Analytische Bestimmung einer Datenallokation für Parallele Data Warehouses

Thomas Stöhr

Universität Leipzig, Institut für Informatik, Postfach 920, D-04009 Leipzig
stoehr@informatik.uni-leipzig.de
http://dbs.uni-leipzig.de

Zusammenfassung. Die stark wachsende Bedeutung der Analyse von Data Warehouse-Inhalten und bequemere Anfrageschnittstellen für Endbenutzer erhöhen das Aufkommen an OLAP-Queries signifikant. Bei der Reduktion des Arbeitsumfanges und dem Erreichen kurzer Antwortzeiten für diese komplexen Anfragen ist neben der Nutzung von Verarbeitungs- und I/O-Parallelität eine adäquate Datenallokation der Schlüssel zu guter Leistungsfähigkeit. Allerdings ist die Bestimmung einer geeigneten Fragmentierung und Allokation für große Datenmengen, wie sie z.B. in Form von Faktentabellen oder Indexstrukturen in relationalen Sternschemas vorliegen, ein schwieriges Problem. Hierfür existiert heutzutage praktisch keine Werkzeugunterstützung. Wir präsentieren daher einen Ansatz zur analytischen Bestimmung einer passenden multi-dimensionalen, hierarchischen Datenallokation. Unser Ansatz dürfte recht einfach in ein Werkzeug zur automatischen Unterstützung des Allokationsproblems integriert werden können.

1 Einleitung

Data Warehouses integrieren massive Datenmengen aus vielen, heterogenen Datenquellen. In diesem Umfeld werden komplexe, analytische Anfragen, insbesondere OLAP- (On-line Analytical Processing-) Queries, durchgeführt [3]. Auch um der gesteigerten Nutzerzahl und dem Mehrbenutzerbetrieb Rechnung zu tragen, bedarf es eines kombinierten Einsatzes verschiedener leistungssteigernder Ansätze, wie z.B. voraggregierte Daten, spezielle Indexstrukturen (Bitmaps) und dem Einsatz von Parallelität. Die in Data Warehouses gespeicherten Datenmengen erreichen häufig den Multi-TeraByte-Bereich, wodurch der Erreichbarkeit angemessener Performance-Ziele für solche Anfragen und insbesondere der Bestimmung einer adäquaten *Datenallokation* in Parallelen Datenbanksystemen (PDBS) eine Schlüsselbedeutung zukommt [8].

Der Datenallokationsprozeß für ein Data Warehouse ist eine äußerst komplexe Aufgabe. Bei den häufig zugrundeliegenden *relationalen, denormalisierten Sternschemas* müssen unterschiedlich skalierte und referenzierte Allokationseinheiten (sehr umfangreiche Faktentabellen, kleine Dimensionstabellen, umfangreiche Indexstrukturen), DBMS-Spezifika, Hardware-Parameter etc. und nicht zuletzt Lastbalancierungsaspekte

berücksichtigt werden. Aufgrund der großen Datenmengen muß hier eine Reduktion des Arbeitsaufwandes für die typischen aggregierenden Anfragen auf Sternschemas, sog. *Star Queries*, verbunden mit einer angemessenen Unterstützung durch I/O- und Anfrageparallelität im Vordergrund stehen.

Aufgrund der Komplexität und Bedeutung dieser Aufgabe für die Performance wäre eine entsprechende Tool-Unterstützung oder eine automatische Bestimmung einer Datenallokation enorm hilfreich. Leider gibt es heutzutage praktisch noch keine Werkzeuge für den Data Warehouse- oder Datenbankadministrator, um den Datenallokationsprozeß zu unterstützen oder gar (P)DBS, die Self-Tuning-Fähigkeiten auf diesem Sektor entwickelt haben.

Diese Lücke gilt es zu füllen. Wir stellen in diesem Papier eine Methode zur analytischen Bestimmung einer Datenallokation für parallele Data Warehouses vor. Die Überlegungen basieren auf der Anwendung einer multi-dimensionalen, hierarchischen Fragmentierungsstrategie *MDHF* [29], welche den entsprechenden inhärenten Charakter relationaler Schemas ausnutzt. Durch *MDHF* kann häufig eine signifikante Reduktion des Arbeitsaufwands für alle Star Queries erreicht werden, die mindestens eine Dimension des Sternschemas referenzieren, welche auch in der Fragmentierung involviert wird. Dabei muß, im Gegensatz zu anderen Strategien, die Fragmentierungsdimension nicht exakt auf dem Fragmentierungsattribut referenziert werden, um diesen Vorteil zu erzielen. Durch eine fragmentbezogene Allokations- und Verarbeitungsstrategie unterstützen wir gleichzeitig die Ausnutzung von sowohl Inter- und Intra-Query- als auch I/O-Parallelität. Wir beziehen uns insbesondere bei der Speicherung von Fragmenten auf Platten auf den Einsatz einer Shared-Disk-(SD-)/Shared-Everything-(SE-) Architektur, weil diese durch die Möglichkeit des Zugriffs von allen Verarbeitungsknoten auf alle Platten eine höhere Allokationsflexibilität bei geringeren Kommunikationskosten im Vergleich zu Shared-Nothing-(SN-)Systemen ermöglichen [23, 25, 26, 30]. Unser Ansatz kann allerdings mit geringem Aufwand auch auf eine SN-Umgebung adaptiert werden.

Mit Hilfe eines in diesem Papier vorgestellten Kostenmodells bestimmen wir diejenige(n) Fragmentierung(en) bzw. Fragmentierungsattribute, welche für einen gegebenen Star Query-Mix minimalen Arbeitsaufwand und minimale Antwortzeit ermöglichen. Wir wenden eine entsprechende kombinierte Metrik an, welche Aufwands- und Antwortzeitziele, die häufig gegenläufige Effekte bewirken, gemeinsam berücksichtigt. Wir quantifizieren mit Hilfe unseres Modells, wie *MDHF* die Query-Verarbeitung beeinflußt und bestimmen daraus eine passende Fragmentierung. Das vorgestellte Modell läßt sich in ein Werkzeug zur automatischen Bestimmung einer passenden Fragmentierung und Allokation für parallele Data Warehouses integrieren, welches wir zur Zeit entwickeln.

Das Papier gliedert sich wie folgt: nach einem Überblick über verwandte Arbeiten (Abschnitt 2) skizzieren wir in Abschnitt 3 ein Sternschema, Star Query-Typen und Indexverfahren, welche wir als Beispielszenario für unsere Überlegungen verwenden. In Abschnitt 4 präsentieren wir die Fragmentierungsstrategie *MDHF* und erläutern unser fragmentbasiertes Anfrageverarbeitungs- und Datenallokationsmodell, welches sowohl Inter- und Intra-Query- als auch I/O-Parallelität unterstützt. In Abschnitt 5 präsentieren wir unser Kostenmodell und die Metrik zur Abschätzung des Aufwandes und der Ant-

wortzeiten für OLAP-Anfragen unter *MDHF*. Abschnitt 6 zeigt anhand eines Beispiels die Bestimmung einer passenden Fragmentierung mit Hilfe unseres Modells. Anschließend fassen wir zusammen und schließen mit einen Ausblick auf zukünftige Aktivitäten.

2 Verwandte Arbeiten

In relationalen PDBS basiert die Datenallokation üblicherweise auf einer horizontalen Fragmentierung von Tabellen, typischerweise als *Round-Robin-*, *Hash-* oder *Bereichsfragmentierung* [8, 24]. Bei Round Robin werden die Tupel einfach gemäß ihrer Einfügereihefolge verteilt, während bei Hash- und Bereichsfragmentierungen eine entsprechende Partitionierungsfunktion auf den Fragmentierungsattributen angewendet wird. Es wurden Studien über solche logischen, i.d.R. 1-dimensionalen Fragmentierungsstrategien durchgeführt, vorwiegend für die SN-Architektur [7, 10, 17]. Eine solche Strategie erlaubt eine Reduktion des Arbeitsaufwandes lediglich für Anfragen auf dem Fragmentierungsattribut, welches auch eine Komposition mehrerer Attribute sein kann, d.h. der multi-dimensionale und hierarchische Charakter von Sternschemas wird nicht berücksichtigt. In [16] wurde ein analytisches Modell zur Bestimmung eines optimalen Parallelitätsgrades für Scan- und Join-Anfragen, basierend auf einer einfachen Allokation, für den Einbenutzerbetrieb vorgestellt, welches ebenfalls eine SN-Architektur voraussetzte.

In [11] wurde eine „echte“ multi-dimensionale Strategie vorgeschlagen, die allerdings den hierarchischen Charakter einer Sternschemaumgebung nicht berücksichtigt. Auch [31] beschränkt sich auf SN-Systeme und nutzt hierarchische Attribute nicht aus.

Einige Studien haben sich mit den Aspekten *physischer* Datenallokation auf Block-Level beschäftigt, um I/O-Parallelität zu unterstützen [6, 14, 15, 28]. Es wurden analytische Modelle entwickelt, um für eine I/O-Last adäquate Parallelitätsgrade zu bestimmen [15, 28]. Allerdings erlaubt eine physische Datenverteilung typischerweise keine Vorhersage bzgl. zu referenzierender Platten durch den Query-Optimierer. Dies kann ggf. zu Plattenkonkurrenz durch Subqueries führen. Weiterhin kann der Zugriff auch nicht auf eine Untermenge der Partitionen begrenzt werden, wie es für logische Fragmentierungen der Fall ist. Damit können diese Ansätze nur eingeschränkt zur Bestimmung einer Allokation für Sternschemas herangezogen werden.

Die Notwendigkeit von Self-Tuning-Fähigkeiten für DBMS wird aktuell propagiert [5]. Es existieren bereits Implementierungen, welche dynamisch Indexe bzw. materialisierte Sichten vorschlagen [1, 4]. Für den Datenallokationsbereich, auch im Data Warehouse-Sektor, existiert nach unserem Wissen noch keine Lösung.

Kommerzielle PDBS. Die meisten kommerziellen PDBS bieten spezielle Unterstützung für Data Warehousing wie Bitmap-Indexe und Star Queries, z.B. ORACLE8i [22], IBM DB2 UNIVERSAL DATABASE (UDB) [2], RED BRICK WAREHOUSE [27] und INFORMIX DYNAMIC SERVER [12]. INFORMIX und ORACLE ermöglichen zudem eine Reihe von Fragmentierungsoptionen, welche auch eine multi-dimensionale Bereichsfragmentierung umfassen [13, 21].

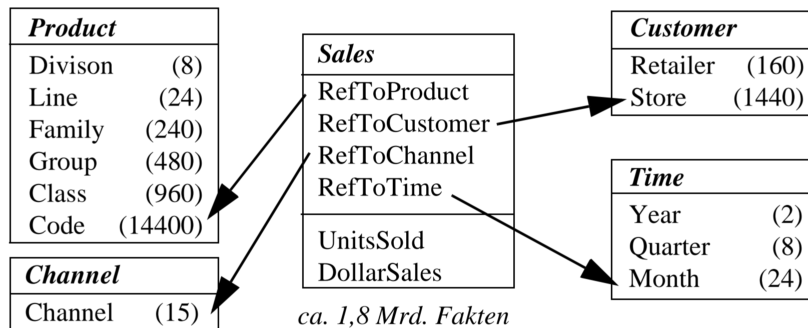

Abb. 1. Betrachtetes Sternschema (abgeleitet aus APB-1 Benchmark-Spezifikation)

Leider bleibt hier weitgehend unklar, wie diese multi-dimensionale Fragmentierung eingesetzt werden kann, um die Query-Arbeit zu reduzieren. Die hierarchische Struktur der Dimensionsattribute wird ebenfalls nicht ausgenutzt. Keiner der genannten Anbieter bietet eine ausreichende Hilfestellung oder gar Werkzeugunterstützung, um eine Datenallokation zu bestimmen. ORACLE z.B. bietet aufgrund seiner SD-Architektur zwar das Potential, Indexstrukturen unabhängig von Tabellenstrukturen zu partitionieren und zu allozieren, beschränkt sich aber auf zu allgemeine Hinweise in den Manuals, wie dies auszunutzen sei.

3 Zugrundeliegendes Sternschema

3.1 DB-Schema und Queries

Für unsere weiteren Betrachtungen beziehen wir uns auf ein relationales Sternschema, das wir aus der Spezifikation des Application Processing Benchmark 1 (APB-1) des OLAP Council abgeleitet haben [19]. Das Schema repräsentiert eine typische mehrdimensionale Struktur zur Verwaltung von Verkaufskennzahlen und umfaßt die Dimensionen *Product*, *Customer*, *Channel* und *Time*. Abb. 1 zeigt das Schema und die Kardinalitäten der Attribute (in Klammern). Alle Dimensionen sind *hierarchisch* organisiert. So werden z.B. alle Produkte in 8 Divisionen (*Division*) eingeteilt, die wiederum jeweils 3 Produktlinien umfassen (insgesamt 24 *Lines*) usw. Die Faktentabelle *Sales* speichert die Faktenattribute *UnitsSold* und *DollarSales*, welche auf nichtaggregierter Stufe vorliegen. Ein Faktum verwaltet also die entsprechenden Verkaufszahlen, den Umsatz etc. für eine Kombination eines *Code*, *Channel*, *Month* und *Store*. Die Verknüpfungen zwischen Faktentabelle und Dimensionstabellen werden über entsprechende Fremdschlüssel (*RefToProduct*, ...) realisiert. Bzgl. der Skalierung orientieren wir uns ebenfalls an der APB1-Spezifikation. Dort wird ein Dichtefaktor für die Faktentabelle angenommen, welcher den Anteil der gespeicherten Fakten an der Anzahl maximal möglicher Kombinationen festlegt. Unsere Konstellation umfaßt einen Zeitraum von 2 Jahren und eine Anzahl von ca. 1,8 Mrd. Fakten.

Wir unterstellen typische Star Queries, welche Fakten über eine oder mehrere Dimensionen auf verschiedenen Hierarchieebenen aggregieren. Eine Beispielanfrage $Q_{GroupMonth}$ (in SQL-Notation)

```
SELECT SUM(UnitsSold),SUM(DollarSales)
FROM Sales S, Product P
WHERE   S.RefToTime= MONTH
AND     P.Group= PRODUCTGROUP
AND     S.RefToProduct = P.Code
```

stellt also ein 2-dimensionale Anfrage dar, welche *UnitsSold*, *DollarSales* für *eine Produktgruppe* innerhalb *eines Monats* aggregiert. Die im Query-Namen vermerkten Hierarchieattribute verweisen dabei auf die referenzierten Hierarchieebenen.

Im Folgenden konzentrieren wir uns auf Queries ohne Gruppierung, so daß ein expliziter Join zwischen Fakten- und Dimensionstabelle zur Bestimmung der Gruppenelemente vermieden wird (z.B. Bestimmung aller *Product Classes* innerhalb einer *Group*). Der zusätzliche Aufwand wäre auch, verglichen mit Index- und Faktentabellenverarbeitung (s.u.), vernachlässigbar.

3.2 Indexe

Als Zugriffstrukturen unterstellen wir die in Data Warehouse-Umgebungen vorherrschenden *Bitmap-Indexe* (BMI) [20]. Üblicherweise enthält eine Bitmap eines BMI pro Attributwert ein Bit, welches anzeigt, ob das assoziierte Tupel diesen Wert besitzt oder nicht (*Standard*-Bitmaps). Für Attribute hoher Kardinalität nutzen wir zwei Erweiterungen dieses Konzeptes, um Verarbeitungs- und Speicheraufwand zu sparen. Zum einen sind dies *kodierte* Bitmaps, bei denen herkömmliche Verfahren die N Ausprägungen eines Attributes durch $\lceil \log_2(N) \rceil$ Bits (und Bitmaps) kodieren. Dies reduziert die Anzahl zu speichernder Bitmaps signifikant, erhöht allerdings den Verarbeitungsaufwand, da alle Bitmaps jeder Anfragedimension zunächst dekodiert werden müssen. Eine Verbesserung stellt die *hierarchische* Kodierung dar [32], bei der die Werte innerhalb einer Dimension *relativ zum nächsthöheren Hierarchie-Level* kodiert werden. Sollen z.B. Fakten einer bestimmte Produktgruppe per Bitmap bestimmt werden, so würden 2 + 3 + 2 + 3 = 10 Bitmaps (statt maximal 15) gelesen[1]: es existieren 4 *Groups* pro *Family*, welche durch $\lceil \log_2(4) \rceil = 2$ Bit kodiert werden. Pro *Line* existieren wiederum 5 *Families*, welche 3 Bits zur Kodierung benötigen usw. Dies erhöht die Anzahl der zu speichernden Bitmaps unwesentlich gegenüber dem nichthierarchischen Verfahren, reduziert aber den Leseaufwand, da die Bitmaps der Attribute zu feineren Hierarchiestufen – bezogen auf den Anfrage-Level – *nicht* dekodiert werden müssen. Für unser APB1-Beispielschema erreichten wir für die Dimensionen höherer Kardinalität *Customer* und *Product* eine Reduktion von 15.840 Standard-Bitmaps über alle Hierarchieebenen auf insgesamt lediglich 27 kodierte Bitmaps. Für die *Time*- und *Channel*-Dimensionen unterstellen wir Standard-Bitmaps.

[1] Durch das Kodierungsverfahren werden für insgesamt ca. 16,000 Attributwerte aller Hierarchieebenen (14,400 Produkt-Codes, 960 Produktklassen usw.) 15 Bitmaps erzeugt

Als weitere Optimierung nutzen wir Bitmaps, welche das Ergebnis vorausgeführter Joins (Bitmap Join Indices, [20]) zwischen Fakten- und Dimensionstabelle darstellen. Hierbei zeigt ein Bit einer Bitmap an, ob ein Attribut eines Faktentupels dem Wert eines *Dimensionsattributs*, also einer bestimmten Hierarchieebene, entspricht. So ist zur Bestimmung von Fakten z.B. einer Produktgruppe, die in unserem Beispiel nichtaggregiert auf der tieferen *Product::Code*-Ebene repräsentiert sind, kein expliziter Join zwischen Fakten- und Dimensionstabelle notwendig, falls die entsprechende vorgenerierte Bitmap vorliegt.

4 Multi-dimensionale, hierarchische Fragmentierung

In [29] haben wir eine *multi-dimensionale, hierarchische Fragmentierungsstrategie (MDHF)* vorgestellt, welche die für Sternschemas inhärente multi-dimensionale, hierarchische Datenorganisation zur signifikanten Reduktion des Arbeitsaufwandes einer großen Menge typischer Star Queries ausnutzt. Bei diesem Ansatz werden ein oder mehrere *Fragmentierungsattribute* aus der Menge der Attribute der Dimensionstabellen ausgewählt, wobei aus jeder Dimensionstabelle *maximal ein Attribut* verwendet wird. Jedes Attribut repräsentiert eine Hierarchiestufe einer involvierten Dimension (vgl. Abb. 1). Auf jedem dieser Attribute kann anschließend eine *Bereichsverteilung* spezifiziert werden. Die so gebildeten Fragmente umfassen damit genau die Menge der Tupel, welche bzgl. ihrer Attributwerte genau einem solchen (mehrdimensionalen) Wertebereich zugeordnet sind.

Zur Vereinfachung wurden sog. „Punkt“fragmentierungen angenommen, bei denen jedes Werteintervall eines Fragmentierungsattributs genau einen Attributwert umfaßt. Dies vereinfacht die Bestimmung geeigneter Wertebereiche und damit das Fragmentierungsproblem erheblich. Hierarchie-Attribute bezeichnen wir in der Form '*d::h*', wobei *d* die Dimension bezeichnet und *h* die Hierarchieebene. So bezeichnet z.B. eine Fragmentierung $F_{MonthGroup} = \{Time::Month, Product::Group\}$ eine 2-dimensionale Fragmentierung auf der Zeit- und Produktdimension, deren Attribute *Month* bzw. *Group* als Fragmentierungsattribute ausgewählt wurden. Durch $F_{MonthGroup}$ werden einem Fragment genau die Fakten zugeordnet, welche bzgl. ihrer Attributwerte *einen bestimmten Monat* und *eine bestimmte Produktgruppe* repräsentieren. Die Fragmente umfassen implizit auch die Tupel, die sich auf einer im Vergleich zum Fragmentierungsattribut *feineren Hierarchiestufe* der Fragmentierungsdimensionen befinden, z.B. die einer Produktgruppe assoziierten Produktklassen und denen zugeordneten Produkt-Codes gemäß der Hierarchie aus Abb. 1. Die Anzahl der generierten Fragmente ergibt sich bei Punktfragmentierungen aus dem Produkt der Kardinalitäten der entsprechenden Fragmentierungsattribute.

4.1 Query-Verarbeitung unter MDHF

Betrachten wir ein Beispiel, bei dem die Anfrage $Q = Q_{QuarterCode}$, welche die Fakten bezogen auf ein einzelnes Produkt über den Zeitraum eines Quartals aggregiert, unter unserer Beispielfragmentierung $F = F_{MonthGroup}$ ausgeführt wird (Abb. 2). *F* erzeugt

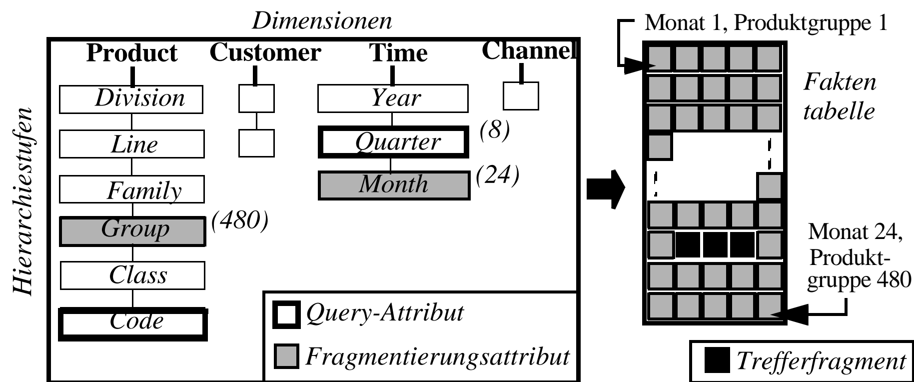

Abb. 2. Query $Q_{QuarterCode}$ unter Fragmentierung $F_{MonthGroup}$

11520 Fragmente (480 Produktgruppen x 24 Monate). *Q* benötigt unter *F* lediglich die Verarbeitung von 3 Fragmenten. Alle *Product::Code*-bezogenen Tupel befinden sich innerhalb der Fragmente, die mit der übergeordneten Produktgruppe assoziiert sind. Bzgl. der Zeitdimension müssen pro Produktgruppe lediglich die 3 „Monatsfragmente" verarbeitet werden, die das betrachtete Quartal umfaßt. Somit wird eine Reduktion der Anzahl der zu verarbeitenden Fragmente um den Faktor 480 x 8 (480 Produktgruppen x 8 Quartale) = 3840 erreicht.

MDHF ermöglicht die Reduktion der zu verarbeitenden Fragmente für alle Queries, welche auf *mindestens einer der Fragmentierungsdimensionen* arbeiten. Es ist dabei nicht erforderlich, daß eine Query exakt das Fragmentierungsattribut nutzt: Zugriffe auf Attribute der Fragmentierungsdimensionen, die eine feinere Hierarchiestufe repräsentieren, erreichen denselben Reduktionsfaktor wie Zugriffe auf den Fragmentierungsattributen. Zugriffe auf höheren Stufen bringen auch eine Reduktion, wenn auch im geringeren Umfang. Zudem werden Anfragen entlang *aller* durch die Fragmentierung und Query involvierten Dimensionen durch Arbeitsreduktion unterstützt, *ohne Beachtung der Reihenfolge* der Fragmentierungsattribute.

Trotz der Einschränkung auf lediglich 3 zu verarbeitende Fragmente ist die o.g. Beispielfragmentierung nicht unbedingt ein „gute" Fragmentierung. Die Verarbeitung weniger Fragmente birgt ein zu geringes Parallelisierungspotential. Den Trade-off zwischen den Zielen (i) Reduktion des Arbeitsaufwandes durch Clustering der Treffer in *wenigen* Fragmenten und (ii) der Reduktion der Antwortzeit durch Parallelverarbeitung auf *vielen*, verteilten Fragmenten gilt es bei der Wahl der Fragmentierung Rechnung zu tragen (s. 5.3).

Neben *MDHF* unterstellen wir eine fragmentorientierte Anfrageverarbeitung. Eine Query gemäß 3.1 wird dabei in *Subqueries* aufgeteilt, welche jeweils ein Faktenfragment und dessen assoziierte Bitmap-Fragmente verarbeiten. Diese lesen, falls sinnvoll, zunächst die Bitmap-Fragmente *parallel* ein, bestimmen durch Boole'sche Verknüpfungen die Lokalisation der Treffer im Faktenfragment und lesen die relevanten Daten von der Platte, um diese nachfolgend zu aggregieren. Damit werden sowohl Inter- als auch Intra-Query-Parallelität unterstützt.

4.2 Allokation

Die Allokation der Faktentabelle(n) und Bitmaps erfolgt ebenfalls fragmentorientiert. Dimensionstabellen werden aufgrund ihrer geringen Größe (insgesamt < 1 MB) nicht weiter betrachtet. Sämtliche Fragmente werden per *Round-Robin*-Strategie konsekutiv auf Platten alloziert. Ein solches Verfahren erreicht i.d.R. eine gut balancierte I/O-Last und eine hohen Verteilgrad zur Unterstützung paralleler Anfragen. Um bzgl. der Bitmap-Verarbeitung zusätzlich I/O-Parallelität zu erzielen, werden alle Bitmaps um je eine Platte versetzt („staggered") gespeichert [29]. Somit können alle für ein Faktenfragment relevanten Bitmap-Fragmente parallel von unterschiedlichen Platten gelesen werden. Diese Allokation kann insbesondere von SD-/SE-Systemen ohne zusätzlichen Kommunikations-Overhead genutzt werden, da hier alle Prozessoren auf allen Platten arbeiten können. Sie ist aber auch für SN-Systeme mit geringem Adaptionsaufwand anwendbar.

5 Kostenmodell

In diesem Abschnitt stellen wir ein Kostenmodell sowie eine Metrik vor, mit deren Hilfe wir die Güte einer Fragmentierung für Star Queries bzgl. des resultierenden *Arbeitsaufwandes* und der *Antwortzeiten* bewerten. Queries, welche nicht auf voraggregierte Daten bzw. materialisierte Sichten zurückgreifen können, werden durch die durchzuführende I/O dominiert. Daher diskutieren wir zunächst den Einfluß von *MDHF* auf das I/O-Verhalten und entwickeln anschließend Formeln, welche die Teilkomponenten unserer Metrik bestimmen. Wir legen dabei das im vorigen Abschnitt vorgestellte Verarbeitungs- und Datenallokationsmodell nebst existierender Bitmap-Join-Indexe für alle Hierarchieattribute zugrunde. Anschließend definieren wir unsere Metrik auf Basis der hergeleiteten Kostenformeln.

5.1 Einfluß von MDHF auf die Query-I/O

Star Queries lassen sich bzgl. ihrer I/O-Anforderungen unter *MDHF* grob in zwei *I/O-Klassen, IOC1* und *IOC2, einteilen*:

IOC1. Dieser Klasse werden sämtliche Queries zugeordnet, welche bzgl. der Hierarchieebenen der Fragmentierungsattribute ausschließlich *gleiche* oder *höhere* Hierarchie-Level referenzieren. Für eine Query $Q \in IOC1$ ergeben sich folgende Eigenschaften mit Bezug auf die I/O-Kosten:

- Treffer liegen innerhalb der Fragmente (und den entsprechenden DB-Seiten) *geclustert* vor
- *Bitmap*-Zugriff ist *nicht* erforderlich, da Treffer aufgrund des Clustering nicht per Bitmap lokalisiert werden müssen
- Eine Einschränkung der Fragmentzahl wird auf *jeder* durch *Q* referenzierten Dimension erreicht

Durch das Clustering wird eine sehr gute I/O-Effizienz, d.h. ein gutes Verhältnis erzielter Treffer zu gelesenen DB-Seiten, erreicht. Desweiteren wird I/O eingespart, da keine Bitmaps einzulesen sind.

Ein Vertreter dieser Klasse ist z.B. die Anfrage $Q_{LineQuarter}$ unter der Fragmentierung $F_{MonthGroup}$, welche Fakten bezogen auf eine Produktlinie (*Line*) innerhalb eines Quartals aggregiert. Da sämtliche Tupel bezogen auf eine Produktgruppe und einen Monat innerhalb eines Fragmentes gebündelt vorliegen, muß diese Query 20 x 3 = 60 Fragmente verarbeiten (eine *Line* umfaßt 20 *Groups,* ein Quartal 3 Monate). Referenziert Q exakt die Fragmentierungsattribute, so wird genau ein Fragment verarbeitet. Dies stellt zwar den minimalen Arbeitsaufwand für eine Query dar, es ergibt sich aber ein ungenügendes Parallelisierungspotential.

IOC2. Diese Klasse umfaßt alle übrigen Queries. $Q \in IOC2$ referenziert daher *mindestens eine Dimension* auf einer Hierarchieebene, welche *feiner* als der entsprechende Fragmentierungs-Level ist. Insbesondere beinhaltet dies den Fall, daß Q eine Dimension involviert, welche F *nicht* umfaßt. Die Treffer liegen somit nicht mehr geclustert vor. So sind z.B. bei Zugriff auf *Product::Code* unter $F_{GroupMonth}$, gemäß der Kardinalitäten aus Abb. 1, lediglich $1/30$ der Tupel eines Fragmentes bzw. einer DB-Seite Treffer[2]. Die I/O-Eigenschaften von $Q \in IOC2$ sind folgendermaßen zu charakterisieren:

- Durch fehlendes Clustering sind die Treffer über eine größere Anzahl Seiten (und Fragmente) verteilt, d.h. die I/O-Effizienz sinkt.
- Bitmap-Zugriff kann notwendig/sinnvoll werden, um die Treffer innerhalb der Fragmente bzw. Seiten zu lokalisieren und damit Verarbeitungs- und I/O-Aufwand auf der Faktentabelle zu sparen.
- Teilen sich Q und F keine Dimensionen, so erfolgt auch keine Reduktion der Anzahl zu verarbeitender Fragmente für Q. Gegebenenfalls sind auch alle für eine Query benötigten Bitmaps vollständig zu lesen, um die Treffer zu lokalisieren. Dieser „teure“ Fall sollte vermieden werden, da die Query-Antwortzeiten in die Größenordnung derjenigen von „Full Scans“ steigen.

5.2 Bestimmung der I/O-Zugriffszeiten

In diesem Abschnitt leiten wir die Formeln zur Bestimmung von I/O-Zugriffszeiten für eine Query unter einer Fragmentierung her, welche die Basisgrößen zur Bewertung von Fragmentierungen in unseren Metriken darstellen. Die wesentlichen zu bestimmenden Größen sind in Tab. 1 aufgeführt.

Tab. 1. Größen zur Bestimmung der I/O-Zugriffszeiten (jeweils für *Q* unter *F*)

F_{FT}	Anzahl zu verarbeiteter Faktentabellenfragmente
$NoBM$	Anzahl zu verarbeitender Bitmaps
$FrgAcc_{FT}$	Zugriffszeit für die Verarbeitung eines Faktentabellenfragments
$FrgAcc_{BM}$	Zugriffszeit für die Verarbeitung eines Bitmap-Fragments

[2] Annahme der Gleichverteilung von Treffern auf Fragmente und DB-Seiten.

$FrgAcc_{FT}$ bzw. $FrgAcc_{BM}$ liefern die Gesamtzugriffszeiten für ein Faktentabellen- bzw. Bitmap-Fragment unter Berücksichtigung der Anzahl der durchzuführenden I/Os pro Fragment und der Zugriffskosten für eine I/O mit oder ohne Nutzung von *Prefetching*. Wie diese Größen in die Bewertungsmetrik eingehen, zeigt 5.3.
Wir unterstellen folgende Basisannahmen für unser Modell:

- Wir verwenden die in [19] eingeführte Hierarchiestufe *TOP*, ein Wurzelelement für jede Dimension *dim*. Die Kardinalität dieses Elementes wird mit 1 angenommen (*card(dim::TOP)* = 1). Die Nichtberücksichtigung einer Dimension durch *Q* oder *F* entspricht der Involvierung auf *TOP*-Level. Dies reduziert die Anzahl der Fallunterscheidungen bzgl. der Abhängigkeiten von *Q* und *F* und erhöht so die Lesbarkeit.
- Die Treffer einer Query werden als über die Fragmente und DB-Seiten *gleichverteilt* angenommen (kein Daten-Skew).
- Wir betrachten Punktfragmentierungen und -Queries. Somit bleibt der Berechnungaufwand auch bei umfangreicheren Starschemas gering.

Tab. 2 zeigt die relevanten Parameter und Größen für unseren Algorithmus sowie abgeleitete Basisgrößen, die in den folgenden Abschnitten eingesetzt werden. Die (wenigen) benötigten Eingabeparameter bzw. die gewählte Fragmentierung als Ausgabe des Algorithmus sind kursiv hervorgehoben. *S* beschreibt ein Dimensionenschema, wobei jede Dimension durch ein Tupel aus seinen Hierarchieattributen in der Form '*dimension::hierarchy*' beschrieben wird.

Plattenzugriff. Wir schätzen die Zugriffskosten für einen einzelnen Plattenzugriff (*disk access*) ab durch

$$Dacc(pf) = pos + tra \cdot pf \tag{1}$$

pf quantifiziert das *Prefetching-Granulat*, also die Anzahl Seiten, die pro I/O gelesen werden. Wird kein Prefetching genutzt, so ist *pf* = 1 zu setzen. *pos* bezeichnet die mittlere Plattenpositionierungszeit (seek time, rotational delay, settle time), *tra* die Zugriffszeit für eine DB-Seite inkl. Kontroller- und Übertragungszeit. Je höher *pf* gewählt wird, desto größer die I/O-Effizienz. Allerdings ist zu beachten, daß speziell für den Mehrbenutzerbetrieb ein einzelner I/O-Job nicht zu groß ausfallen sollte, um die Plattenwartezeiten zu reduzieren. Daher haben wir in unseren späteren Experimenten eine Prefetching-Obergrenze angenommen. *pf* sollte für Faktentabellen- und Bitmap-Zugriffe unterschiedlich gewählt werden können, da sich die Fragmentgrößen $PgsFrg_{FT}$ bzw. $PgsFrg_{BM}$ um Größenordnungen unterscheiden.

Bestimmung der Anzahl Fragmente. Die Anzahl zu verarbeitender Fragmente F_{FT} für *Q* unter *F* ergibt sich als

$$F_{FT} = NoFrgs_{FT} / \left(\prod_{\forall d \in Dim(S)} min(card(f_d), card(q_d)) \right) \tag{2}$$

Der Klammerausdruck bestimmt den Reduktionsfaktor bzgl. der Anzahl zu verarbeitender Fragmente, der für *Q* unter *F* erreicht werden kann. q_d bzw. f_d bezeichnen dabei das Query- bzw. Fragmentierungsattribut für Dimension *d*. Für $hier(q_d) > hier(f_d)$ erreicht der Reduktionsfaktor für Dimension *d* die Kardinalität von q_d, sonst diejenige

von f_d. Eine „echte“ Reduktion (Faktor > 1) wird auf allen Dimensionen erreicht, welche Q und F gemeinsam berücksichtigen. So erreicht $Q_{QuarterCode}$ aus Abschnitt 4.1 unter Fragmentierung $F_{MonthGroup}$ auf der Produktdimension eine Reduktion um den Faktor 480 (= min(card({*Product::Group*}), card({*Product::Code*})), auf der Zeitdimension entsprechend eine Reduktion von 24, also eine Gesamtreduktion um den Faktor 3840.

I/O-Zugriffszeiten pro Faktentabellenfragment. Diese Größe ergibt sich aus der Anzahl durchzuführender I/O pro Fragment und der Zugriffskosten pro I/O als

$$FrgAcc_{FT} = IO_{FT} \cdot Dacc(PF_{FT}) \tag{3}$$

Tab. 2. Relevante Größen / Paramter für Kostenmodell und Metriken

Schema *S*			
S	$= ((d_0::TOP, d_0:h_1, ..d_0::h_k), ..,(..), ... (d_{dmx}::TOP, ..))$		
dmx	Anzahl Dimensionen	hier(a)	Hierarchieebene von a
Dim(S)	Dimensionen von S	*card(a)*	*Attributkardinalität* $(a \in S)$
Query *Q*			
Q_{mix}	*Query-Mix* $(Q_i$*: Query,* w_i*: Gewicht) =*		$\{(Q_1, w_1), (Q_2, w_2), ..\}$,
q_d	*QA auf Dimension d*	p_{TS}	Wahrscheinlichkeit Trefferseite
$NoBM_{st}$	# verarbeitete Standard-BM	IO_{FT}	# I/Os (FT-Fragment)
$NoBM_{en}$	# verarbeitete kodierte BM	IO_{BM}	# I/Os (BM-Fragment)
NoHits	# Treffer (Gesamt)	**Datenbank & Platten**	
NoHitsFrg	# Treffer (Fragment)	*N*	*# FT-Tupel (Gesamt)*
Fragmentierung *F*		*tpp*	*# FT-Tupel (pro Seite)*
$\boldsymbol{f_d}$	***FA auf Dimension d***	*PgSize*	*DB-Seitengröße (Bytes)*
$NoFrgs_{FT}$	Anzahl FT-Fragmente	*NoDisk*	*# Platten*
$NoFrgs_{BM}$	Anzahl BM-Fragmente	*Dacc(pf)*	*Kosten I/O (pf Seiten)*
A_{en}	Kodierte Bitmap-Attribute	*pos*	*Plattenpositionierungszeit*
A_{st}	Standard-Bitmap-Attribute	*tra*	*I/O-Transfer für Seite*
$PgsFrg_{FT}$	# FT-Fragment (in Seiten)	PF_{FT}	*Prefetching-Granulat (FT)*
$PgsFrg_{BM}$	# BM-Fragment (in Seiten)	PF_{BM}	*Prefetching-Granulat (BM)*
Ableitung von Basisgrößen			
$NoHits = N / \prod_{\forall f \in DimF} card(q_d)$		$PgsFrg_{FT} = \lceil N/(tpp \cdot NoFrgs_{FT}) \rceil$	
		$NoHitsFrg = Hits/NoFrgs$	
$NoFrgs_{FT} = \prod_{\forall f \in Dim(F)} card(f_d)$		$NoFrgs_{BM} = NoFrgs_{FT}/(8 \cdot SzTplFT)$	

Abk.: FT= Faktentabelle, BM=Bitmap, FA=Fragmentierungsattribut, QA=Query-Attribut. Eingabeparameter *kursiv*, Ausgabewerte ***kursiv/fett***

Bei vorliegendem Clustering von Treffern (*IOC1*) bzw. durch den Einsatz von Bitmaps sind die Seiten, welche die für eine Query relevante Treffer enthalten (*Trefferseiten*), a priori bestimmbar. Somit können wir unterstellen, daß jede Faktentabellen-I/O mit einer Trefferseite „startet" und ggf., je nach Größe von PF_{FT} und Trefferwahrscheinlichkeit, noch weitere Trefferseiten einliest. Trefferlose Seiten können zudem häufig ausgespart werden.

IO_{FT} bestimmen wir daher aus dem Verhältnis der Abschätzung der Anzahl der *Trefferseiten* eines Fragments zu der Anzahl *Trefferseiten pro Prefetching-I/O* als

$$IO_{FT} = \frac{p_{TS} \cdot PgsFrg_{FT}}{1 + p_{TS} \cdot (PF_{FT} - 1)} \tag{4}$$

p_{TS} bezeichnet die Wahrscheinlichkeit des Ereignisses „Seite ist Trefferseite", d.h. die Wahrscheinlichkeit, daß eine Seite innerhalb eines Fragmentes *mindestens ein Treffertupel* enthält. Wir bestimmen p_{TS} mit Hilfe einer Approximation der *Yao-Formel* [9, 33], welche diese bei Gleichverteilung von t Treffern innerhalb einer Seitenmenge s mit dem Blockungsfaktor b (Anzahl Tupel pro Seite) abschätzt mit

$$p_{TS} = 1 - \overline{p_{TS}} = 1 - \left(1 - \frac{t}{b \cdot s}\right)^b \tag{5}$$

Die daraus resultierende Anzahl Trefferseiten innerhalb s Seiten wird bestimmt mit

$$s \cdot p_{TS} \tag{6}$$

$\overline{p_{TS}}$ $(= 1 - p_{TS})$ berechnet die Wahrscheinlichkeit des Auftretens einer *trefferlosen* Seite innerhalb des Fragments. Dazu wird die mittlere Wahrscheinlichkeit des Ereignisses „Tupel enthält keinen Treffer" über alle b Tupel einer Seite akkumuliert. Die Approximation geht vereinfachend von der Unabhängigkeit dieses Ereignisses von dem „Trefferzustand" der übrigen Tupel aus.

In unserem Falle gilt $t = NoHitsFrg$ und $b = tpp$. Zur Bestimmung der Anzahl Trefferseiten eines Fragmentes setzen wir $s = PgsFrg_{FT}$ in (6) ein. Zur Abschätzung der Anzahl Treffer pro Prefetching-I/O unterstellen wir die erste Seite als Trefferseite und schätzen die Anzahl zusätzlich gelesener Trefferseiten ab indem wir $PF_{FT} - 1$ für s in (6) einsetzen.

Erste Validierungsergebnisse zeigen eine sehr gute Approximation der analytisch bestimmten Anzahl von I/Os an entsprechende Simulationsresultate. Für den Spezialfall *IOC1* (Treffer-Clustering, $p_{TS} = 1$) ergibt sich $IO_{FT} = PgsFrg_{FT}/PF_{FT}$, d.h. alle Seiten eines Fragments werden in Granulaten von PF_{FT} Seiten gelesen.

I/O-Zugriffskosten für Bitmaps. Wir gehen davon aus, daß für alle Hierarchieebenen entweder Standard-Bitmaps oder, für Dimensionen hoher Kardinalität (z.B. *Product*, *Customer*), hierarchisch kodierte Bitmaps gemäß 3.2 vorliegen.

Für *IOC1-An*fragen gilt Treffer-Clustering innerhalb der Faktentabellenfragmente. Daher benötigen diese keinen Bitmap-Zugriff zum Auffinden von Treffern. Für *IOC2*-Anfragen gehen wir davon aus, daß Bitmaps genutzt werden, um die Treffer zu lokalisieren. Dies bedeutet, daß Bitmaps verarbeitet werden, falls

$$\exists q_d \in Q | hier(q_d) > hier(f_d) \quad (7)$$

gilt, d.h. mindestens eine Fragmentierungsdimension auf *feinerem Hierarchie-Level* referenziert wird als das jeweilige Fragmentierungsattribut vorgibt.

Anzahl verarbeiteter Bitmaps (NoBM). Die Anzahl zu lesender Standard-Bitmaps $NoBM_{st}$ für eine Query ist 1 Bitmap pro referenzierter Attributausprägung und Attribut $q_d \in A_{st}$, für welches (7) gilt. Für Punktanfragen resultiert dies also in 1 Bitmap pro referenziertes Attribut aus A_{st}.

Die Anzahl zu betrachtender *hierarchisch kodierter* Bitmaps $NoBM_{en}$ auf Attributen mit umfangreicher Domain ergibt sich für alle $q_d \in A_{en}$ zu:

$$NoBM_{en} = \sum_{q_d \in A_{en}} \sum_{hier(q_d) \geq h > hier(f_d)} \lceil \log_2(card(h)/card(h-1)) \rceil \quad (8)$$

A_{en} bezeichnet die Anzahl der Fragmentierungsattribute der Dimensionen, für die kodierte Bitmaps vorliegen. *h-1* bezeichnet die *nächstgröbere Hierarchiestufe* zu *h*, also gilt z.B.: *h = product::class* ⇒ *h-1 = product::group.*
Um $e = card(h)/card(h-1)$ Elemente des Level *h* relativ zu Level *h-1* zu kodieren (z.B. $e = 15$ Produkte innerhalb einer Produktklasse), werden $\lceil \log_2(e) \rceil$ Bits benötigt. Es werden Bitmaps für alle Hierarchie-Level *auf und oberhalb* des Levels der Query-Attribute und *unterhalb* des entsprechenden Fragmentierungs-Level benötigt. Insbesondere müssen bei Nichtinvolvierung einer Query-Dimension durch eine Fragmentierung die Bitmaps aller Ebenen ab Query-Level und unterhalb des TOP-Level gelesen werden.
Die Gesamtmenge der zu lesenden Bitmaps ergibt sich somit zu

$$NoBM = 0 \quad \text{für } Q \in IOC1 \quad (9a)$$

$$NoBM = NoBM_{st} + NoBM_{en} \quad \text{für } Q \in IOC2 \quad (9b)$$

Die Anzahl zu verarbeitender Bitmap-*Fragmente* $NoFrgs_{BM}$ beträgt $NoBM \cdot F_{FT}$, da pro Faktentabellenfragment und pro Bitmap je ein Bitmap-Fragment verarbeitet wird.

I/O-Zugriffszeiten pro Bitmap-Fragment. Ein Bitmap-Fragment ist um Größenordnungen kleiner als das entsprechende Faktentabellenfragment[3], so daß ein vom für Faktentabellenfragmente abweichendes Prefetching-Granulat i.d.R sinnvoll ist. Die Anzahl I/O zur Verarbeitung eines Bitmap-Fragmentes IO_{BM} mit Prefetching-Granulat PF_{BM} ergibt sich zu

$$IO_{BM} = \lceil PgsFrg_{BM}/PF_{BM} \rceil \quad (10)$$

Die Gesamt-I/O-Kosten pro Fragment betragen

$$FrgAcc_{BM} = IO_{BM} \cdot Dacc(PF_{BM}) \quad (11)$$

[3] Der Größenunterschied beträgt $8 \cdot (\lfloor PgSize/tpp \rfloor)$, da jedes Bit einer Bitmap mit jeweils einem Faktentupel der Größe $\lfloor PgSize/tpp \rfloor$ (in Bytes) korrespondiert.

5.3 Bewertungsmetrik

Das Ziel unserer Bewertung ist es, diejenigen Fragmentierungen zu bestimmen, für die ein Mix von Anfragen möglichst geringen *Aufwand* verursachen und dabei zusätzlich eine möglichst geringe *Antwortzeit* durch Parallelisierung erreichen. Da die betrachteten Star Queries durch I/O dominiert werden, konzentrieren wir uns auf die Bestimmung des *I/O-Aufwandes* und schätzen die Antwortzeit von Star Queries über ihre *I/O-Antwortzeit* ab.

Wir verwenden eine kombinierte Metrik, da die Minimierung des I/O-Aufwandes bzw. der Antwortzeiten alleine ist i.d.R. nicht ausreicht bzw. den vorhandenen *Trade-off* zwischen beiden Zielen nicht berücksichtigt. Bei einem Fokus auf die Reduktion des Aufwandes würden Fragmentierungen ermittelt, welche eine sehr gute I/O-Effizienz durch einen hohen Grad an *Treffer-Clustering* erreichen und entsprechend geringe Kosten verursachen. Die daraus resultierende *geringere Anzahl an Fragmenten* bietet aber ein niedrigeres Verteilungs- und Parallelisierungspotential, welches höhere Antwortzeiten zur Folge hat und das erzielte I/O-Reduktionspotential ggf. „verpuffen" läßt. Ein ähnliches Problem ergäbe sich für eine reine Antwortzeit-, also „Einbenutzermetrik". Hier würden ggf. Fragmentierungen bestimmt, die durch einen hohen Parallelitätsgrad zwar niedrigste Antwortzeiten erreichen, allerdings um den Preis einer höheren Gesamtbelastung der Ressourcen durch Involvierung vieler Platten. Dies würde im Mehrbenutzerbetrieb zu schlechterem Gesamtdurchsatzverhalten und mehr I/O-Umfang führen.

Wir schlagen daher eine zweistufige I/O-Metrik *IOM* vor, welche den IO-Aufwand und die durch Parallelisierung erreichbare I/O-Antwortzeit berücksichtigt. Für die kombinierte Berücksichtigung dieser Kriterien gibt es mehrere Möglichkeiten (u.a. geometrisches Mittel). Da insbesondere im Hinblick auf Mehrbenutzerbetrieb einem geringen I/O-Aufwand eine besondere Bedeutung zukommt, bestimmen wir zunächst Fragmentierungskandidaten, die den I/O-Aufwand minimieren (Metrik IOA_Q). In einem zweiten Schritt ermitteln wir aus den so ermittelten „besten" Fragmentierungen bzgl. Aufwand diejenigen, die eine minimale I/O-Antwortzeit erreichen (Metrik $IORT_Q$).

Tab. 3 zeigt die innerhalb der Metriken verwendeten Bezeichnungen. Auf die Darstellung der Abhängigkeit der Größen von *Q* und *F* haben wir aus Gründen der Übersichtlichkeit verzichtet.

Tab. 3. In Kostenmetriken verwendete Größen

IOA_Q	*Gesamt-I/O-Aufwand für Q unter F*
IOA_{Mix}	*Gesamt-I/O-Aufwand für einen Query-Mix unter F*
IOA_{FT}	*I/O-Aufwand (Faktentabelle) für Q unter F*
IOA_{BM}	*I/O-Aufwand (Bitmaps) für Q unter F*
$IORT_Q$	*Gesamt-I/O-Antwortzeit für Q unter F*
$IORT_{Mix}$	*Gesamt-I/O-Antwortzeit für einen Query-Mix unter F*
$IORT_{FT}$	*I/O-Antwortzeit (Faktentabelle) für Q unter F*
$IORT_{BM}$	*I/O-Antwortzeit (Bitmaps) für Q unter F*

IO-Aufwandsmetriken IOA_Q, IOA_{Mix}. Der Gesamtaufwand für eine Anfrage ergibt aus der Addition der I/O-Zugriffszeiten für die Faktentabellenverarbeitung (= Produkt der Anzahl Fragmente mit den Zugriffszeiten pro Fragment) und entsprechendes für die Bitmap-Verarbeitung, wobei F_{FT} Fragmente *pro relevanter Bitmap* zu verarbeiten sind. Daher gilt für eine Query Q unter F:

$$\begin{aligned} IOA_Q &= IOA_{FT} + IOA_{BM} \\ &= F_{FT} \cdot FrgAcc_{FT} + F_{FT} \cdot NoBM \cdot FrgAcc_{BM} \end{aligned} \tag{12}$$

Zur Berücksichtigung der unterschiedlichen Relevanz von Anfragetypen in einem Data Warehouse (z.B. Ausführung „teurer“ Berechnungen von Gesamtverkaufszahlen einmal im Quartal, dagegen häufigere Adhoc-Aggregierungen über ein Produkt an einem Tag usw.) haben wir entsprechende *Gewichte* in unsere Kostenformel integriert. So gilt für einen Query-Mix Q_{Mix} (vgl. Tab. 2):

$$IOA_{Mix} = \sum_{\forall Q_i \in Q_{Mix}} w_i \cdot IOA_{Q_i} \tag{13}$$

I/O-Antwortzeitmetriken $IORT_Q$, $IORT_{Mix}$. Wir nehmen an, daß die Verarbeitung der Gesamtmenge der Faktentabellen- bzw. Bitmap-Fragmente optimal I/O-parallelisierbar ist, also ein maximaler Verteilgrad der relevanten Fragmente erreicht werden kann. Dies ist i.d.R. gewährleistet, da Round Robin-Allokationen eine sehr gute I/O-Balancierung erreichen. Es gilt daher

$$\begin{aligned} IORT_Q &= IORT_{FT} + IORT_{BM} \\ &= FrgAcc_{FT} \cdot \left\lceil \frac{F_{FT}}{NoDisk} \right\rceil + FrgAcc_{BM} \cdot \left\lceil \frac{NoBM \cdot F_{FT}}{NoDisk} \right\rceil \end{aligned} \tag{14}$$

Wir schätzen die Antwortzeit einer Query für die Faktentabellenverarbeitung mit der I/O-Antwortzeit ab, da diese nahezu ausschließlich den stark dominierenden Anteil an der Antwortzeit der für uns interessanten „teuren“ OLAP Queries darstellen. Aufgrund des unterstellten optimalen I/O-Parallelisierungspotential setzen wir die Gesamt-I/O-Antwortzeit mit denen im Mittel sequentiell zu verarbeitenden Fragmente *pro Platte* gleich. Verarbeitet eine Anfrage z.B. insgesamt 150 Fragmente, die auf einer Anzahl $NoDisk = 100$ unterschiedlichen Platten alloziert sind, so berücksichtigen wir die Antwortzeit mit derjenigen für die Verarbeitung von $\lceil 150/100 \rceil = 2$ Fragmenten auf *einer* Platte. Vergleichbares gilt für die Bitmap-Verarbeitung, wobei zusätzlich die Anzahl zu bearbeitender Bitmaps *NoBM* für eine Query als Faktor einfließt. Die exakte Bestimmung der Antwortzeit ist nicht notwendig, da die Metrik nur der groben Vergleichbarkeit zwischen Alternativen dient.

Zur Bestimmung der Antwortzeit für einen Query-Mix wird deren Gewichtung in derselben Weise wie bei IOA_{Mix} berücksichtigt:

$$IORT_{Mix} = \sum_{\forall Q_i \in Q_{Mix}} w_i \cdot IORT_{Q_i} \tag{15}$$

Wir arbeiten aktuell an der Validierung des vorgestellten Modells mit Hilfe unseres Simulators für PDBS, welchen wir mit Hilfe der CSIM-Simulationsbibliothek [18] an unserem Lehrstuhl implementiert haben [29]. Dieser ist umfangreich parametrisierbar und bildet verschiedenste Architekturen (SD, SE, SN), Lastbalancierungs- und Allokationsstrategien ab. Dabei werden Lastgenerierung und -verarbeitung, Sperrverwaltung, Pufferverwaltung, Netzwerkaspekte, CPU-, Platten-Server etc. detailliert repräsentiert.

6 Bestimmung einer Fragmentierung

Abschließend wollen wir die Wirkungsweise der vorgeschlagenen Methode zur Bestimmung einer guten mehrdimensionalen Fragmentierung anhand eines Experiments illustrieren. Die wesentlichen Parameterbelegungen und Beispiel-Queries zeigt Tab. 4. (Bedeutung der Größen: vgl. Tab. 2). Als Prefetching-Granulat wählen wir von den dort aufgeführten Werten jeweils den größtmöglichen Wert, den die jeweilige Fragmentgröße für eine berechnete Fragmentierung erlaubt. Wir legen das Sternschema aus Abb. 1 mit den dort spezifizierten Kardinalitäten der Hierarchieebenen zugrunde.

6.1 Beispiel Q_{store}

Zunächst bestimmen wir Fragmentierungskandidaten für eine einzelne Query Q_{store}. Diese 1-dimensionale Anfrage referenziert alle Faktentupel, welche einem *Store* der Dimension *Customer* zugeordnet sind. Die Trefferselektivität innerhalb der Faktentabelle beträgt 0.0007, so daß ca. 1.3 Mio. Tupel gelesen werden.

Aus Gründen der Übersichtlichkeit verwenden wir eine Notation zur Bezeichnung der Fragmentierungen, welche jeweils den Hierarchie-Level pro Dimension numerisch anzeigt. Level 0 bezeichnet dabei den TOP-Level (=Nichtinvolvierung), Level 6 steht z.B. für den *Code*-Level der *Product*-Dimension. Die Reihenfolge der Dimensionen innerhalb der Notation haben wir mit *Product*, *Customer*, *Time*, *Channel* gewählt. „0 2 0 0" bezeichnet demnach *{Customer::Store}*, „4 0 3 0" *{Product::Group; Time::Month}* usw. Es sei hier betont, daß die Reihenfolge der Dimensionen aufgrund des multi-dimensionalen Charakters von *MDHF keinerlei Einfluß* auf den Effekt einer Fragmentierung auf Star Queries bzw. auf die erzielten Resultate der Metriken hat.

Tab. 4. Parameterbelegungen für Experimente zur Bestimmung einer Fragmentierung

Datenbank		**I/O**	
N	1.866.240.000	Dacc(PF)	$= 13ms + PF \cdot 1ms$
PgSize	4 KB	NoDisk	100
Größe FT (Seiten)	9.148.236 (ca. 37 GB)	PF_{FT} (in Seiten)	1, 4, 8, 16, 32

Queries		**Notation**	**Fakten**
Q1 =	{*Customer::Store*}	0 2 0 0	1.296.000
Q2 =	{*Product::Group; Time::Month*}	4 0 3 0	162.000
Q3 =	{*Customer::Retailer; Time::Quarter*}	0 1 2 0	1.458.000

Tab. 5 zeigt die Ergebnisse der „Top“-Fragmentierungen, die bzgl. der Einzelmetrik für den I/O-Aufwand (IOA_Q) und der kombinierten Metrik (*IOM*) ermittelt wurden. Bezüglich des Arbeitsaufwandes liefert die Fragmentierung *{Customer::Store}* die besten Ergebnisse. Sämtliche Treffer werden in einem Fragment geclustert und können somit mit höchster I/O-Effizienz gelesen werden. Außerdem ist kein Bitmap-Zugriff notwendig. Auf den weiteren Plätzen folgen jeweils Fragmentierungen, die neben dem Query-Attribut zusätzliche Dimensionen involvieren. Die Treffer liegen dadurch immer noch geclustert vor, da kein gröberer Hierarchie-Level der *Customer*-Dimension durch die Fragmentierungen involviert wird, der die Lokalisation von Treffern (und dadurch zusätzliche I/O) durch Bitmaps nötig machen würde. Die Treffer werden allerdings nun auf 2, 8, 16 usw. Fragmente verteilt, je nach Kardinalität der involvierten Dimensionen. Dadurch ergibt sich ggf. ein „Verschnitt“-Overhead, da durch Verteilung der Treffer auf mehrere Fragmente nun u.U. mehr I/Os durchgeführt werden müssen. Sind z.B. für die Verarbeitung von 1000 Seiten eines Fragmentes mit PF_{FT} = 8 Seiten nur 125 I/Os notwendig, so sind bei Verteilung auf z.B. 4 Fragmente mit jeweils 250 Seiten 32 I/O pro Fragment, also insgesamt 128 I/Os notwendig. Dies, und das bei sehr kleinen Fragmentgrößen geringere und damit ineffizientere Prefetching-Granulat begründen den höheren I/O-Aufwand verglichen mit *{Customer::Store}*.

Unter *IOM* wurden Aufwand und Antwortzeit derart kombiniert, daß die „Top 20%“-Fragmentierungen bzgl. IOA_Q gemäß der besten erreichten Antwortzeiten ausgewählt wurden. Die bzgl. Aufwand optimale Fragmentierung *{Customer::Store}* findet sich erst auf Platz 34. Dies begründet sich damit, daß durch die Reduktion auf 1 Fragment unsere Beispiel-Query nicht parallelisierbar ist, und somit eine schlechtere Antwortzeit erzielt. Die bzgl. *IOM* „beste“ Fragmentierung „3 2 1 0“ erzielt eine um den Faktor 66 bessere Antwortzeit, bei lediglich 20% mehr Aufwand. Es ist festzuhalten, daß mehrdimensionale Fragmentierungen vorgeschlagen werden, welche einerseits durch Berücksichtigung des Hierarchie-Levels des Query-Attributes Treffer-Clustering erlauben und Bitmap-Zugriff vermeiden, andererseits einen solchen Feinheitsgrad erreichen, daß ein ausreichendes Parallelisierungspotential gewährleistet ist.

Es sei erwähnt, daß ein herkömmlicher „Full Scan“ bei einem angenommenen Plattendurchsatz von 10 MB/s und maximaler Parallelität, auch ohne Bitmap-Zugriff, für

Tab. 5. Ergebnisse für Q_{store}

IO-Aufwand				kombinierte IO-Metrik				
Rang	Fragmentierung	F_{FT}	IOA_Q (in s)	Rang	Fragmentierung	F_{FT}	$IORT_Q$ (in s)	IOA_Q (in s)
1	0 2 0 0	1	8.96	1	3 2 1 0	240	0.135	10.8
2	0 2 1 0	2	9.0	2	1 2 1 1	240	0.135	10.8
3	1 2 0 0	8	9.0	3	4 2 0 0	480	0.146	13.9
4	0 2 2 0	8	9.0	4	2 2 3 0	576	0.175	16.7
5	1 2 1 0	16	9.36	5	0 2 2 1	360	0.180	10.8
...				...				
14	3 2 1 0	240	10.8	34	0 2 0 0	1	8.96	8.96

Tab. 6. Ergebnisse für IOM (Query-Mix)

IOM			
Platz	Fragmentierung	$IORT_{Mix}$ (in s)	IOA_{Mix} (in s)
1	3 2 2 0	1.37	133.6
2	3 2 3 0	1.72	168.9
3	2 2 3 0	1.82	178.1

die ca. 37 GB große Faktentabelle auf 100 Platten ca. 37s I/O-Zeit benötigen würde, also um einen Faktor 275 langsamer wäre.

6.2 Query-Mix

Tab. 6 zeigt das Ergebnis für einen Query-Mix, bestehend aus den Anfragen Q1=*{Customer::Store}*, Q2=*{Product::Group; Time::Month}* und Q3=*{Customer::Retailer; Time::Quarter}*. Es werden insgesamt 2.916.000 Fakten bzgl. der 3 Dimensionen *Product*, *Customer* und *Time* auf unterschiedlichen Hierarchieebenen referenziert. Die Anfragen wurden alle identisch gewichtet.

Alle „Top 3“-Fragmentierungen führen zu einem Clustering der Treffer und zu einer Verteilung auf > 100 (= Anzahl Platten) Fragmente für die Queries Q1 und Q3, wodurch sich gute Aufwands- und Antwortzeitergebnisse durch Parallelisierung erzielen lassen. Für Q2 wird durch Referenzierung des *Group*-Level auf der *Product*-Dimension und des *Month*-Level auf der *Time*-Dimension Bitmap-Zugriff eingeführt. Dieser Aufwand ist allerdings wesentlich geringer als der Negativeffekt welcher die Wahl der feingranulären Fragmentierung „4 2 3 0“ implizieren würde, die Bitmap-Zugriff für den Mix generell vermeidet. Diese Fragmentierung erzeugt Faktentabellenfragmente der Größe < 1 Seite, was zu inakzeptablen Ineffizienz der I/O-Zugriffe führt. Für die ermittelten besten Fragmentierungen kann dagegen ein 16- bzw. sogar 32-Seiten Prefetching eingesetzt werden. Alle aufgeführten Fragmentierungen führen Bitmap-Zugriff lediglich für Q2 ein, welche aufgrund ihrer geringen Trefferanzahl auch die unkritischste Anfrage im Mix darstellt.

7 Zusammenfassung und Ausblick

Wir haben einen Ansatz zur automatischen Bestimmung einer multi-dimensionalen, hierarchischen Fragmentierung für Data Warehouses präsentiert. Auf Basis von Kostenformeln haben wir eine Metrik entwickelt, welche die Güte einer Fragmentierung bzgl. der für Star Queries sehr häufig dominanten IO-Zugriffszeiten bestimmt. Unsere kombinierte Metrik berücksichtigt neben der Reduktion des I/O-Arbeitsaufwandes auch die I/O-Antwortzeiten, welche durch Einsatz von Parallelität erreicht werden können. Erste Ergebnisse zeigen, daß mit unserem Modell gute Fragmentierungskandidaten bestimmt werden.

Aktuell arbeiten wir an der detaillierten simulativen Validierung des Modells und an seiner Verfeinerung. Erste Tests zeigen anhand von Simulationsergebnissen eine sehr gute Approximation der I/O-Kosten durch unser Modell. Zur Plattenbalancierung von Fragmenten, deren Größe durch z.B. Skew stark variieren kann, haben wir eine „hitze"basierte Allokationsstrategie in unserem Simulationssystem implementiert, welche zukünftig auch im Kostenmodell reflektiert werden soll. In weiterenen Schritten sollen neben der Integration von CPU-Kosten auch Nichtpunktfragmentierungen/-anfragen berücksichtigt werden. Das beschriebene Modell wird derzeit in einem Werkzeug mit GUI-Oberfläche implementiert, welches basierend auf Sternschemainformationen und DB-Größen für einen gewichteten Query-Mix eine passende Datenallokation automatisch vorschlägt.

Danksagung. Der Autor bedankt sich bei Herrn Prof. Erhard Rahm für detaillierte Anmerkungen und Verbesserungsvorschläge, bei Herrn Holger Märtens für die Diskussion des Kostenmodells sowie bei den Gutachtern für ihre hilfreichen Kommentare.

Literatur

1. Agrawal, S., Chaudhuri, S., Narasayya, V.R.: Automated Selection of Materialized Views and Indexes in Microsoft SQL Server. Proc. 26th Intl. VLDB Conference: 496 – 505, Cairo, Egypt, 2000.
2. Brobst, S., Vecchione, B.: DB2 UDB: Starburst Grows Bright. Database Programming & Design, 1998.
3. Chaudhuri, S., Dayal, U.: An Overview of Data Warehousing and OLAP Technology. SIGMOD Record 26(1): 65-74, 1997.
4. Chaudhuri S., Narasayya, V.: An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server. Proc 23rd Intl. VLDB Conference: 146 – 155, Athens, Greece, 1997.
5. Chaudhuri, S., Weikum, G.: Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. Proc. 26th Intl. VLDB Conference: 1 – 10, Cairo, Egypt, 2000.
6. Chen, P.M., Lee, E.L., Gibson, G.A., Katz, R. H., Patterson, D. A.: RAID: High-Performance, Reliable Secondary Storage. ACM Computing Surveys 26 (2): 145 – 185, 1994.
7. Copeland, G., Alexander, W., Boughter, E., Keller, T.: Data Placement in Bubba. Proc. ACM SIGMOD Conference, Chicago, 1988.
8. DeWitt, D.J., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems. Comm. ACM 35 (6): 85-98, 1992.
9. Gardy, D., Némirovski, L.: Urn Models and Yao's Formula. Proc. 7th ICDT Conference, Jerusalem, 1999.
10. Ghandeharizadeh, S., DeWitt, D.J.: A Multiuser Performance Analysis of Alternative Declustering Strategies. Proc. 6th Intl. Conference on Data Engineering, Los Angeles, 1990.
11. Ghandeharizadeh, S., DeWitt, D.J., Qureshi, W.: A Performance Analysis of Alternative Multi-Attribute Declustering Strategies. Proc. ACM SIGMOD Conference: 29 - 38, 1992.
12. Informix Corporation: Informix Decision Support Indexing for the Enterprise Data Warehouse. White Paper, 1998.

13. Informix Corporation: INFORMIX-OnLine Dynamic Server: Administration Guide. Manual, 2000.
http://www.informix.com/answers/oldsite/answers/pubs/pdf/811xpsu/7624.pdf
14. Katz, R.H., Hong, W.: The Performance of Disk Arrays in Shared-Memory Database Machines. Distr. and Parallel Databases 1 (2): 167 – 198, 1993.
15. Lee, E.K., Katz, R.: An Analytic Performance Model of Disk Arrays, Proc. ACM SIGMETRICS Conference, 98 – 109, Santa Clara, 1993.
16. Marek, R.: Ein Kostenmodell der parallelen Anfragebearbeitung in Shared-Nothing-Datenbanksystemen, Proc. GI-Fachtagung 'Datenbanken in Büro, Technik und Wissenschaft' (BTW'95): 232 – 251, Informatik aktuell, Dresden, 1995.
17. Mehta, M., DeWitt, D.J.: Data Placement in Shared-Nothing Parallel Database Systems. VLDB Journal 6 (1), 1997.
18. Mesquite Software Inc: User's Guide CSIM18 Simulation Engine. Manual, 1996.
19. OLAP Benchmark, Release II. OLAP Council, 1998.
htpp://www.olapcouncil.org/research/bmarkly.htm
20. O'Neil, P., Graefe, G.: Multi-Table Joins Through Bitmapped Join Indices. ACM SIGMOD Record 24 (3), 1995.
21. Oracle Corporation: Oracle 8i Administrator's Guide. Manual, 2000.
http://www.irm.vt.edu/oracle_816_docs/server.816/a76956/index.htm
22. Oracle Corporation: Star Queries in Oracle8. White Paper, 1997.
23. Rahm, E.: Parallel Query Processing in Shared Disk Database Systems. Proc. 5th HPTS Workshop, Asilomar, 1993.
http://dbs.uni-leipzig.de/abstr/Ra93.HPTS.ps
24. Rahm, E.: Mehrrechner-Datenbanksysteme. Grundlagen der verteilten und parallelen Datenbankverarbeitung. Addison-Wesley-Verlag, 443 S., 1994.
http://dbs.uni-leipzig.de/buecher/mrdbs.html
25. Rahm, E., Märtens, H., Stöhr, T.: On Flexible Allocation of Index and Temporary Data in Parallel Database Systems. Proc. 8th HPTS Workshop, Asilomar, 1999.
http://dol.uni-leipzig.de/pub/1999-23
26. Rahm, E., Stöhr, T.: Analysis of Parallel Scan Processing in Parallel Shared Disk Database Systems. Proc. EURO-PAR Conference, Stockholm, LNCS 966, Springer, 1995.
http://dol.uni-leipzig.de/pub/1995-22
27. Red Brick Systems, Inc.: Star Schema Processing for Complex Queries. White Paper, 1998.
28. Scheuermann, G. Weikum, P. Zabback: Data Partitioning and Load Balancing in Parallel Disk Systems, VLDB Journal 7 (1): 48 – 66, 1998.
29. Stöhr, T., Märtens, H., Rahm, E.: Multi-Dimensional Database Allocation for Parallel Data Warehouses. Proc. 26th Intl. VLDB Conference: 273 – 284, Cairo, Egypt, Sep. 2000.
30. Stöhr, T.: Indexallokation in Parallelen Datenbanksystemen. Proc. 11. GI-Workshop "Grundlagen von Datenbanken", Luisenthal/Thüringen, 1999.
31. Sun, J., Grosky, W.I.: Dynamic Maintenance of Multidimensional Range Data Partitioning for Parallel Data Processing. Proc. First ACM Intl. Workshop on Data Warehousing and OLAP (DOLAP), Washington D.C.: 72 – 79, 1998.
32. Wu, M.-C., Buchmann, A.P.: Encoded Bitmap Indexing for Data Warehouses. Proc. 14th Proc. Intl. Conference on Data Engineering (ICDE), Orlando, 1998.
33. Yao, S.B.: Approximating Block Accesses in Data Base Organizations. Comm. ACM 20 (4), 1977.