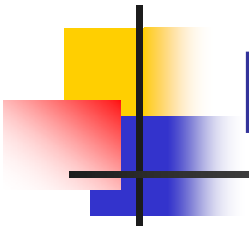
Temporal and Spatio-Temporal Aggregations over Data Streams using Multiple Time Granularities

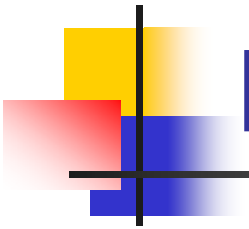
Von Donghui Zhang, Dimitrios Gunopulos, Vassilis J. Tsotras und Bernhard Seeger

Vorgetragen von Christian Humm



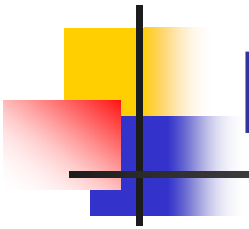
Kapitel 0: Inhalt

- 1: Einführung
- 2: Vorstellung der Aggregationsmodelle
- 3: Temporale Aggregation mit fixem Speicherplatz
- 4: Temporale Aggregation mit fixem Zeitfenster
- 5: Performanz
- 6: Abschluss & Diskussion
- 7: Quellen



Kapitel 1: Einführung

- Volumen an gesammelten Daten wächst rapide an
  - Data Warehouses (z.B. Amazon)
  - Zeitgebundene Datenbanken
    (z.B. Telekommunikationsdaten)
  - Raum / Zeit gebundene Datenbanken (z.B. Wetterdaten)
- Speichervolumen aber begrenzt
- Verfahren zur Komprimierung von Daten nötig!



Einführung (2)

- Aber auch wenn genug Speicher vorhanden ist, braucht man eine sinnvolle Datenstruktur, um gesuchte Einträge effizient finden zu können…
- Berechnung der Aggregationen soll in einem einzigen Durchgang des Datenstroms machbar sein
- Logarithmisches Verhalten bei Such- und Einfügeoperationen erwünscht

-> Geeignete Baumstruktur benutzen!

Einführung (3)

- Meistens braucht man nur neue Daten hochaufgelöst, für ältere Daten reicht eine grobe Auflösung

->Temporale Aggregation der Daten!

- Große Telekommunikationsgesellschaften sammeln beispielsweise ca. 75 GB Daten pro Tag (= 27 TB pro Jahr)!
- Betrachtung von kumulativen Aggregationen (in einem bestimmten Intervall) mit 2SB-Bäumen* anstatt von instantanen Aggregationen (zu einem festen Zeitpunkt), da dies allgemeiner ist
- Wir betrachten als Aggregation SUM, Lösungen mit COUNT oder AVG sind aber analog...

*2SB-Bäume werden in Kapitel 3 erläutert

Anwendungsbeispiele

- Zeitbereichsweise Aggregation:
  - „Finde heraus, wieviele Telefonanrufe im Jahr 2006 in Darmstadt getätigt wurden“
- Erhalte dazu einen Satz Objekte (Telefonanrufe), von denen jeder einen bestimmten Schlüssel (Telefonnummer), ein Zeitintervall (wann der Anruf stattgefunden hat) und einen Wert (hier 1) hat
- Raum-Zeitliche Aggregation
  - „Finde den gemessenen Gesamtniederschlag allen Regens in Darmstadt im Jahre 2006 heraus“
- Erhalte dazu einen Satz Objekte (Niederschlagmessungen) mit je einem (d-1)-dimensionalen Rechteck (Bereich des Messgeräts), einem Zeitintervall und einen Wert (Menge des Niederschlags)

Kapitel 2: Aggregationsmodelle

- Unsere Zeitrechnung hat von Natur aus eine hierarchische Struktur
- Eine k-stufige Zeithierarchie wird notiert als:
  $gran_1 \rightarrow \ldots \rightarrow gran_k$
- $gran_1$ ist gröbste, $gran_k$ feinste Körnung (Granularität)
- Bsp. für 3-stufige Hierarchie:

  Jahr → Monat → Tag

Das HTA-Modell

Das Hierarchisch Temporale Aggregationsmodell (HTA)

orig: Beginn des Datenstroms

t_{div}^{i} sollte dynamisch anpassbar sein, da im Laufe der Zeit die Einteilungen evtl. geändert werden müssen

Das HTA-Modell (2)

Wir unterteilen das HTA-Modell in zwei weitere Untermodelle:

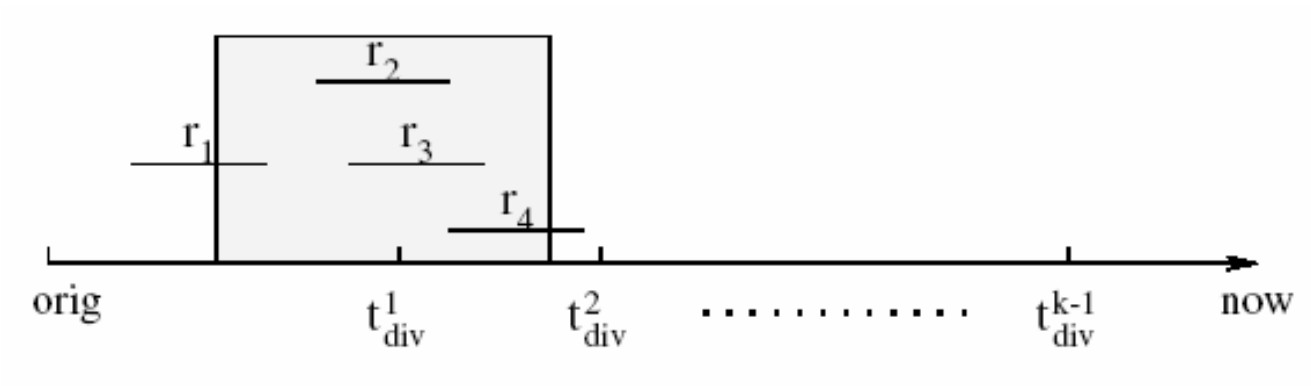
- Das Modell mit fixem Speicherplatz (Fixed Storage Model)
  - Annahme: Speicherplatz ist begrenzt
  - Wenn verbrauchter Platz eine festgelegte Grenze überschreitet, werden ältere Daten grobkörniger aggregiert
- Das Modell mit fixem Zeitfenster (Fixed Time Window Model)
  - Annahme: Länge der Segmente festgelegt
  - Für jedes Segment wird eine festgelegte Granularität verwendet
  - Fenster werden in bestimmten Zeitabständen verschoben

Das HTA-Modell (3)

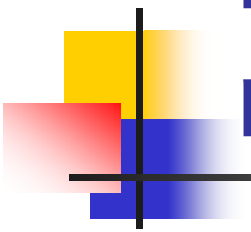
- Beide Modelle entsprechen unterschiedlichen Anforderungen, keines ist offensichtlich „besser“
- Um Effizienz zu waren: Wir erhöhen t_{div}^{i-1} nicht bei jedem Einfügen eines Elements, sondern nur:
  - Wenn der benötigte Platz die bestimmte Größe um einen gewissen Bereich überschritten hat (FS)
  - Wenn ein Zeitfenster seine Größe um ein gewisses Maß überschritten hat (FTW)

Das HTA-Modell (4)

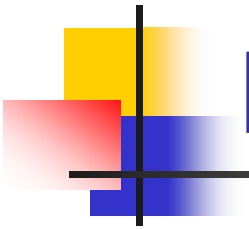
- Ein weiteres Problem, das zu beachten ist: Ein Datensatz kann sich über mehrere Zeitbereiche erstrecken, soll aber natürlich nicht doppelt gezählt werden!



Kapitel 3: Temporale Aggregation mit fixem Speicherplatz



- Wir benutzen eine Baumstruktur ähnlich des B-Baums: der SB-Baum
- Der SB-Baum hat Eigenschaften des Segment-Baumes und des B-Baumes
  - Segment-Baum Eigenschaft: Index kann effizient geupdated werden
  - B-Baum Eigenschaft: Die Struktur ist balanziert
- Jeder innere Knoten enthält zwischen b/2 und b Einträge, jeder repräsentiert ein bestimmtes Zeitintervall



Der SB-Baum

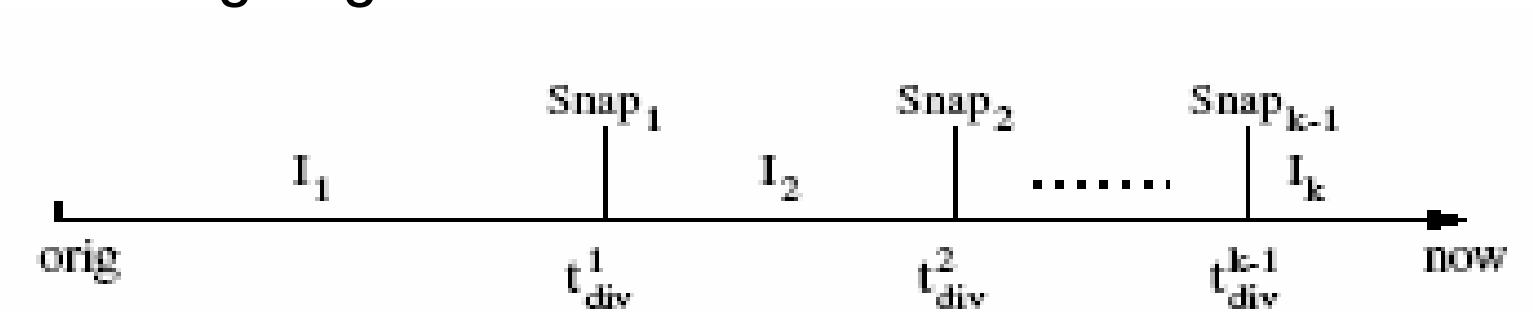
- Informationen sowohl in Knoten als auch in den Blättern
- Durch rekursives Durchsuchen des SB-Baums (angefangen von der Wurzel) und Aufaddieren der aggregierten Werte der Knoten wird eine instantane temporale Aggregierung berechnet
- Da wir in Zeitpunkten (und nicht in Intervallen) aggregieren, umgehen wir damit automatisch das Problem, Daten die sich über mehrere Zeitbereiche erstrecken doppelt zu zählen!
- schnelle Aggregierungsberechnung und Einfügen neuer Werte (beides logarithmisch)

Der 2SB-Baum

- Man benutzt 2 SB-Bäume:
  - Einer enthält die Aggregierung der Daten, deren Anfangspunkt vor einem bestimmten Zeitpunkt liegen (Intervall von t bis + ∞)
  - Der andere enthält die Aggregierung der Daten, deren Ende vor einem bestimmten Zeitpunkt liegen (Intervall von - ∞ bis t)
- Berechnung einer temporalen Aggregierung bzgl. Intervall *i* (von *i.start* bis *i.end*):
  - Summe der Werte aller Daten, deren Startzeit vor *i.end* liegt berechnen und davon die Summe der Werte aller Daten, deren Ende vor *i.start* liegt subtrahieren

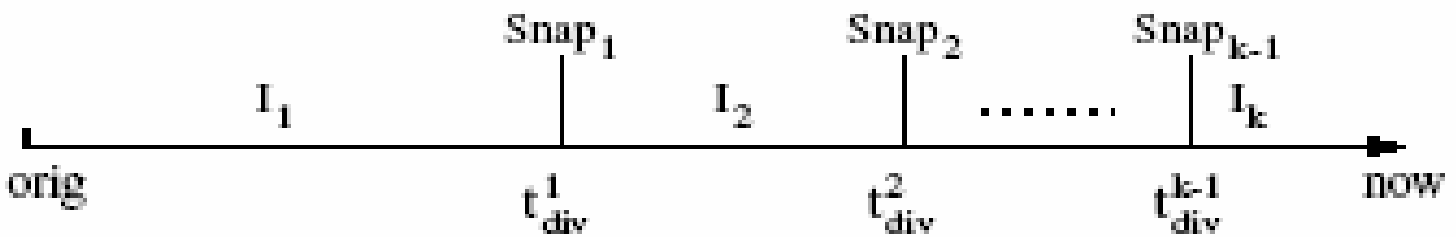
Der SB-Baum mit fixem Speicherplatz (SB^{FS})

- Die Zeitachse wird in k Segmente unterteilt
- Für jedes Segment wird ein SB-Baum I_i erhalten
- Der feste Speicherplatz S kann erzwungen werden, indem die Bäume I_i nur eine bestimmte Menge an Speicherplatz belegen dürfen (kann z.B. vom Warehouse Manager konfiguriert werden)
- Wert $Snap_i$ ist der Gesamtwert aller Punkte, die vor t^i_{div} eingefügt wurden



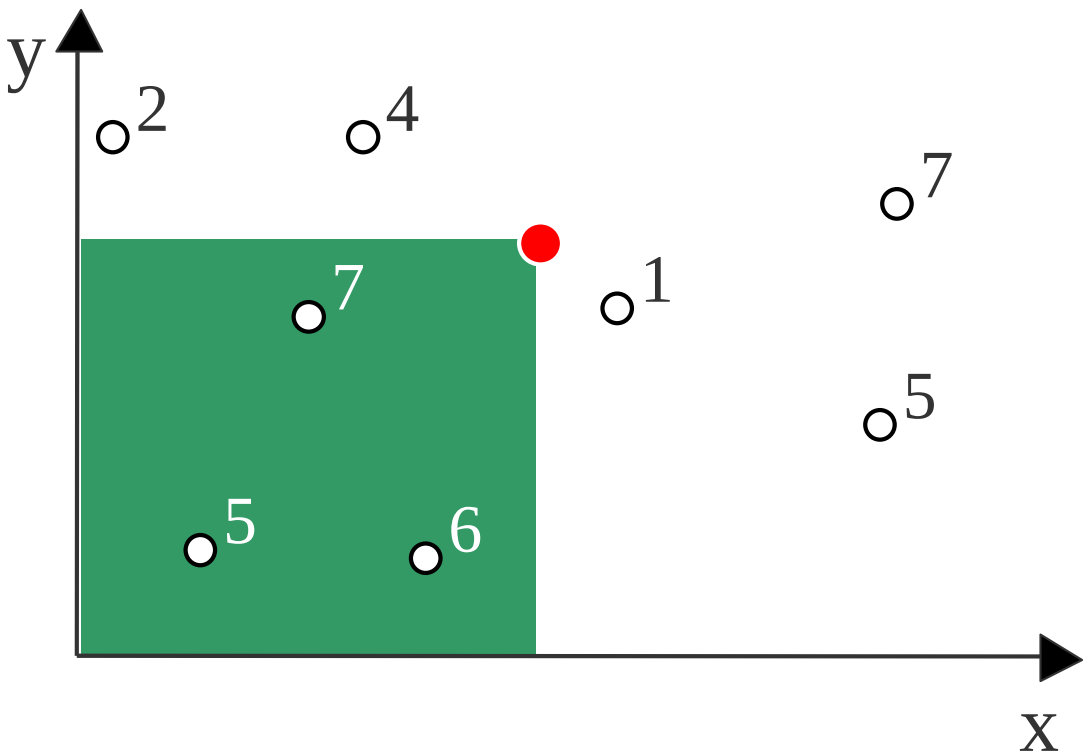
Suche im SB^{FS}-Baum

- Für die Berechnung der Summe zu einem bestimmten Zeitpunkt t_i gehen wir wie folgt vor:
  - Schaue, in welchem Bereich t_i liegt
  - Durchsuche den Baum I_i nach dem gesuchten Eintrag und bilde dort die Dominanzsumme
  - Addiere diese Dominanzsumme zu $Snap_{i-1}$



Dominanz-Summe

Summe der Werte der gewichteten Punkte, die vom Punkt P (roter Punkt im Schaubild) dominiert werden, d.h. links unter ihm liegen

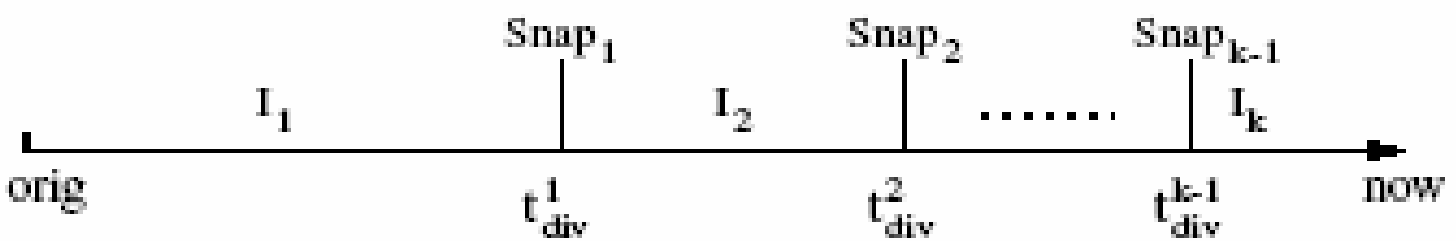


(Hier für Dimension = 2, für Dimension n analog)

Dominanz-Summe = 18

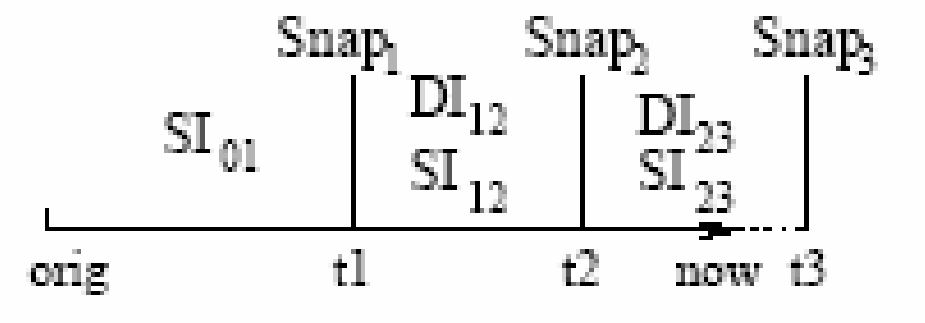
Einfügen im SB^{FS}-Baum

- Finde den richtigen Bereich im Baum und füge den Wert dort ein
- Addiere den Wert auch zu $Snap_i$ … $Snap_{k-1}$
- Wenn Baum zu groß wird, verschiebe das Zeitfenster t_{div}^{i-1} nach rechts



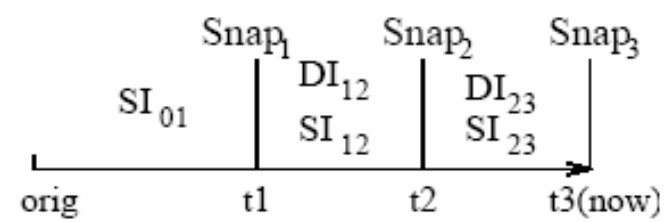
Kapitel 4: Temporale Aggregation mit fixem Zeitfenster (SB^{FTW})

- Der Einfachheit halber benutzen wir nur zwei Stufen in der Zeit-Hierarchie: Tag -> Minute (Erweiterung auf 3 oder mehr Stufen ist aber nicht schwierig)
- SI: Sparse Index (Aggregierung tagesweise)
- DI: Dense Index (Aggregierung minutenweise)
- $Snap_i$ enthält die Werte bis zum Zeitpunkt t_i

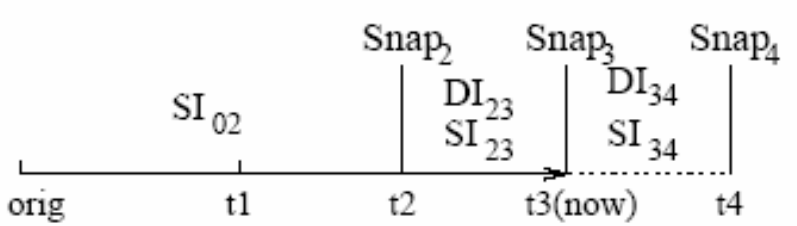


Operationen auf dem SB^{FTW}

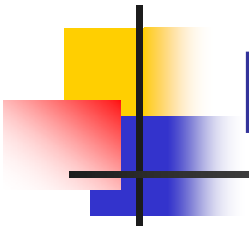
- Suche geht wie in SB^{FS}, aber darauf achten, den richtigen Baum zu durchsuchen (wenn Angabe nur auf Tag genau nicht im DI-Baum suchen...)
- Einfügen geht auch wie in SB^{FS}, aber das Verschieben des Zeitfensters wird fest vorgegeben und ist nicht vom Baum abhängig:



Vor dem Verschieben



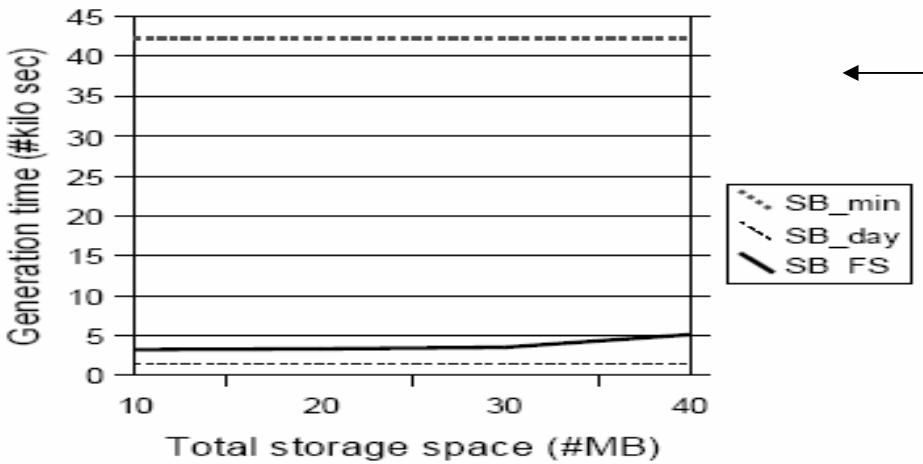
Nach dem Verschieben



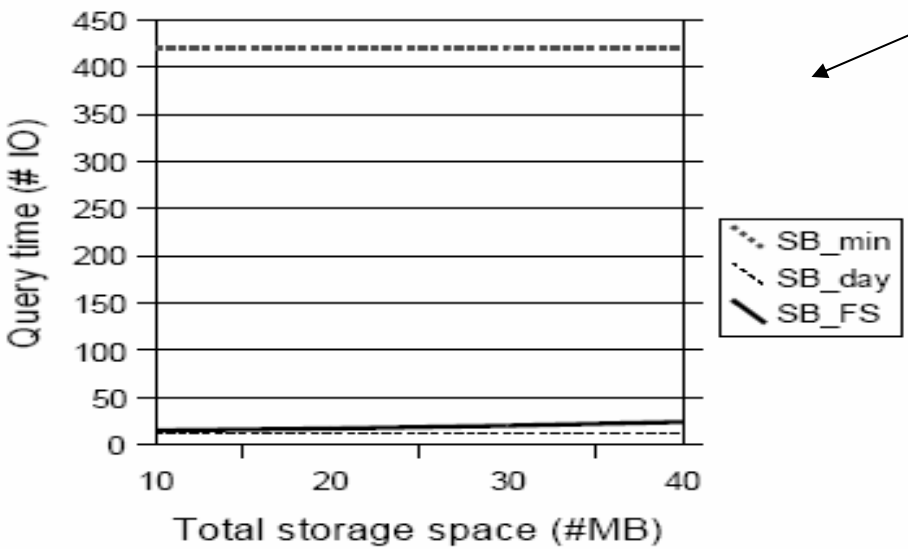
Kapitel 5: Performanz

- Im Folgendenden benutzte Voraussetzungen:
  - Algorithmen mit C++ GNU Compiler implementiert
  - Sun Enterprise 250 Server mit zwei 300 MHZ UltraSPARC-II Prozessoren mit Solaris 2.8
  - Messung der CPU-Kosten mit *getrusage* (Zeit im *user* und *system* Modus addieren)
  - I/O Kosten sind Anzahl I/O‘s * durchschnittliche disk page read access time (10 ms)
  - 8 KB page size
  - LRU Buffering mit Buffersize = 100 pages

Performanz: SB^{FS}



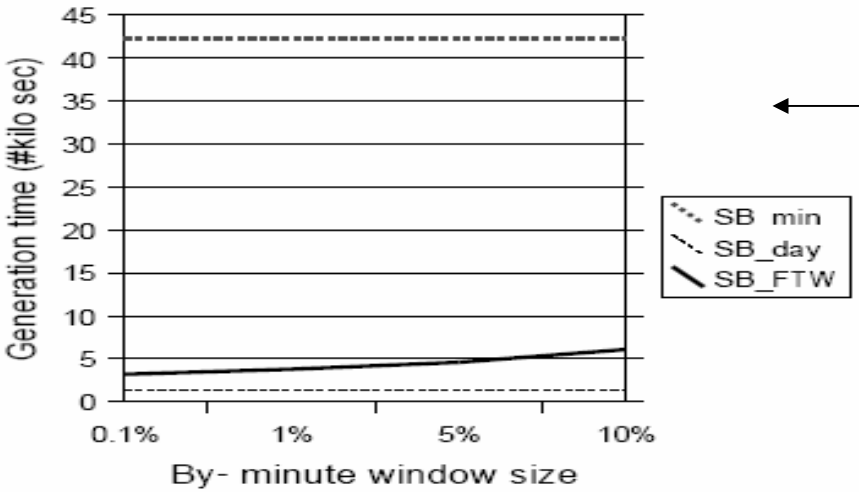
Benötigte Zeit zur Generierung.
Zum Vergleich die benötigten Zeiten, wenn man nur eine Granularität benutzt (Minute oder Tag)



Benötigte Zeit zur Suche.
Zum Vergleich die benötigten Zeiten, wenn man nur eine Granularität benutzt (Minute oder Tag)

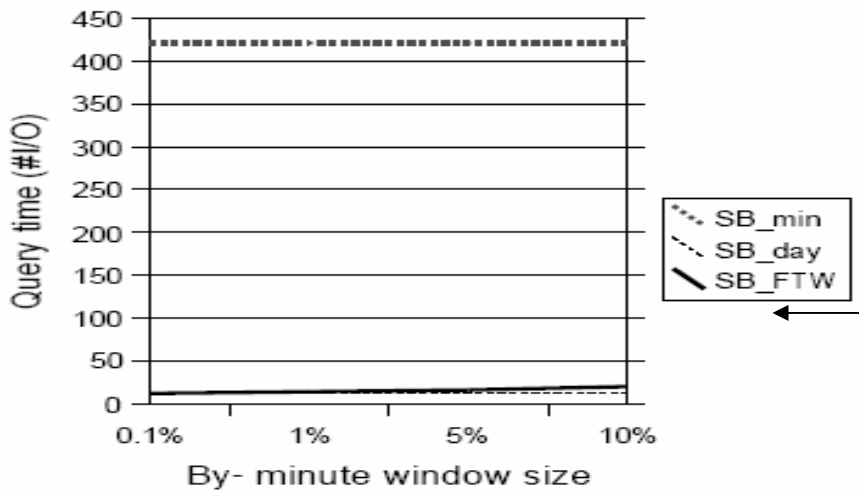
Benötigte Zeit wächst mit wachsender Speicherplatznutzung wegen steigender Indexgröße, bleibt aber nahe SB^{day}

Performanz: SB^{FTW}



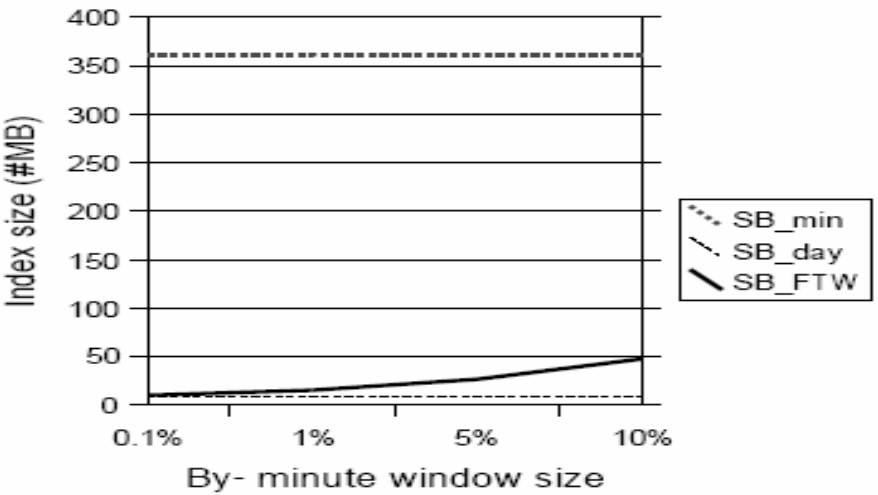
Benötigte Zeit zur Generierung

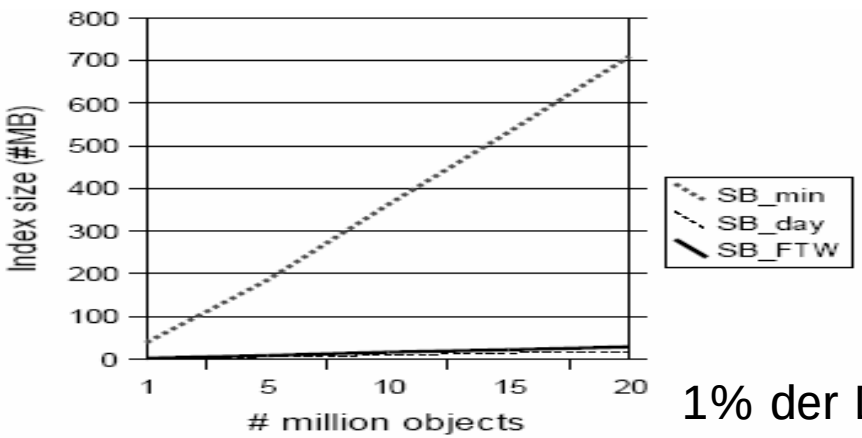
Je größer der Anteil, der in Minuten aufgelöst wird, desto seltener muss t_{div} erhöht werden. Allerdings wächst mit steigender Fenstergröße die Indizierung, wodurch der Zeitaufwand insgesamt steigt



Benötigte Zeit zur Suche

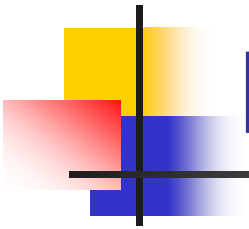
Performanz: Indexgröße SB^{FTW}





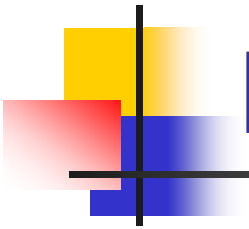
Bei 1% Minuten Fenster braucht SB^{FTW} nur 1/23 von Sb^{min}!

1% der Daten fest in Minuten-Granularität



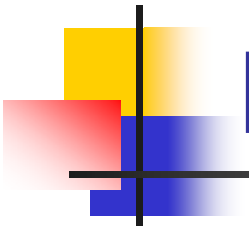
Kapitel 6: Abschluß

- Viele Anwendungen produzieren sehr große Mengen an zeitgebundenen Daten, die auf begrenzt vorhandenem Speicherplatz abgelegt werden müssen
- Diese Daten sollen in einem einzigen Durchgang durch den Datenstrom aggregiert werden können
- Lösung mit unterschiedlicher zeitlicher Granulierung der Daten: alte Daten werden gröber, neue Daten feiner gespeichert
- Dazu zwei Ansätze:
  - Fester Speicherplatz (Verfügbarer Speicherplatz hat eine feste Größe, z.B. 1 TB. Daten werden entsprechend in Tag / Stunden / Minuten eingeteilt)
  - Festes Zeitfenster (Wir betrachten die Daten ab dem 1.1.1990, wobei der letzte Tag minutenweise aufgelöst wird, der letzte Monat stundenweise und das letzte Jahr tageweise)
- Dadurch guter Kompromiß zwischen Genauigkeit und Speicherausnutzung / Effizienz!



Diskussion

- Fragen?



Kapitel 7: Quellen

- Temporal and Spatio-Temporal Aggregations over Data Streams using Multiple Time Granularities (Zhang, Gunopulos, Tsotras, Seeger 2002)
- Advanced Database Aggregation Query Processing (Zhang, 2002)