Vorstellung des Streamkonzepts

Seminar: Datenbanken und Informationssysteme
Datum: Juni 2005

Bearbeiter: Benjamin Mock

Betreuer: Dipl. Inf. Boris Stumm

Inhaltsverzeichnis

1. Einleitung 3

2. Datenströme 4

2.1. Punkt- und Tupelströme 4

2.2. XML-Datenströme 4

3. DSMS vs. DBMS 5

4. Anfragen 6

4.1. Einmalig vs. Kontinuierlich 6

4.2. Ad-hoc vs. Vordefiniert 6

4.3. Angenäherte Anfragebeantwortung 7

4.4. Sliding Windows 7

5. Joins 8

6. Datenstrom-Management-Systeme (DSMS) 10

6.1. Tupelstromsysteme 10

6.1.1. AURORA 10

6.1.2. STREAM 12

6.2. XML-Stromsysteme 14

6.2.1. XPath 14

6.2.2. SPEX 17

7. Load Shedding 19

8. Zusammenfassung 21

9. Literaturverzeichnis 22

1. Einleitung

Datenströme sind eine alltägliche Erscheinung. Selbst die menschliche Wahrnehmung basiert auf einströmenden Daten. Die Datenströme, mit denen sich diese Ausarbeitung auseinander setzt, sind jedoch in der Regel digitaler Natur.

Datenströme gewinnen gerade in letzter Zeit mehr und mehr an Bedeutung. Vor allem im Internet werden die Möglichkeiten von Datenströmen immer öfter genutzt. Beispiele dafür sind Multimediaströme wie Internetradio oder Videostreams. Dabei ist es möglich, noch während des Ladens der Daten, die entsprechenden Medien abzuspielen.

Die Anwendungsgebiete von Datenströmen sind sehr vielfältig. Nicht nur in der Nachrichtenüberwachung, wie bei Presse-, Börsen- oder Meteorologienachrichten, sondern auch bei Systemüberwachungen, wie der Verkehrsüberwachung oder der Netzwerkverwaltung, werden Datenströme eingesetzt. Auch bei der Analyse von wissenschaftlichen Messdaten kommen Datenströme zum Einsatz [BCD+05].

Nachdem in dieser Einleitung schon auf Einsatzgebiete von Datenströmen hingewiesen wurde, werden ihre Charakteristika in Abschnitt *2* noch einmal genauer betrachtet.

Zwischen Datenbank-Management-Systemen (DBMS) und Datenstrom-Management-Systemen (DSMS) bestehen zum Teil fundamentale Unterschiede. Werden bei einem DBMS alle Daten gespeichert und indiziert, so ist dies bei einem DSMS nicht möglich, weil die ankommenden Datenströme potentiell eine unendliche Länge aufweisen. Die weiteren Unterschiede werden in Abschnitt *3* erläutert und dargestellt.

Ein weiterer wichtiger Unterschied zwischen Datenstromsystemen und traditionellen Datenbanksystemen besteht in der Art und Weise, wie Anfragen gestellt werden. Abschnitt *4* unterscheidet die verschiedenen Arten von Anfragen, von denen die kontinuierliche Anfrage einen sehr großen Stellenwert in dieser Ausarbeitung einnimmt.

Abschnitt *5* erläutert die Techniken, um mehrere unterschiedliche Datenströme zueinander in Verbindung zu setzen, unter den Bedingungen, die eine Verarbeitung von Datenströmen mit sich bringt. Dazu gehören Speicherbeschränkungen und vor allem Zeitbeschränkungen.

Existierende Datenstromsysteme für die verschiedenen Stromarten werden in Abschnitt *6* aufgeführt und erläutert.

Oft ist es notwendig, dass ein Datenstrom in Echtzeit verarbeitet wird. Load Shedding ermöglicht dies auch in Situationen der Überlast. Mit Load Shedding ist es möglich, gute, angenäherte Ergebnisse zu erzielen, obwohl nicht alle ankommenden Tupel des Stroms verarbeitet werden können. In Abschnitt *7* wird diese Technik erläutert.

Abgeschlossen wird diese Ausarbeitung mit einer Zusammenfassung in Abschnitt *8*.

2. Datenströme

Die Definition des Wortes „Datenstrom“ lässt sich schon aus dem Wort selbst heraus ableiten. Man versteht unter einem Datenstrom einen kontinuierlichen Strom von Daten. Datenströme bestehen aus einer potentiell unendlichen Folge von Datensätzen. Ankunftsraten der Datensätze können weder vorhergesagt, noch beeinflusst werden [BBD+02].

Die Größe des Stroms, sowie die Menge und die unvorhersehbaren Ankunftszeitpunkte der Daten, machen eine Speicherung vor der Verarbeitung, wie es in traditionellen Datenbanksystemen üblich ist, unmöglich. Ein beliebiger Zugriff auf die Datensätze ist nicht gegeben. Wurde ein Element einmal bearbeitet, wird es im Normalfall verworfen. Nur ein sehr kleiner Prozentsatz der Daten kann tatsächlich gespeichert werden, um späteren Anfragen zur Verfügung zu stehen. Es sind deshalb Verfahren nötig, mit denen es möglich ist, die ankommenden Daten direkt zu analysieren und zu überwachen [BrFO04]. Eine Schwierigkeit dabei ist es, dass diese Verfahren in Echtzeit, ohne Verzögerung, arbeiten müssen. Stellt man sich beispielsweise die Überwachung von Patienten in einem Krankenhaus vor, so wäre es auf jeden Fall unzureichend, kritische Daten auch nur mit einer Minute Verzögerung zu melden.

In [BrFO04] wird generell zwischen drei verschiedenen Arten von Datenströmen unterschieden: Punktströme, Tupelströme und XML-Ströme.

2.1 Punkt- und Tupelströme

Punkt-, sowie Tupelströme sind „flach“. Sie besitzen keine Struktur und weisen keine Schachtelung auf. Die Datensätze dieser Ströme haben alle die gleiche Länge. Tupelströme bestehen aus relationalen Tupeln, wie sie auch in traditionellen Relationen wieder gefunden werden können. Punktströme hingegen bestehen nur aus Zahlen bzw. Zeichen, also skalaren Werten. Man kann sie deshalb als spezielle Art von Tupelströmen ansehen.

2.2 XML-Datenströme

XML-Ströme sind strukturiert und können beliebig tief verschachtelt sein. Der Text, als eigentlicher Inhalt, wird durch die Strukturierung mit zusätzlichen Informationen versehen. Die Länge der einzelnen Datensätze wird dadurch ebenfalls unterschiedlich. Das macht die Auswertung dieser Ströme sehr komplex. Jedoch wird ihnen eine große Bedeutung zugemessen, denn XML wird im Allgemeinen als De-facto-Standard zum Datenaustausch im Internet angesehen.

3. DSMS vs. DBMS

Die Daten in traditionellen Datenbank-Management-Systemen (DBMS) werden im Normalfall dauerhaft in Relationen gespeichert. Es wird davon ausgegangen, dass diese Daten korrekt und vollständig sind. Zur schnelleren Bearbeitung werden sie indiziert. Für die Speicherung der Daten steht fast unbegrenzter Speicherplatz zur Verfügung. Der Zugriff auf die Daten ist beliebig möglich.

Soll eine Anfrage an das System gestellt werden, so ist ein aktiver Nutzer nötig. Man spricht von einer Pull-Kommunikation, denn der Datenverbraucher, in diesem Fall der Anfragesteller, fordert selbst die Daten an, die er benötigt. Eine Anfrage wird komplett ausgewertet, bevor das Ergebnis ausgegeben wird. Sie stellt einen Schnappschuss des Systems zu genau dem Zeitpunkt der Auswertung dar. Das Ergebnis ist exakt und entspricht einer Auswertung des vollständigen Datenbestands. Es wird keine Echzeit-Verarbeitung gefordert. Man kann ein DBMS als passives Repository ansehen, auf dem der Nutzer aktiv Operationen durchführen kann. Daher spricht man in diesem Fall von einem Human-Active, DBMS-Passive (HADP) Modell [CCC+02].

Daten aus Strömen können zumeist nur im Hauptspeicher abgelegt werden, weil sie schnell und in ihrer Frequenz variierend ankommen. Dadurch ist der Speicherplatz stark eingeschränkt. Nur ein Bruchteil des Datenstroms kann zwischengespeichert werden. Das bedeutet, dass meistens nur ein einmaliger, sequentieller Zugriff auf die Daten möglich ist. Deshalb werden so genannte Einpass-Verfahren in Datenstrom-Management-Systemen (DSMS) eingesetzt.
Anfragen verweilen im System und werden zur Anfrageoptimierung indiziert. Sie werden kontinuierlich an neu ankommende Daten gestellt. Es ist kein aktiver Benutzer nötig, der die Anfrage jedes Mal stellt. Man spricht daher von einer Push-Kommunikation. Die entsprechenden Daten werden automatisch an den Benutzer übersandt. Die Ergebnisse sind allerdings nicht immer genau. Falls die Frequenz der ankommenden Daten zu hoch zum Bearbeiten ist, wird es nötig, Tupel zu verwerfen, ohne sie in die Auswertung mit einfließen zu lassen. Dadurch verlieren die Ergebnisse an Genauigkeit. Anfragen können durch die potentielle Unendlichkeit von Datenströmen nie vollständig ausgewertet werden. Es werden immer nur Teilergebnisse erzeugt. Die Daten kommen nicht vom Benutzer, der sie durch Transaktionen eingibt, sondern durch externe Quellen. Solche externen Quellen können beispielsweise Sensoren sein. Aufgabe des Systems ist es den Benutzer über abnormale Werte, oder zu bestimmten Zeitpunkten zu informieren. Dieses Modell nennt man daher DBMS-Active, Human-Passive (DAHP).

	DSMS	DBMS
Daten	transient	persistent
Anfragen	persistent	transient
Änderungen	(zumeist) nur hinzufügen	beliebig
Ergebnisse	evtl angenähert	exakt
Datenzugriff	möglichst Einpass-Verfahren	beliebig
Indizierung	Anfragen	Daten

Abbildung 1 Unterschiede zwischen DSMS und DBMS

Die Unterschiede zwischen einem DBMS und einem DSMS sind in Abbildung noch einmal zusammenfassend dargestellt.

4. Anfragen

Nach [BrFO04] können Anfragen an Datenbanksysteme bzw. Datenstromsysteme völlig unterschiedlicher Natur sein. Die verschiedenen Arten und ihre jeweiligen Verwendungen werden in diesem Abschnitt vorgestellt.

4.1 Einmalige Anfragen vs. Kontinuierliche Anfragen

Unter einer einmaligen Anfrage versteht man eine Anfrage, die nur einmal an das System gestellt wird. Sie stellen einen Schnappschuss des Zustands des Systems, zu genau dem Zeitpunkt der Anfragestellung dar. Das Ergebnis der Auswertung wird nach Auswertung der Anfrage sofort an den Benutzer weitergegeben. Diesen Typ von Anfrage findet man häufig in traditionellen DBMS.

Unter kontinuierlichen Anfragen versteht man Anfragen an das System, die immer dann automatisch gestellt werden, wenn neue Daten ankommen. Die Auswertung einer solchen Anfrage findet über einen längeren Zeitraum statt. Allerdings bezieht sich die Kontinuität nur auf die Auswertung der Daten, nicht jedoch auf die Berichterstattung an den Benutzer. Diese ist zwar ebenfalls kontinuierlich, also mit Datenströmen möglich, jedoch nicht zwingend. Genauso ist es vorstellbar, dass die Ergebnisse in bestimmten Zeitintervallen, beim Auftreten bestimmter Ereignisse oder per Aufforderung durch den Benutzer weitergegeben werden. Es gibt kein einzelnes Ergebnis, sondern die Ergebnisse werden inkrementell erstellt und spiegeln genau den Datenstrom wieder, der bis zu diesem Zeitpunkt „gesehen“ und abgearbeitet wurde.

4.2 Vordefinierte Anfragen vs. Ad-hoc-Anfragen

Vordefinierte Anfragen sind dem System bekannt, bevor Daten angekommen sind, die für die Auswertung der Anfrage relevant sind. Im Normalfall sind das kontinuierliche Anfragen, die dem System im Voraus bekannt sind, es können aber auch geplante einmalige Anfragen sein.

Ad-hoc-Anfragen werden zu einem beliebigen Zeitpunkt an das System gestellt. Da die Daten eines Stroms, die bereits bearbeitet wurden, im Normalfall nicht gespeichert werden, ist es unmöglich, zu einem späteren Zeitpunkt auf sie zurückzugreifen. Trotzdem können Ad-hoc-Anfragen gestellt werden, die sich auch auf vergangene Daten beziehen. Um dieses Problem zu behandeln gibt es zwei Möglichkeiten: Die erste ist, Ad-hoc-Anfragen nur auf zukünftige Daten zu beziehen. Vergangene Daten werden einfach ignoriert. Eine zweite Möglichkeit ist, knappe Zusammenfassungen der alten Daten aufzuheben. Kommt dieser Ansatz zum Einsatz, ist allerdings eine Trade-off-Entscheidung zwischen dem benötigten Speicher (und auch der Zeit zum Schreiben der Daten) und einer möglichst breiten Abdeckung möglicher späterer Anfragen notwendig. Je mehr Zusammenfassungen gespeichert werden und je ausführlicher diese sind, desto mehr mögliche spätere Anfragen werden abgedeckt. Jedoch werden so der Speicherbedarf und die Zeit, die zum Schreiben der Daten benötigt wird, sehr groß.

4.3 Angenäherte Anfragebeantwortung

Auf Grund der Tatsache, dass die Ressourcen eines Systems beschränkt sind, ist es nicht immer möglich, völlig exakte Anfrageergebnisse zu erstellen. Ist beispielsweise die Ankunftsrate der Daten so hoch, dass nicht alle Daten verarbeitet oder gespeichert werden können, so ist es notwendig angenäherte Ergebnisse, die immer noch eine ausreichende Genauigkeit bieten, zu erstellen.

Ein generelles Systemgerüst wird in [BBD+02] vorgestellt. Dort wird angenommen, dass eine Anfrage mit Hilfe einer Datenstruktur beantwortet wird, die inkrementell verwaltet werden kann. Die allgemeinste Beschreibung einer solchen Datenstruktur besteht aus zwei Operationen, update(tupel) und computeAnswer(). Sobald ein Tupel des Datenstroms ankommt, aktualisiert die update-Operation die Datenstruktur. Neue oder aktualisierte Ergebnisse der entsprechenden Anfrage werden von der computeAnswer-Operation erstellt. Sind beide Operationen schnell genug, wird für jedes ankommende Tupel sofort ein aktualisiertes Ergebnis erstellt. In diesem Fall sind keine speziellen Techniken nötig. Ist jedoch eine Operation zu langsam, oder beide, ist es nicht möglich, das Ergebnis immer aktuell zu halten.

Falls die computeAnswer-Operation zu langsam ist, müssen die Tupel gepuffert und computeAnswer so oft wie möglich ausgeführt werden. Das Ergebnis ist nicht aktuell und weißt nicht die maximale Genauigkeit für den momentanen Zeitpunkt auf. Das genaue Ergebnis kommt immer erst mit etwas Verzögerung an.

Ist die update-Operation zu langsam bedeutet das, dass die Tupel schneller ankommen als sie verarbeitet werden können. Es ist so unmöglich, ein Ergebnis unter Betrachtung aller Tupel zu berechnen. Man wendet in diesem Fall Sampling-Algorithmen an, die nur einen Teil des Datenstroms, ein Sample, zur Berechnung des Ergebnisses benutzen.

In Absatz *7* wird noch einmal genauer auf das Thema der angenäherten Anfragebeantwortung eingegangen.

4.4 Sliding Windows

Eine weitere Möglichkeit angenäherte Ergebnisse auf eine Anfrage zu erstellen, sind so genannte „Sliding Windows“ (sich verschiebende Fenster). Beispielsweise sind nur Daten des letzten Tags interessant und kommen daher für eine Berechnung in Frage. Man spricht deswegen von einem sich verschiebenden Fenster, weil man die angewandte Technik damit gut beschreibt: auf einer Seite des Rahmens kommen neue Daten an, verschiebt man ihn, fallen auf der anderen Seite automatisch ältere Daten hinaus.

Ein großer Vorteil dieser Methode ist, dass neue Daten, die auch in der realen Anwendung größere Bedeutung haben als ältere, hervorgehoben werden. Betrachtet man beispielsweise das Verkehrsaufkommen auf einer Autobahn, ist es im Allgemeinen interessanter, Daten der letzten Stunden auszuwerten, als Daten, die schon einige Jahre alt sind. Ein weiterer Vorteil ist die Einfachheit dieses Annäherungsverfahrens. Der Benutzer versteht genau, wie seine Anfrageergebnisse zu werten sind und was bei der Auswertung nicht berücksichtigt wurde. Im Gegensatz zu vielen anderen Annäherungsverfahren sind Sliding Windows deterministisch und liefern somit auf zwei gleiche Anfragen über dieselben Daten das gleiche Ergebnis.

5. Joins

Oftmals ist es notwendig, die Daten von zwei oder mehr unterschiedlichen Eingabeströmen zueinander in Verbindung zu setzen, um aussagekräftige Ergebnisse zu erhalten. Ein solcher Join- bzw. Verbund- Algorithmus soll in diesem Abschnitt vorgestellt werden.

Bei einem normalen Hash-Join kommt es irgendwann zum Problem des Hauptspeicher-Überlaufs, wenn nicht mehr alle Daten im Hauptspeicher festgehalten werden können. Der XJoin, ein erweiterter Hash Join, umgeht dieses Problem, indem er die Tupel der Operanden in Partitionen aufgliedert. Ein Teil wird im Hauptspeicher abgelegt, der andere auf der Festplatte. Wird der Platz im Hauptspeicher knapp, werden ältere Tupel auf die Festplatte geschrieben.

Der ganze XJoin-Vorgang läuft in 3 Phasen ab:

- *Memory-to-Memory Phase:*
  In dieser Phase werden die Tupel im Hauptspeicher gespeichert und miteinander verglichen. Kommen Tupel in dieser Phase an und es ist noch genügend Platz im Hauptspeicher, so wird das ankommende Tupel mit der vorhandenen Hash Tabelle verglichen und an die eigene Tabelle angehängt. Ist kein Platz mehr im Hauptspeicher um ein Tupel zu speichern, so werden einige der älteren Tupel auf der Festplatte abgelegt, damit das neue Tupel im Hauptspeicher an die entsprechende Hash-Tabelle angehängt werden kann.

- Disk-to-Memory-Phase
  Diese Phase beginnt, wenn gerade keine neuen Tupel ankommen, das System also nicht ausgelastet ist. Eine Menge, der auf der Platte gespeicherten Tupel, wird auf Treffer mit den Tupeln im Hauptspeicher überprüft. Das passiert so lange, bis wieder neue Tupel ankommen. So kann XJoin auch Ergebnisse liefern, obwohl gerade keine neuen Daten ankommen.

- Disk-to-Disk-Phase
  Die letzte Phase, die Disk-to-Disk-Phase beginnt, wenn sichergestellt ist, dass keine neuen Tupel mehr ankommen. Diese Phase garantiert, dass alle Treffer gefunden werden können. Die vorhandenen Hash-Tabellen werden vollständig miteinander in Verbund gesetzt. Allerdings kann diese Phase nur ausgeführt werden, wenn es sich um endliche Ströme handelt.

Während in der ersten Phase nur Ergebnisse zu Tupeln geliefert werden, die gleichzeitig im Hauptspeicher waren, werden in der zweiten Phase die Ergebnisse geliefert, bei denen ein Teil der Tupel auf der Festplatte, der andere Teil hingegen im Hauptspeicher abgelegt ist. Ergebnisse zu Tupeln, die sich nur auf der Festplatte befinden, werden erst in Phase 3 gefunden. Jedoch kann es so zu Duplikaten kommen, da sich die Phasen überlappen. Um das zu verhindern, werden alle Tupel mit einem Zeitstempel versehen. Die Tupel erhalten bei ihrer Ankunft im Hauptspeicher (ATS = arrival timestamp) sowie beim Ablegen auf der Festplatte (DTS = departure timestamp) jeweils einen Zeitstempel. Somit wird durch ATS und DTS das genaue Zeitintervall festgehalten, in dem sich ein Tupel im Hauptspeicher befunden hat.

In der ersten Phase kann es nicht zu Duplikaten kommen, weil nur Tupel verarbeitet werden, die sich alle im Hauptspeicher befinden. Das bedeutet, dass alle im Hauptspeicher möglichen Ergebnisse bereits erzeugt wurden. Bei der Ausführung der zweiten Phase hingegen, sind

Duplikate möglich. Deshalb wird geprüft, ob sich ATS und DTS überschneiden. Ist das der Fall, waren die Tupel gleichzeitig im Hauptspeicher und wurden bereits miteinander verbunden.

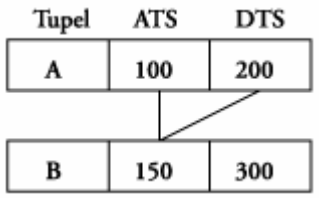

Abbildung 2 Zeitstempel der Tupel überlappen sich, ein Verbund wurde schon in Phase 1 durchgeführt

Abbildung 2 zeigt zwei Tupel, A und B, und die zugehörigen Zeitstempel. Tupel A ist zum Zeitpunkt 100 im Hauptspeicher angekommen und wurde zum Zeitpunkt 200 auf die Festplatte ausgelagert. Tupel B hingegen ist zum Zeitpunkt 150 im Hauptspeicher angekommen und wurde zum Zeitpunkt 300 ausgelagert. Die beiden Tupel waren zwischen Zeitpunkt 150 und 200 zusammen im Hauptspeicher und wurden daher schon miteinander verbunden. Ein Verbund in Phase 2 ist deshalb nicht mehr nötig. In Abbildung 3 überlappen sich die Zeitstempel nicht. Tupel B kam erst im Hauptspeicher an, als Tupel A schon auf die Festplatte ausgelagert war. Sie waren nicht zusammen im Hauptspeicher, daher kann in Phase 2 ein Verbund durchgeführt werden, ohne dass es zu Duplikaten kommt.

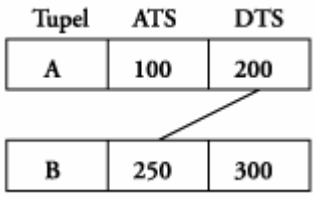

Abbildung 3 Zeitstempel der Tupel überlappen sich nicht, daher kann ein Join in Phase 2 durchgeführt werden

Um ausschließen zu können, dass in Phase 3 Duplikate gebildet werden, müssen weitere Zeitstempel eingeführt werden. Durch ATS und DTS wird nämlich nur festgehalten, wann die Tupel sich im Hauptspeicher befunden haben, nicht jedoch, ob sie in Phase 2 schon verbunden wurden. Daher wird dem Teil der Tupel, der auf der Festplatte gespeichert ist, eine History Liste angehängt. Diese enthält Einträge der Form {DTSlast, ProbeTS}. DTSlast ist der DTS Wert des letzten Tupels des auf der Festplatte gespeicherten Teils, das auf Treffer mit den Tupeln im Hauptspeicher überprüft wurde. In ProbeTS wird der Zeitstempel gespeichert, der angibt, wann diese Prüfung stattfand, also wann Phase 2 für die entsprechenden Tupel durchlaufen wurde. Mit diesen beiden Angaben kann in der dritten Phase überprüft werden, ob ein Verbund durchgeführt werden kann, ohne, dass ein Duplikat erstellt wird.

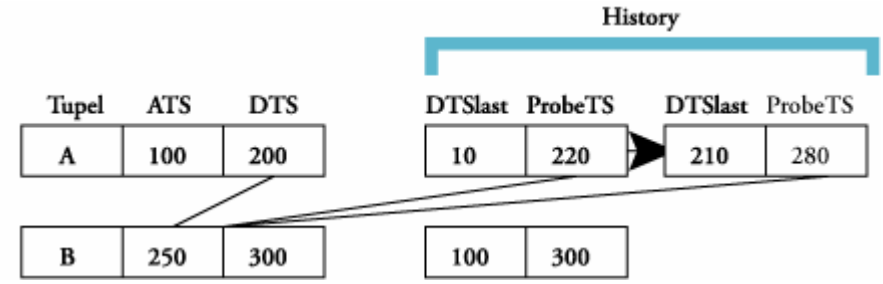

Abbildung 4 Überprüfung, ob ein Join in Phase 3 erlaubt ist, ohne Duplikate zu erstellen

Abbildung 4 veranschaulicht dieses Vorgehen. ATS und DTS überschneiden sich nicht, deshalb kam es in Phase 1 nicht zu einem Verbund der beiden Tupel. Um zu prüfen, ob die Tupel in Phase 2 verbunden wurden, werden die History-Listen genauer betrachtet. An der oberen History-Liste kann abgelesen werden, dass Tupel A bereits einmal mit Tupeln aus dem Hauptspeicher verbunden wurde. Zwar war Tupel A zum Zeitpunkt 10 noch nicht auf der Festplatte vorhanden und wurde so beim ersten Verbund zum Zeitpunkt 220 noch nicht mit eingebunden, doch beim zweiten Verbund zum Zeitpunkt 280 war auch Tupel A beteiligt, denn der DTS von Tupel A, nämlich 200, ist kleiner als der DTSlast Eintrag in der History mit 210. Da Tupel B zum Zeitpunkt des Verbunds (280) noch im Hauptspeicher war, ist klar, dass diese beiden Tupel in Phase 2 bereits miteinander verbunden wurden. Zwar ist der XJoin somit großen Datenmengen gewachsen und kann in der ersten Phase schnell Teilergebnisse liefern, er funktioniert aber nur mit 2 Datenströmen als Eingabe. Der Mjoin [UrFr99] hat diesen Nachteil nicht, er funktioniert auch mit mehreren Eingabeströmen.

6. Datenstrom-Management-Systeme (DSMS)

Um einen Datenstrom zu überwachen, bzw. Anfragen an einen Datenstrom zu stellen, braucht der Benutzer ein DSMS. Diese DSMS stellen dem Benutzer eine oftmals SQL-ähnliche Anfragesprache zur Verfügung um die entsprechenden Anfragen formulieren zu können. Zumeist erhält der Benutzer seine Anfrageergebnisse auch als Datenstrom. Dies hängt aber ganz vom verwendeten DSMS und den Einstellungen des Benutzers ab.

Da man generell zwischen Punkt- bzw. Tupelströmen und XML-Strömen unterscheidet und diese auch eine völlig verschiedene Bearbeitung benötigen, wird diese Unterscheidung auch bei den DSMS getroffen. AURORA und STREAM sind zwei der bekanntesten Vertreter für Punkt und Tupelströme, SPEX hingegen bearbeitet mit Hilfe von XPATH XML-Ströme.

6.1 Tupelstromsysteme

Zwar werden in Zukunft wohl mehr und mehr die XML-Ströme und somit deren DSMS Verbreitung finden, weil durch XML eine Strukturierung der Daten möglich ist, dennoch sind für viele Zwecke Punkt- und Tupelströme ausreichend. Wissenschaftliche Messdaten beispielsweise benötigen keine weitere Strukturierung und können effizienter als Tupel bearbeitet werden. Zwei der bekanntesten und in der Entwicklung am weitesten vorangeschrittenen DSMS für Tupelströme sind AURORA und STREAM.

6.1.1 AURORA

Aurora wird als Kollaboration der Brandeis University, der Brown University sowie des MIT entwickelt. Die Aufgabe von AURORA ist es, ankommende Datenströme genau so zu bearbeiten, wie es vom Systemadministrator festgelegt wurde [CCC+02]. Dazu benutzt AURORA das *Box-and-Arrow*-Paradigma. Die ankommenden Daten fließen durch einen azyklischen, gerichteten Graphen und werden so von Operatoren (Boxen) bearbeitet. Abbildung 5 zeigt ein solches Box-and Arrow-Modell des AURORA Systems. Von links kommen die Eingabeströme, die an die verschiedenen Operatoren der unterschiedlichen Anfragen weitergeleitet werden.

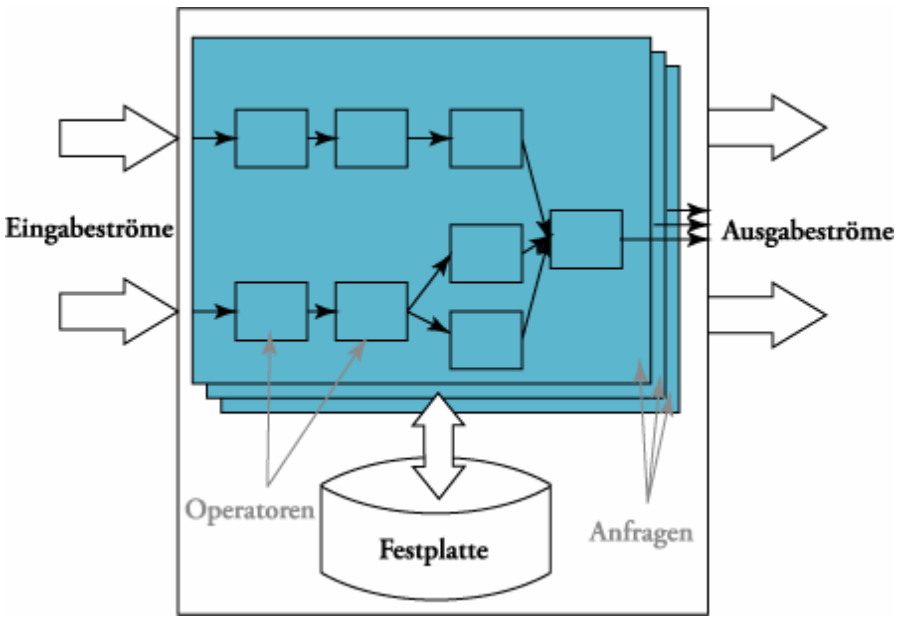

Abbildung 5 Modell des AURORA Systems

Diese Operatoren, in der Abbildung als Boxen bzw. Rechtecke dargestellt, verarbeiten und verändern die Datenströme, um so ein Ergebnis erzeugen zu können. Um Ad-hoc-Anfragen bearbeiten zu können ist es AURORA möglich, alte Daten zu speichern und zu verwalten. Daher findet auch ein Datenaustausch mit der Festplatte statt. Benötigte Daten können zur Anfrageauswertung herangezogen werden, oder für spätere Auswertungen abgelegt werden. Auf der rechten Seite sind die Ausgabeströme des Modells zu erkennen. Sie stellen die Ergebnisse der Anfragen dar und werden an die anfragenden Applikationen bzw. den anfragenden Benutzer weitergeleitet.

SQuAl steht für **S**tream **Qu**ery **Al**gebra. Es handelt sich dabei um die Anfrage-Algebra von AURORA. Sie besteht aus sieben primitiven Operatoren, von denen viele Ähnlichkeiten haben mit Operationen aus relationalen Anfragesprachen. Trotzdem unterscheiden sie sich an fundamentalen Stellen, um die speziellen Erfordernisse der Bearbeitung von Datenströmen zu erfüllen.

- *Filter:*

  Dieser Operator spaltet den Eingabestrom in verschiedene Ausgabeströme, je nachdem welches Prädikat das ankommende Tupel erfüllt.

- *Map:*

  Der Map Operator ist ein verallgemeinerter Projektionsoperator der Relationenalgebra. Er wendet eine benutzerdefinierte Funktion auf den Eingabestrom an und ändert so in den meisten Fällen auch das Schema des Stroms.

- *Union:*

  Eingehende Datenströme mit gemeinsamem Schema können mit dem Union Operator zu einem Datenstrom verbunden werden.

- *BSort:*

  Mit dem BSort Operator ist es möglich, einen Eingabestrom angenähert zu sortieren. Da eine vollständige Sortierung eines unendlichen Eingabestroms mit eingeschränktem Speicherplatz und bei eingeschränkter Zeit nicht möglich ist, wird dieses effiziente und Speicherplatz sparende Verfahren eingesetzt. Die ankommenden Tupel werden in einem Puffer gespeichert. Ist der Puffer gefüllt, wird jeweils das größte bzw. kleinste sich im Puffer befindliche Element ausgegeben.

- *Aggregate:*

  Um etwa die Berechnung einer SQL-ähnlichen Aggregat-Operation auf einem Sliding Window durchzuführen wird diese Operation benutzt. Ein Tupel des Eingabestroms fällt in ein solches Fenster, wenn seine Attribute in Wertebereichen liegen, die in der Aggregate-Operation festgelegt wurden. Da AURORA automatisch jedem eintreffenden Tupel einen einmaligen Zeitstempel zuordnet, wird dieser oftmals als Attribut für die Bestimmung der Zugehörigkeit zu einem Sliding Window benutzt. Ein Beispiel dieser Anwendung wird in [ACC+03] aufgezeigt: ein stündlicher Durchschnittspreis einer Aktie.

- *Join:*

  Der Join-Operator ist ein binärer Join und entspricht im Allgemeinen dem Join einer Relationenalgebra. Der Unterschied liegt nur darin, dass dieser Join nicht über alle Daten ausgeführt werden kann, da diese im Normalfall nicht vollständig gespeichert sind. Bei AURORA wird dieser Join deshalb über Sliding Windows ausgeführt.

- *Resample:*
  Ähnlich einem Join Operator kombiniert der Resample Operator Tupel von zwei verschiedenen Eingabeströme. Die benutzen Ströme haben meist eine thematische Ähnlichkeit. Da sie aber im Normalfall nicht gleichzeitig Tupel senden, können Tupel des einen Stroms dazu verwendet werden, einen interpolierten Wert des anderen Stroms aus einem Sliding Window heraus zu erzeugen.

AURORA unterstützt im Gegensatz zu STREAM (siehe Abschnitt 6.1.2) neben kontinuierlichen Anfragen auch Sichten und Ad-hoc-Anfragen. Die Bearbeitung von Datenströmen in AURORA basiert auf QoS Spezifikationen, einer Spezifikation der jeweiligen Dienstgüte. Jedem Ausgabestrom, der produziert wird, werden QoS Graphen bezüglich seines Nutzens unter Performanz- und Qualitätssichtweisen, zugeordnet.

Die Anfragen werden bei AURORA über ein GUI erstellt, in dem der Benutzer selbst seine Anfragepläne konstruiert. Dazu hat er die Möglichkeit, die vorgestellten Operatoren als Boxen geeignet anzuordnen.

6.1.2 STREAM

STREAM steht für **ST**anford st**RE**am dat**A** **M**anager und ist ein DSMS der Stanford University zur Bearbeitung kontinuierlicher Anfragen über kontinuierliche Datenströme und gespeicherte Relationen. Nach [ABB+03b] wird ein Datenstrom als das Paar <Tupel, Zeitstempel> modelliert. Eine Relation hingegen ist eine Sammlung von Tupeln unterschiedlicher und schwankender Zeitpunkte, die aktualisieren, löschen und einfügen unterstützt.

Zur Formulierung von Anfragen kommt bei STREAM eine Erweiterung von SQL zum Einsatz: die CQL (continuous query language) [ABB+03b]. Um die parallele Verarbeitung von Relationen und Datenströmen zu ermöglichen, existieren CQL-Operatoren, die einen Strom auf eine Relation bzw. eine Relation auf einen Strom abbilden können. CQL-Operatoren lassen sich in drei Klassen unterteilen:

- *Relation-zu-Relation Operatoren*
  SQL wird von CQL als relationale Anfragesprache benutzt. Somit sind theoretisch auch alle SQL Konstrukte zur Anfragekreierung möglich, jedoch unterstützt die aktuelle Version von STREAM noch nicht den vollen Sprachumfang.
- *Strom-zu-Relation Operatoren*
  Um Tupel aus Strömen zu entnehmen, wird eine Sprache zur Fensterspezifikation verwendet. Diese Fensterverfahren unterscheidet man in tupelbasierte, zeitbasierte und partitionierte Verfahren. Bei tupelbasierten Fenstern befindet sich eine festgelegte Anzahl von Tupeln in diesem Fenster. Bei einem zeitbasierten Fenster hingegen erstrecken sich die darin enthaltenen Tupel über ein bestimmtes Zeitintervall. Partitionierte Fenster sind mit einer GROUP BY Anweisung aus SQL vergleichbar. Sie enthalten nur Daten, die ein bestimmtes Attribut erfüllen.
- *Relation-zu-Strom Operatoren*
  Um Relationen auf einen Strom abzubilden existieren drei verschiedene Operatoren: einen, der für jede Einfügeoperation auf der Relation die entsprechenden Tupel als Strom ausgibt, einen, der für jedes gelöschte Tupel in der Relation dieses Tupel an den Strom anhängt und einen, der alle Tupel der Relation als Datenstrom ausgibt. Diese Operatoren werden IStream, DStream und RStream genannt, was für Insert-Stream, Delete- Stream und Relation-Stream steht.

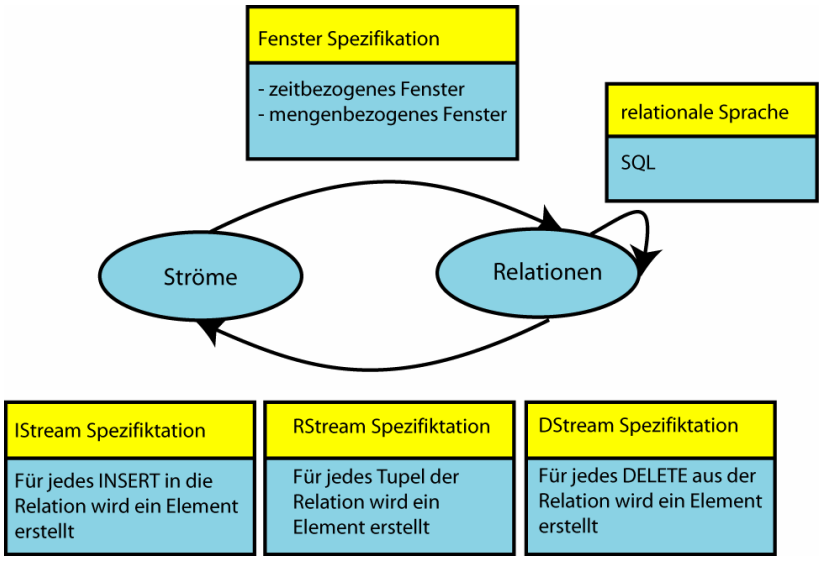

Abbildung 6 Zusammenspiel der Operatoren

Das Zusammenspiel dieser verschiedenen Operatoren verdeutlicht Abbildung 6. Um einen Strom auf eine Relation abzubilden, werden Fenster-Spezifikationen benutzt. Um von Relationen zu Strömen zu gelangen, werden Einfüge- bzw. Löschströme benutzt. Um nur auf Relationen zu arbeiten, wird wie aus DBMS bekannt, mit einer relationalen Sprache gearbeitet. In den meisten Fällen ist das SQL. Ein Operator, der nur auf Strömen operiert existiert nicht. Hier muss zuerst mit einem Fenster eine Relation erstellt und diese dann mit IStream, DStream oder RStream wieder in einen Strom verwandelt werden.

Ist eine kontinuierliche Anfrage mit CQL spezifiziert und bei STREAM angemeldet worden, wird sie in einen Anfrageplan übersetzt, der kontinuierlich im System läuft und aus drei Typen von Komponenten aufgebaut ist:

- Anfrage-Operatoren, entsprechend den Operatoren, die gerade vorgestellt wurden.
- Inter-Operator-Warteschlangen, welche den Output von Operatoren puffern, der noch als Input für weitere Operatoren dient
- Synopsen, die den zwischenzeitlichen Status von Operatoren speichern, falls dieser für künftige Berechnungen noch benötigt wird

Abbildung 7 zeigt einen solchen Anfrageplan über zwei Ströme R und S. Er beinhaltet einen Join Operator, zwei Synopsen und drei Warteschlangen. Abgebildet wurde die Anfrage

```
SELECT *
FROM R, S
WHERE R.name = S.name
```

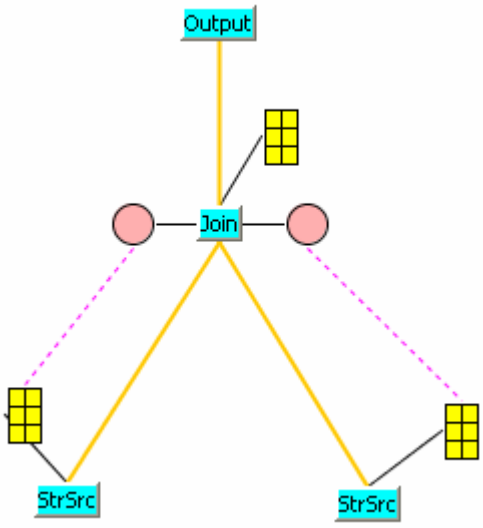

Abbildung 7 STREAM Anfrageplan

Erstellt wurde dieser Anfrageplan vom STREAM Visualizer [Stre01], einer Anwendung, die es ermöglicht CQL Anfragen zu formulieren, auszuführen und den entsprechenden Anfrageplan darzustellen. Die Ausführung des Anfrageplans wird von einem globalen Scheduler überwacht. Sobald der Scheduler aufgerufen wird, ruft er einen Operator auf, der ausgeführt werden soll. Als Parameter wird diesem seine maximale Ausführungszeit übermittelt, die er arbeiten darf. Ist die Eingabeschlange schon früher leer, wird sofort zum Scheduler gewechselt, spätestens aber nach Ablauf der maximalen Bearbeitungszeit. Neben den Zielen eines Schedulers in einem herkömmlichen DBMS, wie etwa der Maximierung des Durchsatzes oder

der Minimierung der Laufzeit, ist für einen Scheduler vor allem auch das Hauptspeichermanagement wichtig. Durch die völlig unterschiedlichen und unkontrollierbaren Eingangsraten eines Datenstroms kann es nämlich notwendig sein, in Zeiten hohen Datenverkehrs einen Teil dieser Daten zu puffern, um diesen später bearbeiten zu können.

Um diesem Problem gerecht zu werden, wird im STREAM-Projekt ein so genanntes *Chain-Scheduling* eingesetzt. Es beruht darauf, dass die Anfragepläne in disjunkte Ketten (Chains) von aufeinander folgenden Operatoren zerlegt werden. Bevorzugt werden hier Operatoren, die viele Tupel pro Zeitintervall aufnehmen und verarbeiten, aber nur wenige als Ausgabe haben. Allerdings hat diese Methode nicht nur Vorteile: Das Verhungern von Anfragen und schlechte Antwortzeiten in Zeiten mit viel Datenverkehr gehören zu den schwerwiegendsten Nachteilen, an denen derzeit noch gearbeitet und geforscht wird.

6.2 XML-Stromsysteme

Wie bereits angesprochen, haben XML Ströme im Gegensatz zu Tupel- und Punktströmen eine unterschiedliche, und zum Teil große Schachtelungstiefe. Daraus resultiert eine schwierigere Verarbeitung dieser Ströme. Da XML aber mehr und mehr Bedeutung für den Datenaustausch im Internet gewinnt, sind sie dennoch von großer Wichtigkeit.

Ein erster Ansatz eines Algorithmus für die Verarbeitung von XML-Dokumenten ist das Document Object Model (DOM) [W3C00]. Dieses beschreibt, wie man programmiersprachenunabhängig auf ein XML-Dokument zugreifen kann. Jedoch sieht es im Normalfall vor, dass vor der Verarbeitung das gesamte XML-Dokument im Hauptspeicher repräsentiert wird, was es für große Dokumente, wie z.B. medizinischen Messdaten, und vor allem für unendliche Ströme unbrauchbar macht, weil der Hauptspeicherbedarf enorm ist.

Die hier vorgestellten strombasierten Algorithmen sind effizienter im Umgang mit Speicher und besitzen zusätzlich den Vorteil der progressiven Verarbeitung, der Möglichkeit der Erzeugung von Teilergebnissen vor Auswertung aller Eingabedaten.

6.2.1 XPath

Die Adressierung von Teilen eines XML-Dokuments ist die Hauptaufgabe von XPath. Der Name XPath ist durch die genutzte Pfad-Notation (engl. path), mit der die hierarchische Struktur eines XML-Dokuments zu adressieren ist, entstanden [Beck02]. Jedes XML-Dokument wird als Baum aus verschiedenen Knoten modelliert. Der Baum zu folgendem Beispiel [BFM+01] ist in Abbildung 8 dargestellt.

```
<journal>
    <title>databases</title>
    <editor>anna</editor>
    <authors>
        <name>anna</name>
        <name>bob</name>
    </authors>
    <price />
</journal>
```

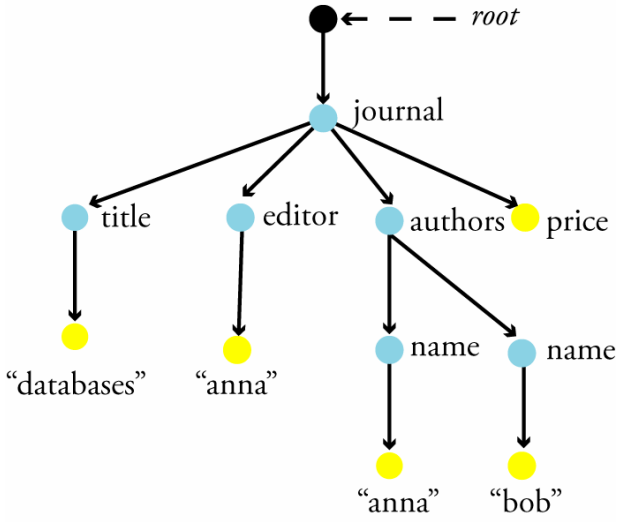

Abbildung 8 Baum aus XML-Dokument

Die Knoten des Baums werden mittels eines Lokalisierungspfades adressiert. Dieser Lokalisierungspfad besteht aus mehreren Schritten. Die Knotenmenge wird in jedem Schritt relativ zu einem Kontextknoten ausgewählt. Unter einem Kontextknoten versteht man den Knoten, der über einen solchen Ausführungsschritt ausgewählt wurde und jetzt als Ausgangspunkt für den neuen Schritt dient. Nimmt man *journal* als Kontextknoten und wendet den Lokalisierungspfad

```
child::*
```

auf ihn an, so werden *title*, *editor*, *authors* und *price* ausgewählt.

Ein Lokalisierungspfad besteht aus einem bzw. mehreren Lokalisierungsschritten, welche durch das „/“ Zeichen getrennt werden, wobei die Schritte von links nach rechts zusammengesetzt werden. Ein solcher Lokalisierungsschritt, auch location step genannt, setzt sich zusammen aus einer Achse, die die Beziehung von ausgewähltem Knoten zum Kontextknoten beschreibt, einem Knotentest, der den Knotentyp spezifiziert und optional von Prädikaten, die die ausgewählte Knotenmenge weiter verfeinern können.

Beispielsweise enthält der Lokalisierungspfad

```
parent::ktest[position()=1]
```

die Achse *parent*, den Knotentest *ktest* und ein Prädikat *[position()=1]*. Die Knotenmenge, die durch den Lokalisierungspfad ausgewählt wird, ergibt sich durch die Anwendung der Prädikate auf die durch Achse und Knotentest bestimmten Knoten.

Nun kommt es aber bei Rückwärtsachsen in XML-Strömen zu Problemen, denn der Kontextknoten kommt erst nach den Knoten an, welche zur ausgewählten Knotenmenge gehören würden und nicht mehr verfügbar sind. Deswegen müsste eine Pufferung des XML-Ausdrucks erfolgen, was so gesehen genau dem DOM entsprechen würde. Auch hier wäre wieder ein Hauptspeicherproblem die Folge.

Vorwärtsachse	Rückwärtsachse
child	parent
descendant	ancestor
following-sibling	preceding-sibling
following	preceding
descendant-or-self	ancestor-or-self

Abbildung 9 XPath Achsen

Eine weitere Möglichkeit bietet hier aber der XPath-Rewriter [BFM+01]. Das wichtigste Konzept des XPath-Rewriters ist die Achensymmetrie der XPath-Achsen. So ist zum Beispiel die child-Achse, die die Kinder des Kontextknotens enthält, symmetrisch zur parent-Achse, die den Elternknoten des Kontextknotens enthält. Eine weitere Symmetrie besteht zwischen der descendant-Achse, die alle Nachkommen, also das Kind, die Kinder des Kinds usw., des Kontextknotens enthält und der ancestor-Achse, die die Vorfahren des Kontextknotens enthält. Grafisch dargestellt sind diese beiden Beisiele in Abbildung 10. Abbildung 9 enthält noch einmal alle Paare symmetrischer Achsen.

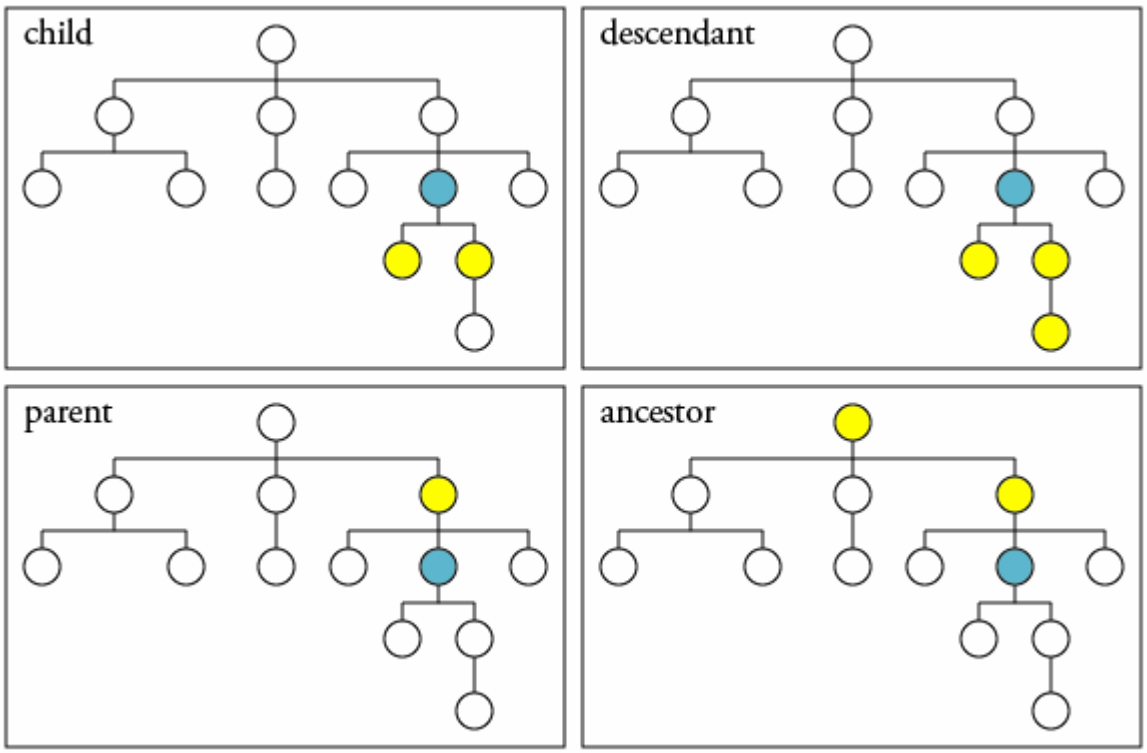

Abbildung 10 Achsensymmetrie

Ziel des XPath-Rewriters ist es, alle vorhandenen Rückwärtsachsen, die eine Pufferung des gesamten XML-Dokuments erfordern würden, in entsprechende Vorwärtsachsen umzuwandeln, so dass keine Speicherung des Stroms mehr von Nöten ist. Man spricht dann von ForwardXPath-Anfragen [BrFO04].

6.2.2 SPEX

SPEX (Streamed and Progressive Emulator for XPATH) [BrFO04] kann ForwardXPath-Anfragen gegen XML-Ströme auswerten. Unter ForwardXPath-Anfragen versteht man XPath-Anfragen, die keine Rückwärtsachsen wie etwa preceding oder parent enthalten. Um eine solche Anfrage auszuwerten, wird aus ihr ein Netzwerk aus endlichen Automaten generiert. Abbildung 11 zeigt ein solches Automatennetzwerk. Jeder dieser Automaten verfügt über einen Kellerspeicher, aus dem er immer das oberste Element lesen kann. Am Anfang eines solchen Netzwerks steht immer ein „in"-Automat, welcher den Eingabestrom ins Netzwerk einspeist und den Anfang des Stroms annotiert. Der Strom wird von den folgenden Automaten entweder unverändert oder ebenfalls annotiert weitergeleitet. Durch die Annotationen und die Verwendung der Speicher der Automaten ist es möglich die relative Position der einzelnen XML-Token zu bestimmen. Jedes XML-Token entspricht einem SAX-Ereignis [SAX98]. SAX steht für *Simple API for XML*. Es handelt sich dabei um eine XML-Schnittstelle. Zwar wurde sie anfangs für Java entwickelt, jedoch stehen mittlerweile für so viele Programmiersprachen Bibliotheken zur Verfügung, dass SAX als Standard in diesem Bereich angesehen wird. Ein Abschnitt eines XML-Stroms könnte beispielsweise so aussehen:

```
 <impressum>
     Version 1.0 of the Simple API for XML (SAX), created
     collectively by the membership of the XML-DEV mailing
     list, is hereby released into the public domain.
</impressum>
```

Die Verarbeitung durch SAX erfolgt sequentiell. Ein wahlfreier Zugriff wie bei DOM ist also nicht möglich. Bei der Verarbeitung des Beispiels werden drei Ereignisse ausgelöst.. Die Ereignisse werden bei der öffnenden Markierung *<impressum>,* beim Text und bei der schließenden Markierung *</impressum>* ausgelöst. Bei allen ausgelösten Ereignissen wird der jeweilige Eventhandler aufgerufen und ausgeführt. Es kann also für jedes auftretende Ereignis ein spezielles Vorgehen spezifiziert werden.

Am Ende jedes Netzwerks steht der „out"-Automat, der die potentiellen Ergebnisse puffert, bis jeweils entschieden werden kann, ob alle zugehörigen Prädikate erfüllt werden.

Des Weiteren befindet sich für jeden Lokalisierungsschritt des Lokalisierungspfades ein entsprechender Automat im Netzwerk. So ist es möglich, dass der Datenstrom das Netzwerk nur einmal sequentiell durchlaufen muss. Durch die Kenntnis der relativen Position der XML-Elemente ist es für den Kellerautomaten möglich, genau einen Lokalisierungsschritt durchzuführen und die sich ergebende Menge an Kontextknoten entsprechend zu vermerken, damit auch der nächste Automat seinen Lokalisierungsschritt ausführen kann.

Der Anfrageplan zu der Anfrage

```
/descedant::a[child::b[descendant::d or child::e]]following-sibing::c
```

ist in Abbildung 11 dargestellt[BrFO04]. Der „in"- sowie der „out"-Automat, die in jedem Netz vorhanden sein müssen sind blau hervorgehoben. Bei dem gelb markierten Rechteck handelt es sich um einen so genannten Kopf. Er kann jeweils nur ein einziges Mal in einem Netzwerk vorkommen. Die sich durch diesen Lokalisierungsschritt ergebende Menge von Knoten entspricht direkt einer potentiellen Ergebnismenge, die allerdings noch von weiteren Prädikaten abhängig sein kann.

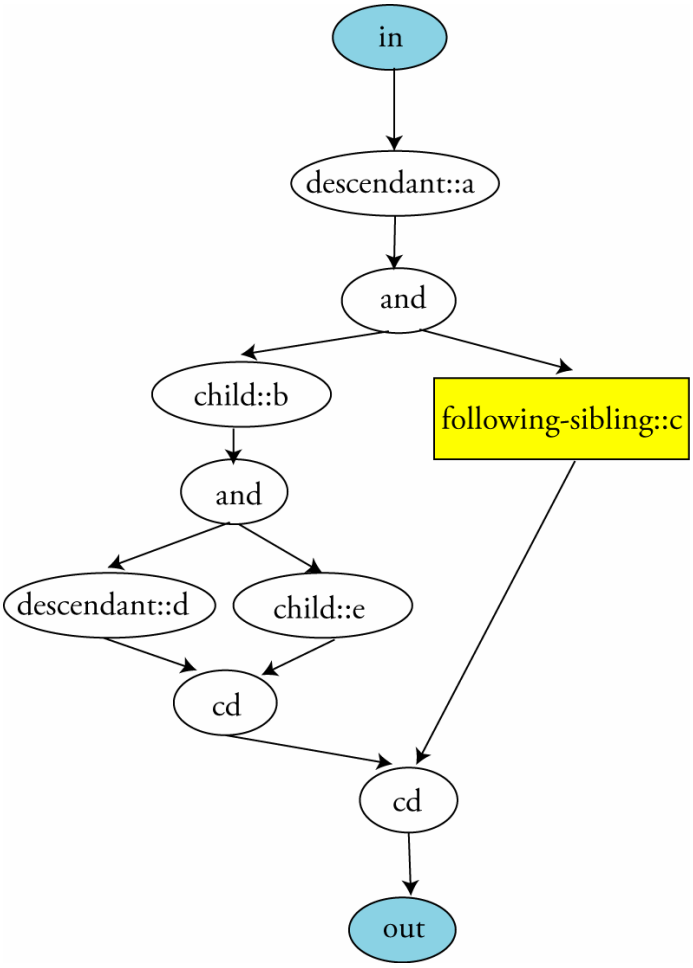

Abbildung 11 physischer Anfrageplan

Durch die Möglichkeit einen Automaten mehrmals zu verwenden sind umfassende Optimierungsmöglichkeiten gegeben. Erweiterungen sind recht einfach durch das schlichte Hinzufügen von Automaten möglich, weil dadurch, wegen der Unabhängigkeit untereinander keine anderen Automaten beeinflusst werden.
Die Erstellung eines solchen Anfragenetzwerks und die Bearbeitung des XML-Stroms durch SPEX lässt sich in vier Schritte untergliedern. Im ersten Schritt wird, wie bereits beschrieben, die XPath-Anfrage in eine ausschließlich vorwärts gerichtete Anfrage umgeschrieben. Im zweiten Schritt wird aus dieser Anfrage dann ein logischer Anfrageplan erstellt. Auf Basis dieses logischen Anfrageplans wird dann in Schritt drei ein physischer Anfrageplan, wie in Abbildung 11 abgebildet, durch Erweiterungen, wie Unterscheidungsoperatoren oder Operatoren zum Sammeln der Ergebnisse, erstellt. Im vierten und letzten Schritt wird der XML-Datenstrom dann kontinuierlich durch den physischen Anfrageplan, dem Automatennetzwerk, bearbeitet. Außerdem wird fortlaufend ein Ausgabestrom generiert, der die entsprechenden Anfrageergebnisse beinhaltet.

Der SPEX-Viewer ermöglicht die Visualisierung einiger dieser Schritte. Er zeigt das Umschreiben der XPath-Anfragen in Anfragen ohne Rückwärtsachsen, das Netzwerk der Automaten, das auf Basis der Anfrage generiert wird, die inkrementelle Abarbeitung des XML-Stroms und die fortlaufende Erzeugung von Anfrageergebnissen.

7. Load Shedding

Eine der großen Aufgaben von DSMS ist die Echtzeitverarbeitung eines Datenstroms trotz Ressourcenbeschränkungen wie CPU, Hauptspeicher oder Bandbreite. Ein plötzliches und unvorhergesehenes Anschwellen des Datenstroms könnte nämlich bei der Verarbeitung des Stroms zu Latenzzeiten führen. Um zu späte Ergebnisse zu verhindern, müssen Teile des Stroms auf kontrollierte Art und Weise verworfen werden. Unter *Load Shedding* versteht man den Prozess des Verwerfens von Daten, wenn die benötigten Ressourcen größer als die Systemkapazitäten sind. Man spricht von Anpassung des Systems an Überlast. Überlast bedeutet, dass mehr Daten ankommen als verarbeitet oder gespeichert werden können. Das kann zum einen an einer zu langsamen CPU, zum anderen an einem zu kleinen Hauptspeicher für die, in diesem Moment ankommenden Datenmengen liegen. Der schlimmste Fall ist, wenn beide Probleme gleichzeitig auftreten. Es müssen Vorkehrungen getroffen werden, um trotzdem repräsentative Ergebnisse der Auswertung zu erhalten.

Allerdings hat ein solches Vorgehen des Verwerfens von Daten auch einen Nachteil: Zwar werden die benötigten Ressourcen durch das Verwerfen von ankommenden Daten niedriger, aber auch die Genauigkeit der Ergebnisse wird niedriger. Deshalb ist ein Hauptziel der Forschungen zum Thema Load Shedding die Minimierung des Genauigkeitsverlustes.

Aurora behandelt Datenströme mit einem Anfrage Netzwerk, einer Ansammlung von kontinuierlichen Anfragen, die aus Sequenzen von Operatoren bestehen. Ist Load Shedding auf Grund mangelnder Ressourcen von Nöten, werden Drop-Operatoren in dieses Netzwerk eingefügt. Zwar gibt es zwei verschiedene Arten von Drop-Operatoren, zufällige und semantische, trotzdem funktionieren beide nach demselben Prinzip: Sie liefern weniger Tupel zurück, als sie erhalten. Zufällige Drop-Operatoren verwerfen Teile des ankommenden Datenstroms nach dem Zufallsprinzip, semantische Drop-Operatoren hingegen berücksichtigen benutzerdefinierte Quality-of-Service (QoS) Funktionen. So soll die Wichtigkeit der Elemente für die Anfragen berücksichtigt werden. Der Latency-Graph repräsentiert den Nutzen des Tupels gegenüber seiner Verzögerungszeit. Je länger eine Verarbeitung dauert, je größer die Verzögerung ist, desto geringer ist der Nutzen des Tupels für den Benutzer. Der Value-Based-Graph repräsentiert den jeweiligen Nutzen der Tupel gegenüber ihren Werten. Beispielsweise sind Messwerte innerhalb der Toleranzen weit weniger aufschlussreich und nützen dem Benutzer weniger als Werte außerhalb des Toleranzbereichs. Der Loss-Tolerance-Graph hingegen repräsentiert den durchschnittlichen Nutzen von jedem Ausgabe Tupel gegenüber dem Verlust-Parameter. Er zeigt an, dass die Genauigkeit und somit die Aussagekraft eines Stroms geringer wird, je größer der prozentuale Anteil an verworfenen Tupeln ist. Der Algorithmus, der von Aurora benutzt wird, wählt demnach auf Basis der benutzerdefinierten QoS-Funktionen Tupel zum Verwerfen aus, die eine möglichst geringe Bedeutung für die Auswertung haben, um so die Genauigkeit des Ergebnisses möglichst hoch zu halten, obwohl zwangsläufig nicht alle Eingabedaten ausgewertet werden konnten.

Generell gibt es drei Fragen, die ein Load-Shedding-Algorithmus berücksichtigen muss: Wann, wo im Anfrage-Netzwerk und wie viel des ankommenden Datenstroms soll verworfen werden?

Um herauszufinden, wann Teile des Stroms verworfen werden müssen, wird die Systemlast fortlaufend überwacht. Sobald die Systemlast die vorhandenen Kapazitäten überschreitet, werden die Drop-Operatoren eingefügt um den Überschuss an Daten zu verwerfen. Fällt die Last wieder, müssen die Drop-Operatoren wieder entfernt werden.

Die Entscheidung, wo die überschüssigen Tupel verworfen werden sollen, wird vor allem durch zwei Entscheidungsfaktoren, die einen bestimmten Punkt im System besser oder schlechter machen, bestimmt: Zum einen ist das die maximale Last Minderung, zum anderen ein minimaler Gesamtnutzenverlust. Die überschüssigen Teile des Stroms sollen an einem Punkt gelöscht werden, an dem ein möglichst kleiner Genauigkeitsverlust des Ergebnisses auftritt und eine möglichst große Lastminderung stattfindet. Um die bestmöglichen Stellen im System herauszufinden, wird für jeden Platz ein Verhältnis aus Verlust von Genauigkeit / Verminderung der Last berechnet. Je kleiner dieses Verhältnis ist, desto besser ist es für das System.
Überlappend mit der „wo"-Entscheidung wird die „wie viel"-Entscheidung getroffen. Die überschüssigen Tupel, die zu verwerfen sind, werden bei zufälligen Drop-Operatoren in Prozent ausgedrückt, bei semantischen Drop-Operatoren muss zusätzlich noch entschieden werden, welche Tupel gelöscht werden. Dazu nimmt man eine Filterfunktion, die auf dem Value-Based-QoS aufsetzen. Die Tupel, die Werte in einem bestimmten Intervall mit niedrigem Nutzen haben, werden verworfen.

8. Zusammenfassung

Die Eigenschaften von Datenströmen unterscheiden sich elementar von denen eines Relationenmodells. Daher ist es nicht möglich bekannte Techniken einfach zu übertragen. Datenströme sind potentiell unendlich und fluktuieren in ihrem Aufkommen. Daher ist es im Normalfall nicht möglich den ganzen Strom zu speichern. Teilweise ist es nicht einmal möglich alle Tupel zu bearbeiten.

Anfragen an Datenströme werden nicht vollständig ausgewertet, sondern liefern möglichst schnell Teilergebnisse, die ebenfalls als Datenstrom weitergegeben werden. Die Anfragen verweilen im System. Bei traditionellen Datenbanksystemen hingegen, werden die Anfragen einmalig und vollständig ausgewertet, bevor ein genaues Ergebnis ausgegeben wird. Genaue Ergebnisse sind bei Datenströmen nicht immer möglich. Ist das Datenaufkommen zu hoch, helfen Load-Shedding-Techniken, dennoch gute angenäherte Ergebnisse zu liefern.

Werden Ad-hoc-Anfragen gestellt, oder Anfragen, die etwa Sortier- oder Aggregat-Operatoren beinhalten, können diese nur mit Hilfe so genannter Sliding Windows beantwortet werden. Diese beinhalten Daten eines gewissen Zeitraums, einer bestimmten Menge oder eines speziellen Attributs.

9. Literaturverzeichnis

[ABB+03a] A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, R. Motwani, C. Olston, J. Rosenstein. R. Varma, Query Processing, Resource Management, and Approximation in a Data Stream Management System. CIDR Conference, 2003: 1- 12

[ABB+03b] A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thoms, R. Varma, J. Wiom, STREAM: The Stanford Stream Data Manager. IEEE, 2003: 1- 5

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom: Models and Issues in Data Stream Systems. PODS 2002: 1-24

[BCD+05] F. Bry, F. Coskun, S. Durmaz, T. Furche, D. Olteanu, M. Spannagel, The XML Stream Query Processor SPEX. ICDE, 2005: 1-4

[Beck02] O. Becker, XML Path Language (XPath) Version 1.0 Deutsche, kommentierte Übersetzung, 2002. Elektronisch verfügbar: http://www.obqo.de/w3c-trans/xpath-de-20020226

[BFM+01] F. Bry, T. Furche, H. Meuss, Dan Olteanu: Symmetrie in XPath. Forschungsbericht PMS-FB-2001-16, 2001, 1-5

[BrFO04] F. Bry, T. Furche, D. Olteanu: Datenströme. Forschungsbereicht PMS-FB-2004-2, 2004: 1-6

[CCC+02] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. B. Zdonik: Monitoring streams - a new class of data management applications. VLDB Conference 2002: 1-12

[CCT+03] U. Cetintemel, M. Cherniack, N. Tatbul, S. Zdonik, Load Shedding on Data Streams. MPDS, 2003: 1-2

[Mour03] K. I. Mouratidis, Data Stream Processing: An Overview of Recent Research. 2003, 1- 32

[SAX98] Mitglieder der XML-DEV mailing list, 1998. Elektronisch verfügbar: http://www.saxproject.org/

[Stre01] STREAM Group, STREAM Visualizer, 2001. Elektronisch verfügbar: http://skate.stanford.edu:8080/server?

[UrFr99] T. Urhan, M. J. Franklin, XJoin: Getting Fast Answers From Slow and Bursty Networks. UMIACS-TR-99-13 1999, 1-12

[W3C00] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne: Document Object Model (DOM) Level 2 Core Specification, 2000. Elektronisch verfügbar: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113