replikation zu verwenden. Bei der Allokation werden die einzelnen Teilanfragen schließlich Knoten im Netzwerk zugewiesen, um dort ausgeführt zu werden.

Für jeden der drei Schritte können unabhängige Strategien verwendet werden. Dieser modulare Aufbau ermöglicht zum Einen eine individuelle Anfrageverteilung. Zum Anderen können bereits vorhandene Strategien aus anderen Arbeiten und Systemen (z.B. eine Allokationsstrategie) integriert werden. In dieser Arbeit werden für jeden der drei Teilschritte beispielhafte Strategien vorgestellt. Außerdem zeigen zwei Anwendungsbeispiele die Vorteile des vorgestellten, modularen Ansatzes gegenüber einer festen Verteilungsstrategie.

1 Einleitung

In verarbeitenden Systemen ist es häufig notwendig, den persistenten Teil des Systems mehrfach und verteilt vorzuhalten, um Ausfälle kompensieren zu können. In verteilten Datenbankmanagementsystemen (DBMS) werden persistente Daten disjunkt (Fragmentierung) oder redundant (Replikation) auf verschiedene Knoten des Netzwerks verteilt. Anfragen werden einmalig gestellt und greifen ausschließlich auf die Knoten zu, die die betreffenden Daten besitzen.

Betrachtet man allerdings verteilte Datenstrommanagementsysteme (DSMS), so ist eine Verteilung der Datenstromelemente allein nicht zielführend, da diese flüchtig sind. Hier sind die Anfragen persistent, da sie theoretisch kontinuierlich ausgeführt werden. Eine solche, auf mehrere Knoten verteilte, *kontinuierliche Anfrage* wird *verteilte, kontinuierliche Anfrage* genannt. Eine verteilte, kontinuierliche Anfrage befindet sich auf mehreren Knoten, indem die einzelnen Operationen den Knoten zugeordnet und dort ausgeführt

werden (in sogenannten *Teilanfragen*). Zwischenergebnisse der Teilanfragen werden von Knoten zu Knoten gesendet, bis schließlich die Ergebnisse an den Nutzer gesendet werden. Für eine Zerlegung einer kontinuierlichen Anfrage in Teilanfragen gibt es allerdings viele Möglichkeiten (z.B. eine Teilanfrage für die gesamte kontinuierliche Anfrage oder je eine Teilanfrage pro Operation).

Die Zerlegung in Teilanfragen ist nur ein Aspekt, der bereits aufzeigt, dass unterschiedliche Strategien für eine Anfrageverteilung in verteilten DSMSs existieren. Ein weiterer Aspekt ist die Netzwerk-Architektur (bspw. homogene Cluster, Client/Server-Architekturen und Peer-To-Peer (P2P)-Netzwerke). In einem homogenen Cluster verfügen alle Knoten i.d.R. über die gleichen Mengen an Systemressourcen. Daher muss bei einer Zuweisung von (Teil-) Anfragen an einen Knoten in homogenen Clustern keinerlei Heterogenität berücksichtigt werden. Kommt hingegen ein heterogenes P2P-Netzwerk zum Einsatz, kann es vorkommen, dass nicht jeder Knoten jede (Teil-) Anfrage ausführen kann (bspw. aufgrund komplexer Operatoren).

In vielen verteilten DSMSs ist die Anfrageverteilung system-intern auf eine bestimmte Netzwerk-Architektur ausgelegt (z.B. Client/Server). Somit wird es aufwendig, die zugrunde liegende Netzwerk-Architektur nachträglich zu wechseln. Ein Grund für eine solche Einschränkung ist die Tatsache, dass keine optimale Verteilungsstrategie existiert, die für alle Netzwerk-Architekturen und Knoten eingesetzt werden kann. Neben unterschiedlichen Netzwerk-Architekturen können ebenso anwendungsspezifische Kenntnisse des Nutzers die Anfrageverteilung optimieren (z.B. die Identifikation von Operatoren, die eine hohe Systemlast erzeugen). Eine solche manuelle Optimierung ist allerdings nur möglich, wenn der Nutzer die Anfrageverteilung für eine konkrete kontinuierliche Anfrage konfigurieren kann.

Diese Arbeit stellt ein Konzept für eine flexible und erweiterbare Anfrageverteilung in einem verteilten DSMS vor. Dazu wird die Verteilung in drei Phasen unterteilt. Jede Phase bietet eine eindeutige Schnittstelle mit Ein- und Ausgaben, die die Umsetzung mehrerer Strategien ermöglicht. Die Anfrageverteilung kann individuell für die zugrunde liegende Netzwerk-Architektur und den konkreten Anwendungsfall konfiguriert und ggfs. optimiert werden. Außerdem ist es möglich, bereits existierende Strategien aus anderen Quellen zu übernehmen. Es entsteht eine Sammlung an Strategien, die je nach Anwendungsfall individuell kombiniert werden können.

Der Rest der Arbeit ist wie folgt gegliedert: Abschnitt 2 gibt einen Überblick über andere Systeme, die ebenfalls kontinuierliche Anfragen verteilen. Das modulare Verteilungskonzept wird in Abschnitt 3 vorgestellt. Dabei liegt der Fokus auf der Erläuterung des Konzeptes. Auf eine vollständige Übersicht aller zum jetzigen Zeitpunkt verfügbarer Strategien wird in dieser Arbeit verzichtet. Abschnitt 4 beinhaltet den aktuellen Stand der Implementierung und beispielhafte Anwendungsszenarien. In Abschnitt 5 wird die Arbeit abschließend zusammengefasst.

2 Verwandte Arbeiten

In den DSMSs Borealis [CBB+03] und StreamGlobe [KSKR05] werden neue kontinuierliche Anfragen zunächst grob zerlegt und im Netzwerk verteilt. Häufig werden einzelne Operatoren Knoten zugeordnet. Nach der anfänglichen Verteilung werden Teile der kontinuierlichen Anfrage während der Verarbeitung verschoben, d.h., sie werden von Knoten zu Knoten übertragen. Dadurch können Kommunikationskosten gespart werden, indem bspw. Teilanfragen verschiedener Knoten zusammengefasst werden.

Das DSMS Stormy [LHKK12] zerlegt keine kontinuierlichen Anfragen, sondern führt sie stets vollständig auf einem Knoten aus. Die Auswahl des Knotens geschieht mittels eines Hashwertes, welcher aus der kontinuierlichen Anfrage gebildet wird: Jeder Knoten übernimmt einen Teil des Wertebereichs, sodass kontinuierliche Anfragen mit bestimmten Hashwerten bestimmten Knoten zugeordnet werden.

StreamCloud [GJPPM+12], Storm [TTS+14] und Stratosphere [WK09] nutzen Cloud-Infrastrukturen, um bei Bedarf verarbeitende, virtuelle Knoten in der Cloud zu erzeugen und Teilanfragen zuzuordnen. In Storm kann der Nutzer zusätzlich angeben, wie viele Instanzen für eine Operation erstellt werden sollen (bspw. für Fragmentierung oder Replikation). StreamCloud bietet unterschiedliche Zerlegungsstrategien, die später in dieser Arbeit aufgegriffen werden.

In SPADE, einer Anfragesprache von System S [GAW09], werden kontinuierliche Anfragen nach einem Greedy-Algorithmus zerlegt. Die Zuweisung der so entstandenen Teilanfragen wird mittels eines Clustering-Ansatzes durchgeführt. Dadurch sollen die Teilanfragen möglichst wenigen Knoten zugeordnet werden.

Daum et al. [DLBMW11] haben ein Verfahren für eine automatisierte, kostenoptimale Anfrageverteilung vorgestellt. Sie verfolgen damit andere Ziele als das in dieser Arbeit vorgestellte Konzept, bei dem es viel mehr um Flexibilität und Modularität als um Automatisierung geht.

Jedes hier vorgestellte DSMS besitzt seine eigene Vorgehensweise (mit eigenen Vor- und Nachteilen), kontinuierliche Anfragen im Netzwerk zu verteilen. Jedoch bieten sie – im Gegensatz zu dem hier vorgestellten Konzept – nicht die Flexibilität und Modularität, um Verteilungsstrategien bei Bedarf zu wechseln. Sie erlauben häufig nur mit großem Aufwand neue, evtl. domänenspezifische Strategien zu implementieren und einzusetzen.

3 Konzept

Häufig werden kontinuierliche Anfragen vom Nutzer deklarativ gestellt. Anschließend wird der Anfragetext i.d.R. in eine sprachenunabhängige Struktur übersetzt, die sich an Anfragepläne von DBMSs orientiert: die sogenannten *logischen Operatorgraphen*. Sie können als gerichtete, azyklische Graphen interpretiert werden, wobei die Knoten die Operatoren repräsentieren. Die Kanten stellen die Datenströme zwischen den Operatoren dar. Ein *logischer Operator* beschreibt, welche Operation auf einen Datenstrom ausgeführt

werden soll (bspw. Selektion, Projektion, Join). Sie beinhaltet jedoch nicht die konkrete Implementierung. Diese wird erst bei der tatsächlichen Ausführung der (Teil-) Anfragen auf einem Knoten eingesetzt. Aufgrund der Unabhängigkeit von der Sprache und von der Implementierung basiert das hier vorgestellte Konzept auf logischen Operatorgraphen. Es beschreibt also, wie die Verteilung eines logischen Operatorgraphen flexibel und erweiterbar durchgeführt werden kann.

Wie bereits erwähnt, ist eine einzige, fest programmierte Vorgehensweise zur Anfrageverteilung häufig nicht praktikabel. Dementsprechend wird in dieser Arbeit eine mehrschrittige Vorgehensweise verfolgt: (1) Partitionierung, (2) Modifikation und (3) Allokation. Abbildung 1 soll den Zusammenhang der Phasen zueinander verdeutlichen. Für jede Phase werden mehrere Strategien zur Verfügung gestellt, aus denen der Nutzer für jede kontinuierliche Anfrage wählen kann. Als Alternative ist eine automatisierte Auswahl vorstellbar, diese wird jedoch in dieser Arbeit nicht weiter verfolgt.

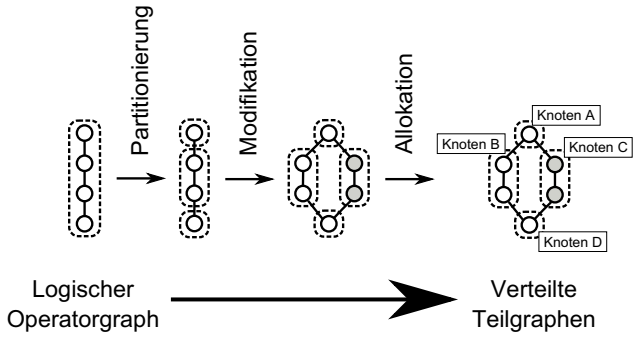

Abbildung 1: Phasen der Verteilung kontinuierlicher Anfragen.

Ausgehend von einem logischen Operatorgraphen wird in der ersten Phase, der *Partitionierung*, der Graph in *Teilgraphen* zerlegt, wobei keinerlei Änderungen am Graphen vorgenommen werden (z.B. neue Operatoren). Diese Teilgraphen werden dann *modifiziert*, um weitere Eigenschaften im Graphen sicherzustellen. Dabei können die Teilgraphen ergänzt, verändert oder auch entfernt werden. Beispielsweise wurde in der Abbildung der mittlere Teilgraph repliziert. Anschließend wird in der *Allokation* eine Zuordnung zwischen Teilgraph und ausführenden Knoten hergestellt und der Teilgraph schließlich übermittelt. In dieser Phase werden die Teilgraphen nicht mehr verändert.

```
1 #NODE_PARTITION <Partitionierungsstrategie> <Parameter>
2 #NODE_MODIFICATION <Modifikationsstrategie> <Parameter>
3 #NODE_ALLOCATE <Allokationsstrategie> <Parameter>
4
5 kontinuierliche Anfrage
```

Listing 1: Selektion der Strategien zur Verteilung einer kontinuierlichen Anfrage.

Die Auswahl der Strategien wird hier dem Nutzer überlassen. Ein Beispiel, wie der Nutzer eine kontinuierliche Anfrage im DSMS Odysseus [AGG+12] verteilen und entsprechende

Strategien auswählen kann, ist in Listing 1 zu sehen. Zu Beginn gibt der Nutzer die Strategie für jede Phase an (Zeilen 1 bis 3). Mittels Parameter können die Strategien weiter verfeinert werden (bspw. Replikationsgrad). Anschließend wird die eigentliche Anfrage formuliert, welche anhand der gewählten Kombination der Strategien verteilt wird.

Der Nutzer muss den Anfragetext nicht speziell anpassen, um diese verteilen zu können. Es müssen lediglich zuvor die eingesetzten Strategien spezifiziert werden. Für eine andere Anfrage können wiederum andere Strategien gewählt werden. In den folgenden Abschnitten werden die Phasen zur Verteilung genauer beschrieben und beispielhafte Strategien kurz erläutert.

3.1 Partitionierung

Aufgrund der Tatsache, dass eine gegebene kontinuierliche Anfrage in einem Netzwerk verteilt werden soll, werden einzelne Knoten häufig Teile der Anfrage erhalten. Dementsprechend muss zuvor entschieden werden, wie die Anfrage zerlegt werden soll. Das bedeutet in diesem Fall, dass der logische Operatorgraph in mehrere Teilgraphen zerlegt werden muss. Teilgraphen beschreiben somit, welche logischen Operatoren zusammen auf einem Knoten ausgeführt werden sollen. Es ist dabei wichtig, dass die Teilgraphen zueinander disjunkt sind, d.h., jeder logischer Operator der kontinuierlichen Anfrage befindet sich in genau einem Teilgraphen. Die logischen Operatoren müssen innerhalb eines Teilgraphen jedoch nicht zusammenhängend sein.

Das Finden einer geeigneten Zerlegung eines Graphen ist NP-hart [BMS+13]. Dementsprechend wird vorgeschlagen, mehrere Strategien zur Partitionierung anzubieten. Eine *Partitionierungsstrategie* erhält einen logischen Operatorgraphen und liefert eine Menge an disjunkten Teilgraphen. Einige Partitionierungsstrategien sind die folgenden:

QueryCloud Der logische Operatorgraph wird als ein Teilgraph behandelt. Das bedeutet, dass keine Zerlegung durchgeführt wird. Dies ist nützlich, wenn es sich um eine Anfrage mit einer geringen Zahl an logischen Operatoren handelt und eine Zerlegung nicht praktikabel erscheint.

OperatorCloud Jeder logischer Operator ist ein Teilgraph. Dies repräsentiert die maximale Zerlegung des Operatorgraphen. Diese Strategie ist für kontinuierliche Anfragen interessant, die wenige, jedoch sehr komplexe logische Operatoren beinhalten.

OperatorSetCloud In vielen kontinuierlichen Anfragen verursachen die zustandsbehafteten Operatoren die meiste Systemlast, wie bspw. Aggregationen [GJPPM+12]. Die Partitionierungsstrategie *OperatorSetCloud* zerlegt den logischen Operatorgraphen, sodass jeder Teilgraph maximal einen solchen Operator enthält. Dadurch ist es möglich, die zustandsbehafteten Operatoren verschiedenen Knoten zuzuteilen, sodass die Systemlast besser im Netzwerk verteilt werden kann.

Nutzerbasiert Falls es die Anfragesprache erlaubt, kann der Nutzer direkt angeben, welche Operatoren zusammen auf einem Knoten ausgeführt werden sollen. Diese Strategie ist besonders für Evaluationen praktisch, da damit bestimmte Szenarien der Verteilung nachgestellt und reproduziert werden können.

Auf Details wird im Rahmen dieser Arbeit verzichtet. Die ersten drei Strategien wurden dem Vorbild des DSMS StreamCloud nachempfunden und werden in [GJPPM+12] genauer erläutert. Es ist möglich, dass Entwickler weitere Strategien konzipieren und einsetzen. Dadurch können in konkreten Anwendungsszenarien bspw. spezielle Eigenschaften der Knoten und des Netzwerks ausgenutzt werden.

3.2 Modifikation

In der zweiten Phase wird der logische Operatorgraph modifiziert, um weitere Eigenschaften in der kontinuierlichen Anfrage sicherzustellen. Dieser Schritt ist optional und kann übersprungen werden, wenn keine Änderungen am logischen Operatorgraphen notwendig sind. Sind jedoch mehrere Modifikationen notwendig, kann diese Phase mehrfach durchgeführt werden. An dieser Stelle sind ebenfalls unterschiedliche Möglichkeiten vorstellbar. Jede *Modifikationsstrategien* erhält als Eingabe die Menge an Teilgraphen aus der ersten Phase. Die Ausgabe beinhaltet eine modifizierte Menge an Teilgraphen.

In dieser Phase ist ebenfalls vorgesehen, dass Entwickler weitere Strategien konzipieren und einsetzen. Jedoch wurden in der vorliegenden Arbeit folgende Modifikationsstrategien betrachtet:

Replikation Jeder Teilgraph wird (u. U. mehrfach) repliziert. Dadurch kann jeder Teilgraph auf mehreren Knoten ausgeführt werden. Solange mindestens ein Knoten den Teilgraphen ausführt, können (Zwischen-) Ergebnisse berechnet und gesendet werden.

Horizontale Fragmentierung Ähnlich zur Replikation wird jeder Teilgraph repliziert, jedoch empfängt jede Kopie nur einen Teil des Datenstroms. Die Ergebnisse werden am Ende wieder zusammengefasst. Damit kann die Verarbeitung parallelisiert werden, was bei besonders hohen Datenraten oder komplexen Berechnungen sinnvoll ist.

Eine Illustration beider Strategien ist in Abbildung 2 zu sehen. Links ist der Einsatz der

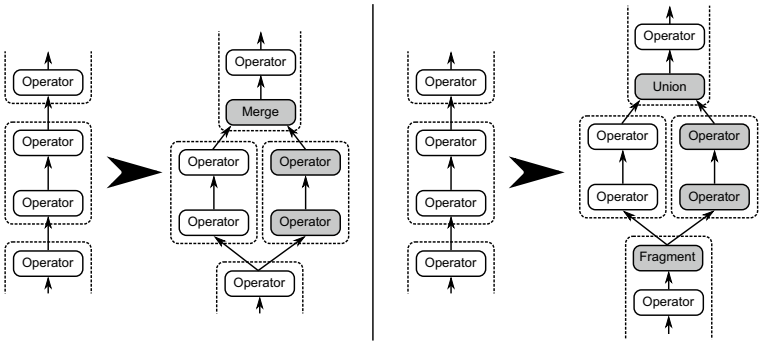

Abbildung 2: Verwendung der Replikations- (links) und der horizontalen Fragmentierungsstrategie (rechts).

Replikation als Modifikationsstrategie zu sehen: der Teilgraph wird kopiert und die replizierten (Teil-) Ergebnisse werden mittels eines speziellen Merge-Operator vereinigt. Der *Merge*-Operator erkennt und entfernt Duplikate in den Datenströmen, sodass die Replikation die Verarbeitungsergebnisse nicht unnötig vervielfacht. Im Rahmen der horizontalen Fragmentierung werden Teilgraphen ebenfalls kopiert (in der Abbildung rechts). Der vorgelagerte *Fragment*-Operator zerlegt den eintreffenden Datenstrom in disjunkte Fragmente, die parallel verarbeitet werden (bspw. mittels Hashwerten der Datenstromelemente). Der Union-Operator vereinigt die Teilmengen schließlich zu einem Datenstrom. In der Modifikationsphase ist ebenfalls vorgesehen, dass Entwickler eigene Strategien entwickeln und einsetzen (z.B. vertikale Fragmentierung).

3.3 Allokation

Die dritte Phase – die Allokation – beinhaltet die Aufgabe, Teilanfragen den Knoten im Netzwerk zur Ausführung zuzuordnen. Auch hier sind in Abhängigkeit zum vorliegenden Netzwerk verschiedene Vorgehensweisen vorstellbar, sodass im Rahmen dieser Arbeit mehrere Strategien betrachtet werden. Jede *Allokationsstrategie* erhält als Eingabe die Menge an (ggfs. replizierten und/oder fragmentierten) Teilgraphen. Die Ausgabe umfasst eine 1:n-Zuordnung zwischen ausführenden Knoten und Teilanfragen. Das bedeutet, dass ein Knoten mehrere Teilgraphen erhalten kann, jedoch wird jeder Teilgraph genau einem Knoten zugeordnet. Folgende Allokationsstrategien wurden bisher verfolgt:

Nutzerbasiert Der Nutzer gibt die Zuordnung vor (bspw. über eine grafische Oberfläche oder durch Annotationen im Anfragetext). Der Nutzer kann über Spezialwissen verfügen, die es ermöglichen, eine optimale Zuordnung anzugeben.

Round-Robin Die Teilgraphen werden der Reihe nach an die Knoten verteilt.

Lastorientiert Die Teilgraphen werden an die Knoten verteilt, welche aktuell die geringste Auslastung aufweisen. Durch diese Vorgehensweise kann die Systemlast im Netzwerk verteilt werden.

Contract-Net Für jeden Teilgraphen wird eine Auktion ausgeschrieben, und jeder Knoten kann bei Interesse ein Gebot abgeben. Ein Gebot beschreibt die Bereitschaft, den Teilgraphen zu übernehmen. Dabei kann ein Gebot aus verschiedenen Faktoren zusammengesetzt werden. Beispielsweise spielt die Menge an verfügbaren Systemressourcen eine Rolle: Je mehr Ressourcen frei sind, desto besser kann der Teilgraph ausgeführt werden. Der Knoten mit dem höchsten Gebot erhält schlussendlich den Teilgraphen.

Ist das Netzwerk bekannt und sind alle Knoten homogen, kann Round-Robin für eine schnelle und einfache Verteilung der Anfrage genutzt werden. Sollen Auslastungen der Knoten berücksichtigt werden, ist die lastorientierte Strategie vorzuziehen. Sie kann auch eingesetzt werden, wenn die Knoten über unterschiedliche Leistungskapazitäten verfügen. Contract-Net sollte benutzt werden, wenn die Autonomie der Knoten zu berücksichtigen ist (d.h., die Knoten entscheiden selbstständig, welche Teilgraphen sie ausführen wollen). Das in dieser Arbeit vorgestellte Konzept sieht vor, dass Entwickler eigene Allokationss-

trategien implementieren können, um bspw. domänenspezifisches Wissen einzusetzen oder spezielle Netzwerkstrukturen zu berücksichtigen.

4 Aktueller Stand

Das oben beschriebene Konzept wurde in Odysseus [AGG+12] als zusätzliche Komponente implementiert. Jede oben genannte Strategie ist verfügbar und kann in kontinuierlichen Anfragen eingesetzt werden. Dadurch kann Odysseus in unterschiedlichen Netzwerk-Architekturen eingesetzt werden, ohne dass umfangreiche Änderungen an der Verteilung durchgeführt werden müssen (es muss lediglich die Strategieauswahl angepasst werden). Im Folgenden werden zwei Anwendungsbeispiele von Odysseus vorgestellt, die aufzeigen sollen, wie das oben genannte Konzept flexibel eingesetzt werden kann.

Anwendungsfall 1: In einem Anwendungsfall wird Odysseus auf mehrere Knoten in einem heterogenen und autonomen P2P-Netzwerk eingesetzt, um Sportereignisse in Echtzeit auszuwerten. Die Daten werden mit Hilfe von aktiven Sensoren aufgenommen und an das Netzwerk gesendet. Die Sensoren sind dabei an spielrelevanten Entitäten wie den Spielern und dem Ball angebracht. Die Analyse geschieht mit zuvor verteilten kontinuierlichen Anfragen. Da zum einen ein solches Sensornetzwerk mehrere tausend Datenstromelemente pro Sekunde erzeugen kann und zum anderen die Analyse dieser Daten teuer ist, bietet sich eine Anfrageverteilung an, die in Listing 2 dargestellt ist. Konkret sollen

```
1 #NODE_PARTITION operatorsetcloud
2 #NODE_MODIFICATION fragmentation_horizontal hash n
3 #NODE_ALLOCATE contractnet
4
5 originale Analyse-Anfrage
```

Listing 2: Beispielhafte Verwendung der Anfrageverteilung für eine Sportanalyse.

Operatoren, die viel Last erzeugen, auf unterschiedlichen Knoten ausgeführt werden (*OperatorSetCloud*-Partitionierungsstrategie). Dadurch werden unterschiedliche Sportanalysen von verschiedenen Knoten des Netzwerks übernommen. Zusätzlich soll der Datenstrom fragmentiert werden, um die Systemlast für einzelne Knoten zu verringern (Modifikationsstrategie hash-basierte *horizontale Fragmentierung* und *n* Fragmenten). Da die Knoten heterogen und autonom sind, wird in diesem Fall die *Contract-Net*-Allokationsstrategie eingesetzt.

Anwendungsfall 2: In einem Windpark liefert jedes Windrad kontinuierlich Statusinformationen, die als Datenstrom interpretiert werden (z.B. die aktuell erzeugte Energie sowie Windrichtung und -geschwindigkeit). Diese Datenströme werden zur Überwachung und Kontrolle der Windräder benötigt und an ein homogenen Cluster aus Odysseus-Instanzen gesendet. Es können spezielle Datenstromelemente versendet werden, die Alarmmeldungen oder Störungen signalisieren. Aus diesem Grund ist es wichtig, dass jedes Datenstromelement (jede Alarmmeldung oder Störung) verarbeitet wird, auch wenn ein Knoten in dem verarbeitendem Cluster ausfällt. Listing 3 zeigt eine mögliche Anfrageverteilung für dieses

Szenario unter der Verwendung eines homogenen Clusters. Konkret sollen alle Operatoren

```
#NODE_PARTITION querycloud
#NODE_MODIFICATION replication n
#NODE_ALLOCATE roundrobin

originale Überwachungs-Anfrage
```

Listing 3: Beispielhafte Verwendung der Anfrageverteilung für eine Windpark-Überwachung.

der Anfrage auf einen Knoten ausgeführt werden, da die Cluster-Knoten mit ausreichend Ressourcen ausgestattet sind (*QueryCloud*-Partitionierungsstrategie). Zusätzlich wird der Datenstrom repliziert, um die Ausfallsicherheit zu erhöhen und um Alarmmeldungen nicht zu verlieren (Modifikationsstrategie *Replikation* mit *n* Replikaten). Als Allokator kommt die *Round-Robin*-Strategie zum Einsatz, da es sich um ein homogenes Cluster mit identischen Knoten handelt.

5 Zusammenfassung

Für eine Anfrageverteilung in verteilten DSMSs gibt es je nach Netzwerk-Architektur und Anwendungsfall unterschiedliche Strategien. Viele Systeme haben sich auf eine Verteilungsstrategie spezialisiert, wodurch sie nur mit großem Aufwand an eine Änderung der Netzwerk-Architektur angepasst werden können. Außerdem ist die Anfrageverteilung in vielen Systemen nicht durch den Nutzer konfigurierbar, was es unmöglich macht anwendungsspezifische Kenntnisse einzubringen.

Aus diesem Grund wurde in dieser Arbeit ein modularer Konzept für eine flexible und erweiterbare Anfrageverteilung in verteilten DSMSs vorgestellt. Das Konzept sieht dabei einen logischen Operatorgraphen als Eingabe vor und liefert verteilte Teilgraphen als Ausgabe. Strukturell umfasst er drei Schritte: (1) Partitionierung, (2) Modifikation und (3) Allokation.

Bei der Partitionierung wird der logische Operatorgraph in disjunkte Teilgraphen zerlegt, um die Operatoren zu identifizieren, die gemeinsam auf einem Knoten im Netzwerk ausgeführt werden sollen. Die optionale Modifikation erlaubt es Mechanismen wie Fragmentierung oder Replikation zu verwenden, indem die Teilgraphen modifiziert werden. In der Allokationsphase werden die einzelnen (modifizierten) Teilgraphen Knoten im Netzwerk zugewiesen.

Für jeden der drei Schritte gibt es Schnittstellen, wodurch unabhängige Strategien miteinander kombiniert werden können. Dieser modulare Aufbau ermöglicht zum einen eine individuelle Anfrageverteilung. Zum anderen können bereits vorhandene Strategien aus anderen Arbeiten und Systemen (z.B. eine Allokationsstrategie) integriert werden. In dieser Arbeit wurden für jeden der drei Teilschritte beispielhafte Strategien vorgestellt.

Das Konzept wurde im DSMS Odysseus als zusätzliche Komponente implementiert und erfolgreich in verschiedenen Anwendungsszenarien eingesetzt. Zwei Anwendungsszena-

rien wurden in dieser Arbeit kurz vorgestellt: (1) Die Sportanalyse in Echtzeit mittels einem P2P-Netzwerk aus heterogenen Knoten und (2) die Überwachung eines Windparks. Odysseus musste in beiden Anwendungsfällen lediglich bei der Strategieauswahl angepasst werden. Dies zeigt, dass das oben beschriebene Konzept zur Verteilung kontinuierlicher Anfragen flexibel und erweiterbar ist.

Literatur

[AGG+12] H.-Jürgen Appelrath, Dennis Geesen, Marco Grawunder, Timo Michelsen und Daniela Nicklas. Odysseus: a highly customizable framework for creating efficient event stream management systems. DEBS '12, Seiten 367–368. ACM, 2012.

[BMS+13] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders und Christian Schulz. Recent Advances in Graph Partitioning. *CoRR*, abs/1311.3144, 2013.

[CBB+03] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing und Stan Zdonik. Scalable Distributed Stream Processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.

[DLBMW11] Michael Daum, Frank Lauterwald, Philipp Baumgärtel und Klaus Meyer-Wegener. Kalibrierung von Kostenmodellen für föderierte DSMS. In *BTW Workshops*, Seiten 13–22, 2011.

[GAW09] Buğra Gedik, Henrique Andrade und Kun-Lung Wu. A Code Generation Approach to Optimizing High-performance Distributed Data Stream Processing. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, Seiten 847–856, New York, NY, USA, 2009. ACM.

[GJPPM+12] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente und Patrick Valduriez. StreamCloud: An Elastic and Scalable Data Streaming System. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2351–2365, 2012.

[KSKR05] Richard Kuntschke, Bernhard Stegmaier, Alfons Kemper und Angelika Reiser. Streamglobe: Processing and sharing data streams in grid-based p2p infrastructures. In *Proceedings of the 31st international conference on Very large data bases*, Seiten 1259–1262. VLDB Endowment, 2005.

[LHKK12] Simon Loesing, Martin Hentschel, Tim Kraska und Donald Kossmann. Stormy: an elastic and highly available streaming service in the cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, Seiten 55–60, New York, NY, USA, 2012. ACM.

[TTS+14] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthikeyan Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal und Dmitriy V. Ryaboy. Storm@twitter. In *SIGMOD Conference*, Seiten 147–156, 2014.

[WK09] Daniel Warneke und Odej Kao. Nephele: Efficient Parallel Data Processing in the Cloud. In *Proceedings of the 2Nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, Seiten 8:1–8:10, New York, NY, USA, 2009. ACM.