Operator-Kostenmodelle für Fortschrittsschätzung und Optimierung in Datenbanksystemen

Jan Kristof Nidzwetzki

23. Oktober 2012

Übersicht

Ziele der Bachelorarbeit

1. Entwicklung von neuen Kostenmodellen
2. Refactoring bestehender Operatoren
3. Auslagerung von Konstanten
4. Untersuchung des Zusammenhangs zwischen Arbeitsspeicher und Laufzeit
5. Entwicklung von Hilfsmitteln

Kostenmodelle

Kostenmodelle . . .

- . . . modellieren den Ressourcenbedarf eines Operators (*IO, Speicher, Rechenzeit*).
- . . . helfen dem Optimierer die Laufzeit von Operatoren abzuschätzen.
- . . . bilden die Grundlage für die Fortschrittsschätzung.
- . . . erlauben eine Abschätzung, wie viel Arbeitsspeicher ein Operator benötigt.

Kostenmodell - Laufzeit Mergejoin

$$T_{mergejoin} = (C_1 + C_2) \cdot uMergejoin + C_{mergejoin} \cdot (SN_1 + SN_2) \cdot yMergeJoin$$

Kostenmodell - Laufzeit Mergejoin

- Kosten für das Durchlaufen der Tupelströme

$$T_{mergejoin} = (C_1 + C_2) \cdot uMergejoin + C_{mergejoin} \cdot (SN_1 + SN_2) \cdot yMergeJoin$$

Kostenmodell - Laufzeit Mergejoin

- Kosten für das Durchlaufen der Tupelströme

$$T_{mergejoin} = (C_1 + C_2) \cdot uMergejoin + C_{mergejoin} \cdot (SN_1 + SN_2) \cdot yMergeJoin$$

Kostenmodell - Laufzeit Mergejoin

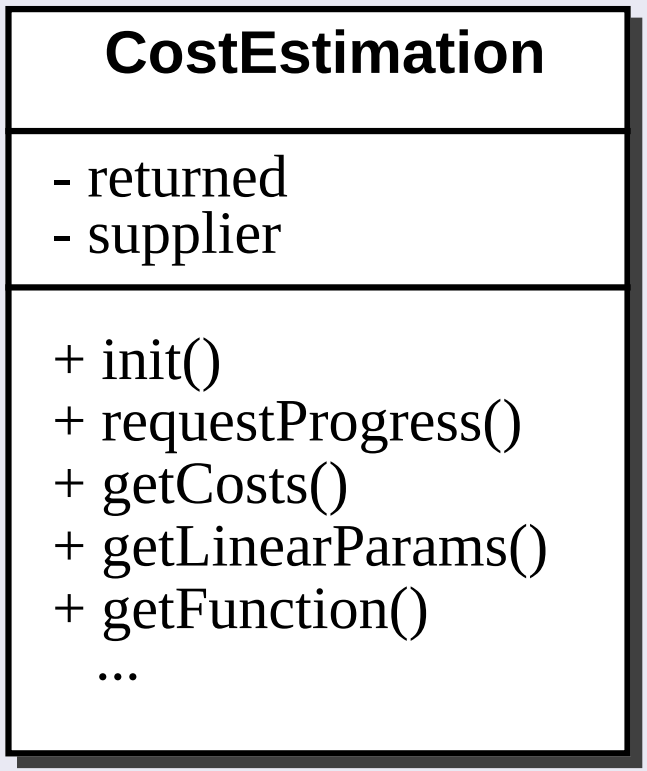
- Kosten für das Durchlaufen der Tupelströme

$$T_{mergejoin} = (C_1 + C_2) \cdot uMergejoin + C_{mergejoin} \cdot (SN_1 + SN_2) \cdot yMergeJoin$$

- Kosten für das Erzeugen der Ergebnistupel

Die Klasse *CostEstimation*

Klassendiagram

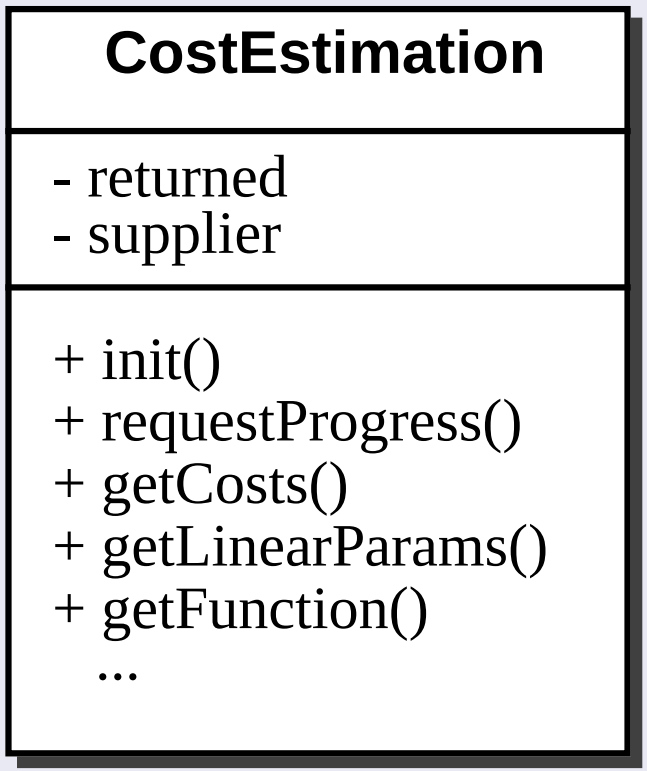


Vorteile

- Trennung Kostenmodell / Operator
- Verbesserte Datenkapselung
- Anbindung an den Optimierer
- Schätzung von benötigtem Arbeitsspeicher

Die Klasse *CostEstimation*

Klassendiagram



Vorteile

- Trennung Kostenmodell / Operator
- Verbesserte Datenkapselung
- Anbindung an den Optimierer
- Schätzung von benötigtem Arbeitsspeicher

Entwickeln von neuen Kostenmodellen

Analytisches Vorgehen

Analyse des Codes des Operators.

Experimentelles Vorgehen

Durchführen von Experimenten auf synthetischen Testdaten um Verhalten der Eingabegrößen zu ermitteln.

Entwickeln von neuen Kostenmodellen

Erzeugen von synthetischen Testdaten

Mit definierter Anzahl von Tupeln und Attributen.

Durchgeführte Experimente

- Verändern der Größe der Eingabetupel
- Verändern der Anzahl der Eingabetupel
- Verändern der Selektivität

Neue und angepasste Kostenmodelle

- Refactoring der *RelationAlgebra* (*feed*, *project*, *filter*, *feedproject*, *etc.*)
- Entwicklung von neuen Kostenmodellen (*itHashJoin*, *itSpatialJoin*)
- Überarbeiten von Kostenmodellen (*symmjoin*, *loopjoin*, *sortmergejoin*, *mergejoin*, *gracehashjoin*, *hybridhashjoin*)

Secondo Query Watcher

Durchsucht alle 100 *ms* das `proglogt.csv` von SECONDO und wertet die Einträge aus.

Funktionen

- Betrachten der Fortschrittsschätzung
- Bewerten der Fortschrittsschätzung
- Export der Daten als LaTeX Tabelle
- Export der Bilder als `.svg` Grafik

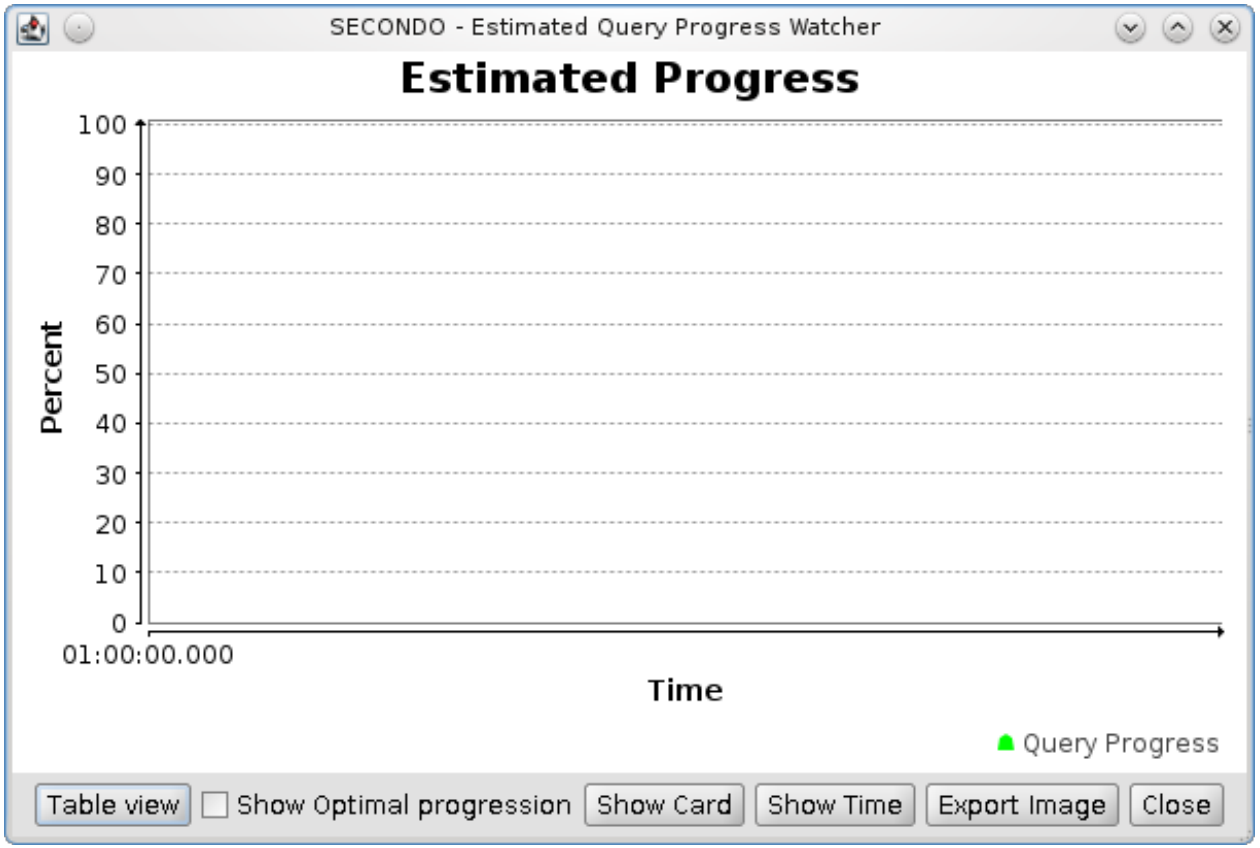
Secondo Query Watcher

Durchsucht alle 100 *ms* das `proglogt.csv` von SECONDO und wertet die Einträge aus.

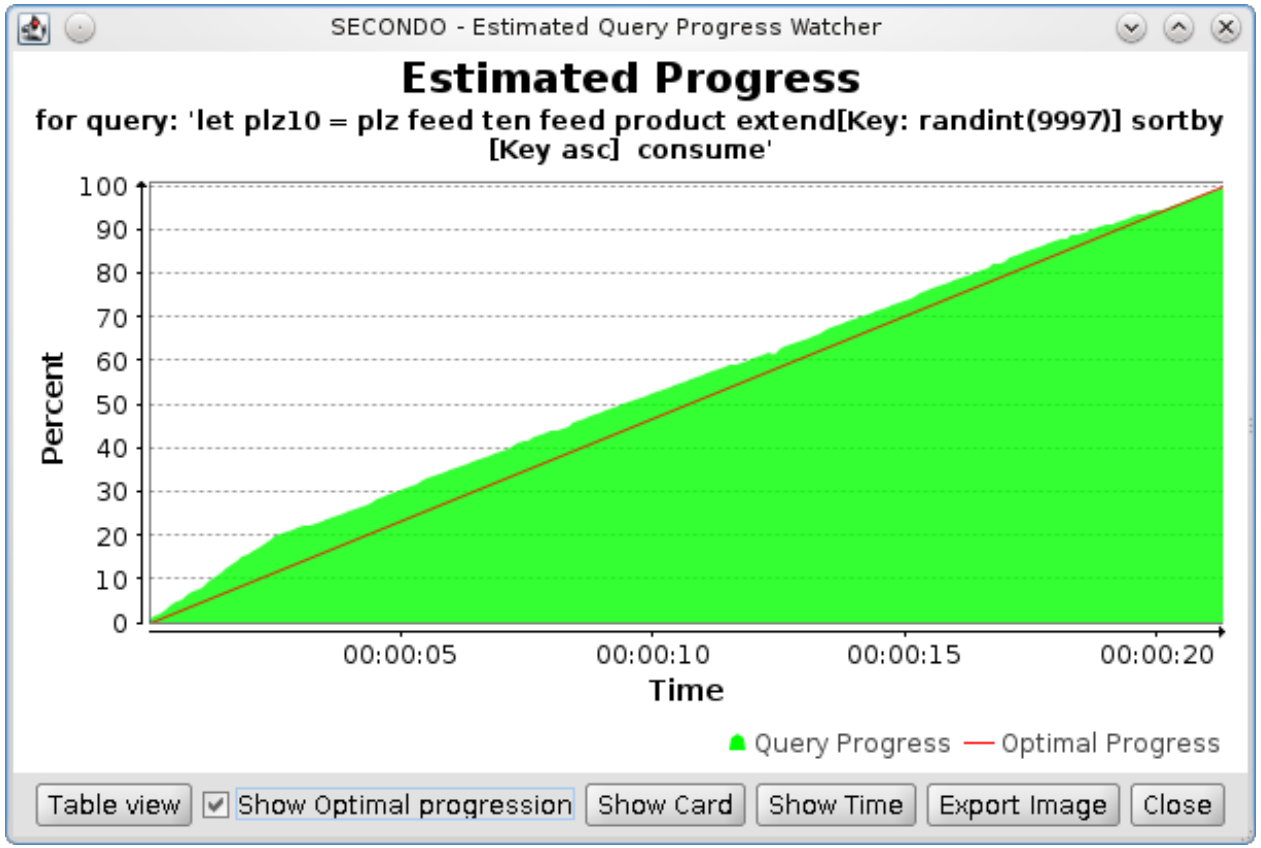
Funktionen

- Betrachten der Fortschrittsschätzung
- Bewerten der Fortschrittsschätzung
- Export der Daten als LaTeX Tabelle
- Export der Bilder als `.svg` Grafik

Secondo Query Watcher - Beispiel 1



Secondo Query Watcher - Beispiel 2



Secondo Query Watcher - Beispiel 3

SECONDO - Detailed progress view

Time (ms)	Progress	Opt Progress	Progress diff
0	0.00	0.00	0.00
110	0.82	0.50	0.32
220	1.78	0.99	0.79
330	2.73	1.49	1.24
440	3.12	1.98	1.14
550	4.08	2.48	1.60
660	4.91	2.97	1.94
770	5.49	3.47	2.02
880	6.30	3.97	2.33
990	7.05	4.46	2.59
1100	7.75	4.96	2.79
1210	8.51	5.45	3.06
1320	9.19	5.95	3.24
1430	9.76	6.44	3.32
1540	10.52	6.94	3.58
1650	11.29	7.44	3.85
1760	12.16	7.93	4.23
1870	12.95	8.43	4.52
1980	13.86	8.92	4.94
2090	14.53	9.42	5.11
2200	15.19	9.91	5.28
2310	15.75	10.41	5.34
2420	16.46	10.91	5.55
2530	17.09	11.40	5.69
2640	17.84	11.90	5.94

Error Max: 7 Error Int: 90489

Export as .csv | Export as .tex | Close

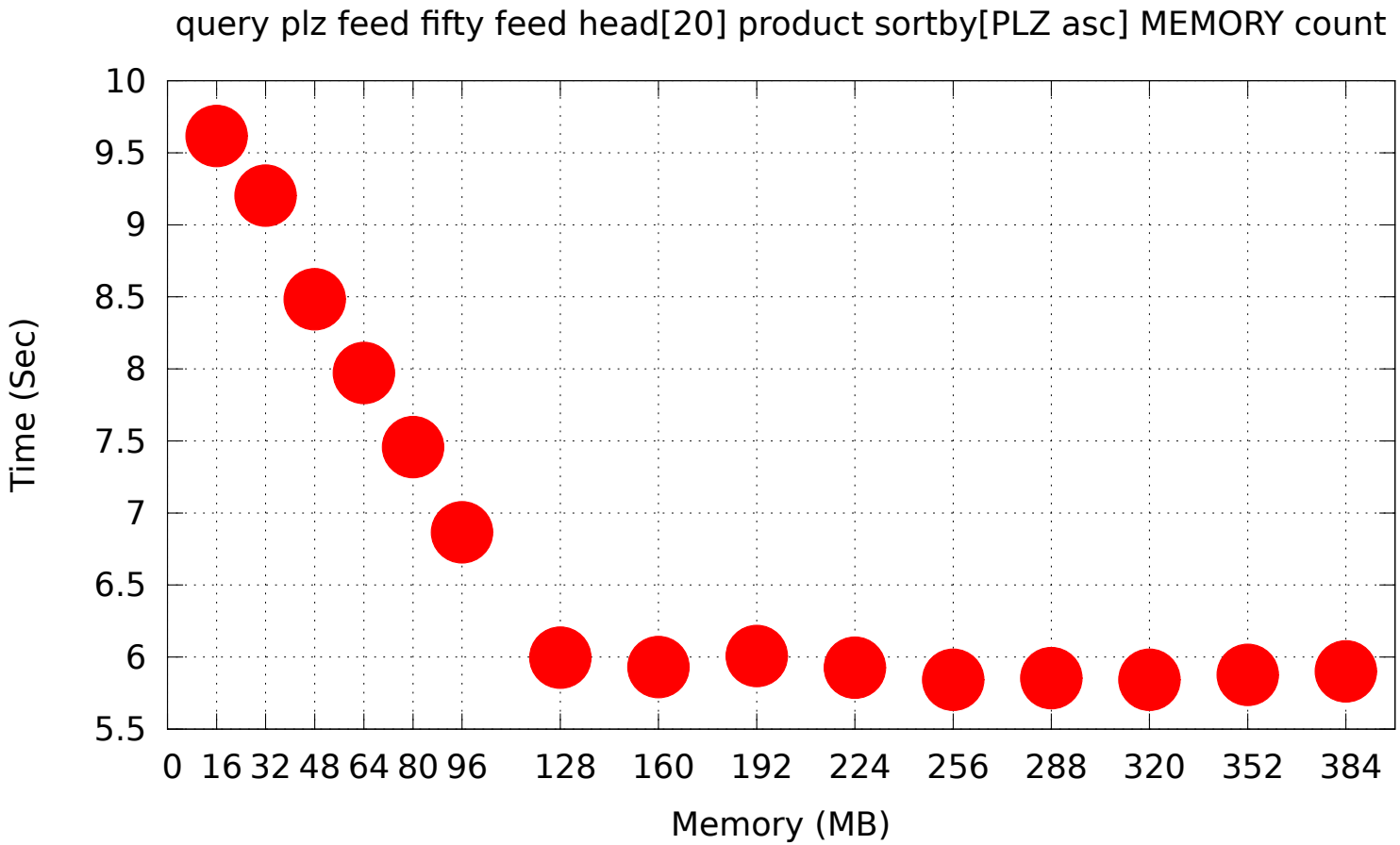
operator-memory.sh

- Führt eine Anfrage mit unterschiedlichen Arbeitsspeicherzuteilungen aus
- Stellt Ergebnisse als *CSV* zur Verfügung
- Aufbereitung der Daten mittels *GNUPlot*

Beispiel:

```
operator-memory.sh "query plz feed fifty
   feed head[20] product sortby[PLZ asc]
   _MEMORY_ count"
```

operator-memory.sh - Beispiel



Zuteilung von Arbeitsspeicher

Der *Optimierer* teilt den Operatoren eine bestimmte Menge an Arbeitsspeicher zu. In der Query kann mittels *{memory . . . }* Arbeitsspeicher an die Operatoren verteilt werden.

Beispiel:

```
query C2000000T50 feed {r1} C2000000T50
   feed {r2} itHashJoin[N_r1, N_r2]
   {memory 64} count
```

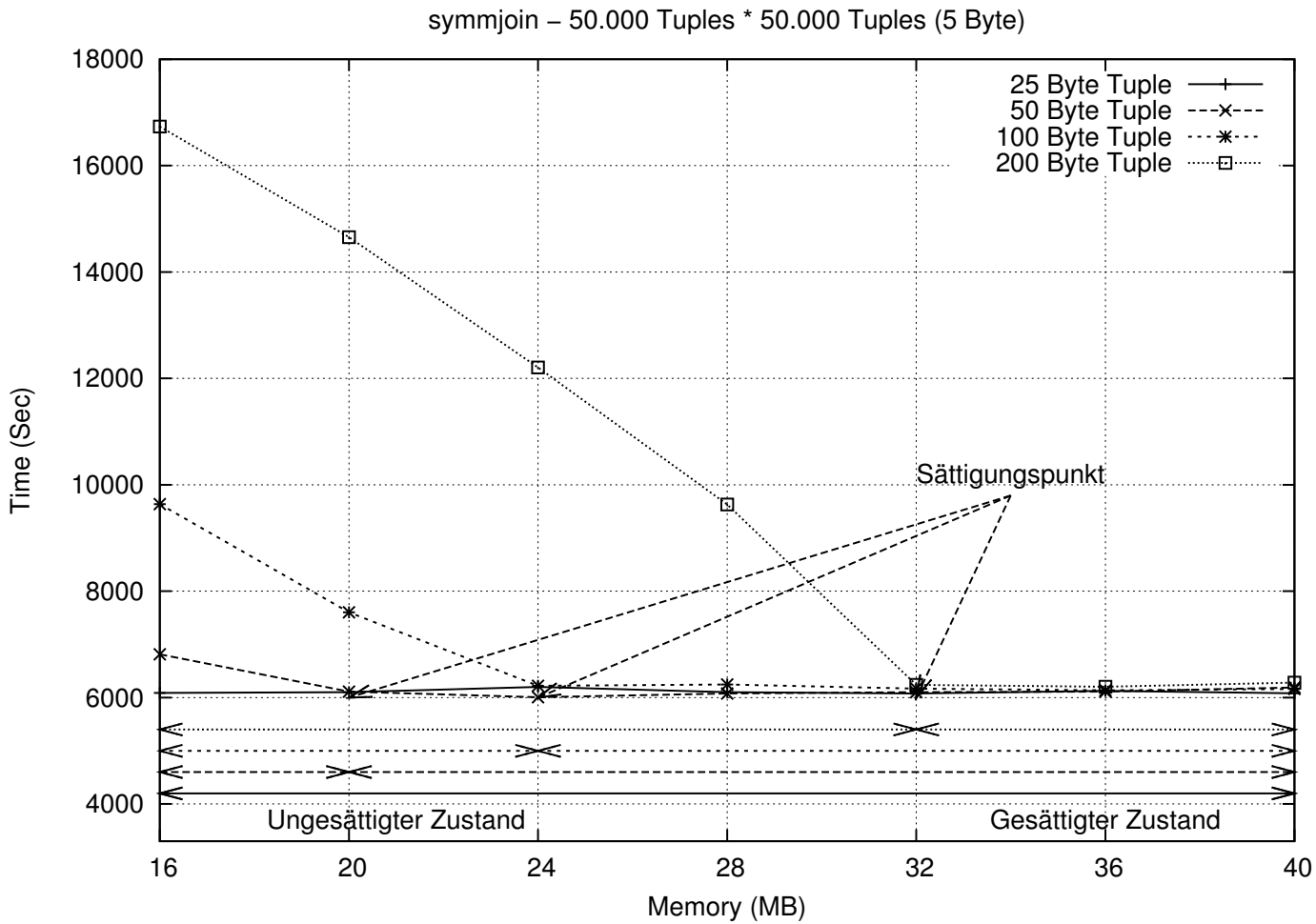
Arbeitsspeicher und Laufzeit

- Mit steigendem Arbeitsspeicher sinkt die Laufzeit
- Es gibt einen Punkt (*Sättigungspunkt*), ab dem die Laufzeit nicht weiter zu senken ist

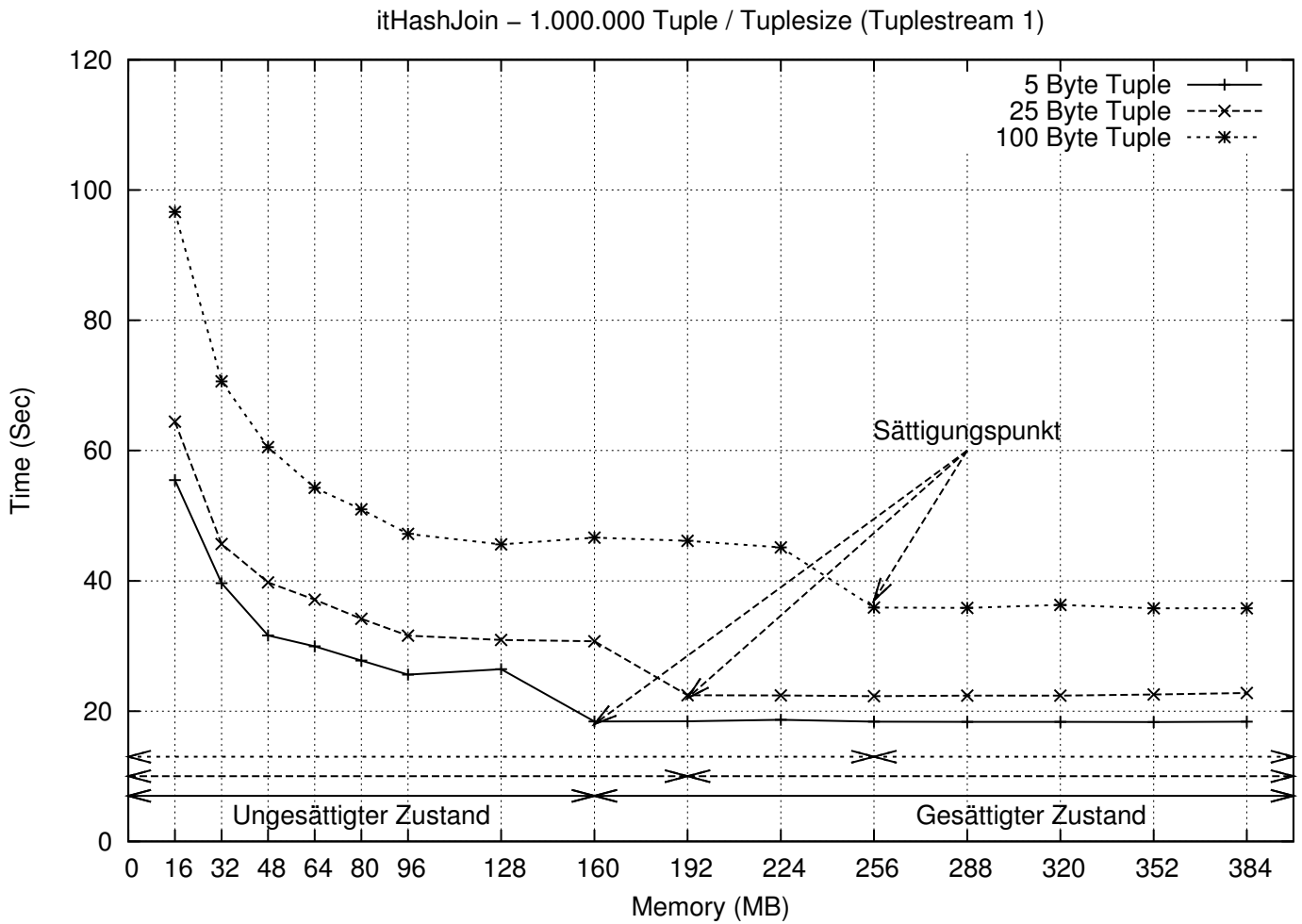
Es gibt Operatoren mit abweichendem Verhalten!

Gracehashjoin und *Hybirdhashjoin* können bei der Verwendung von zu wenigen Buckets langsamer werden.

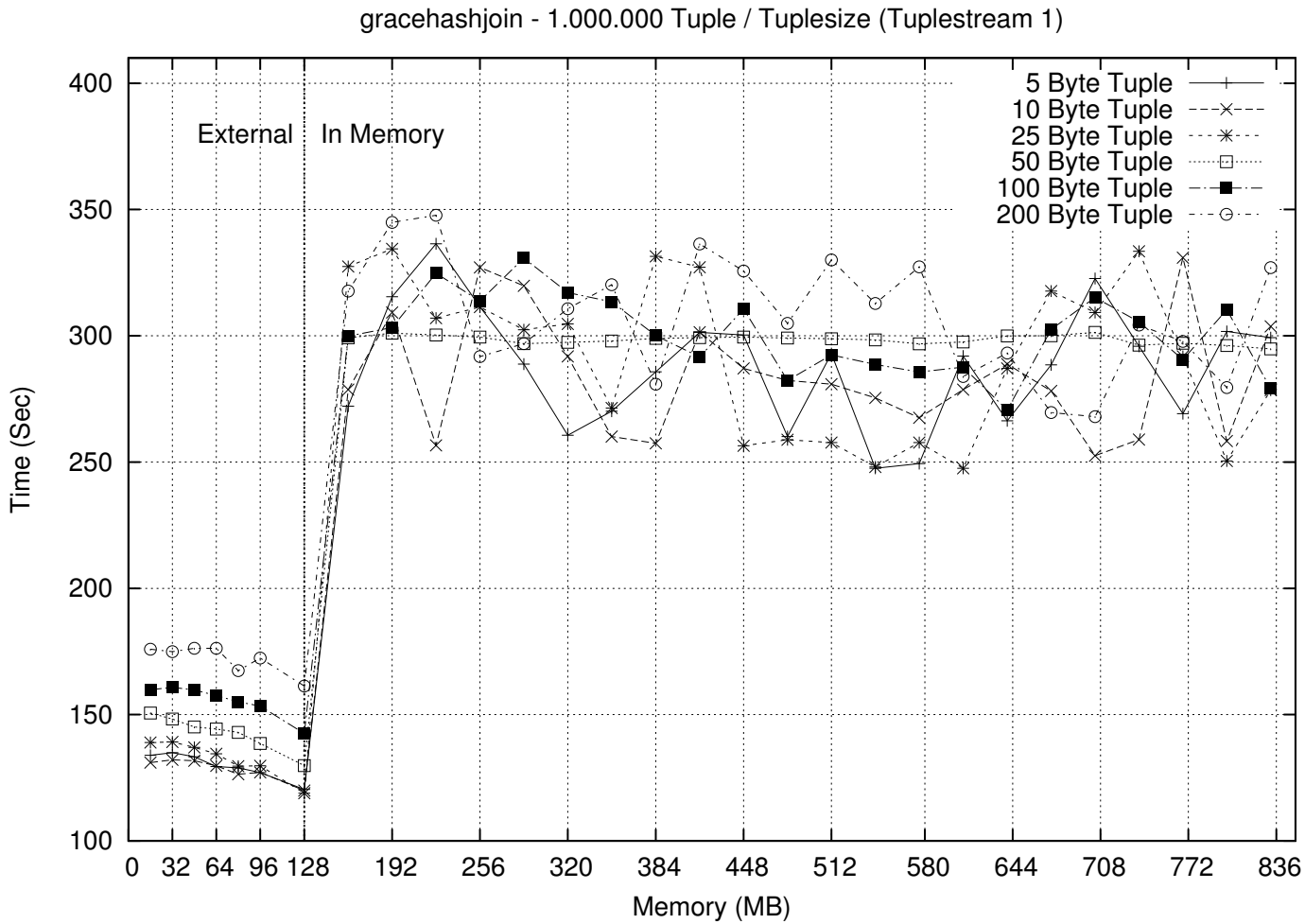
Laufzeitverhalten von Symmjoin



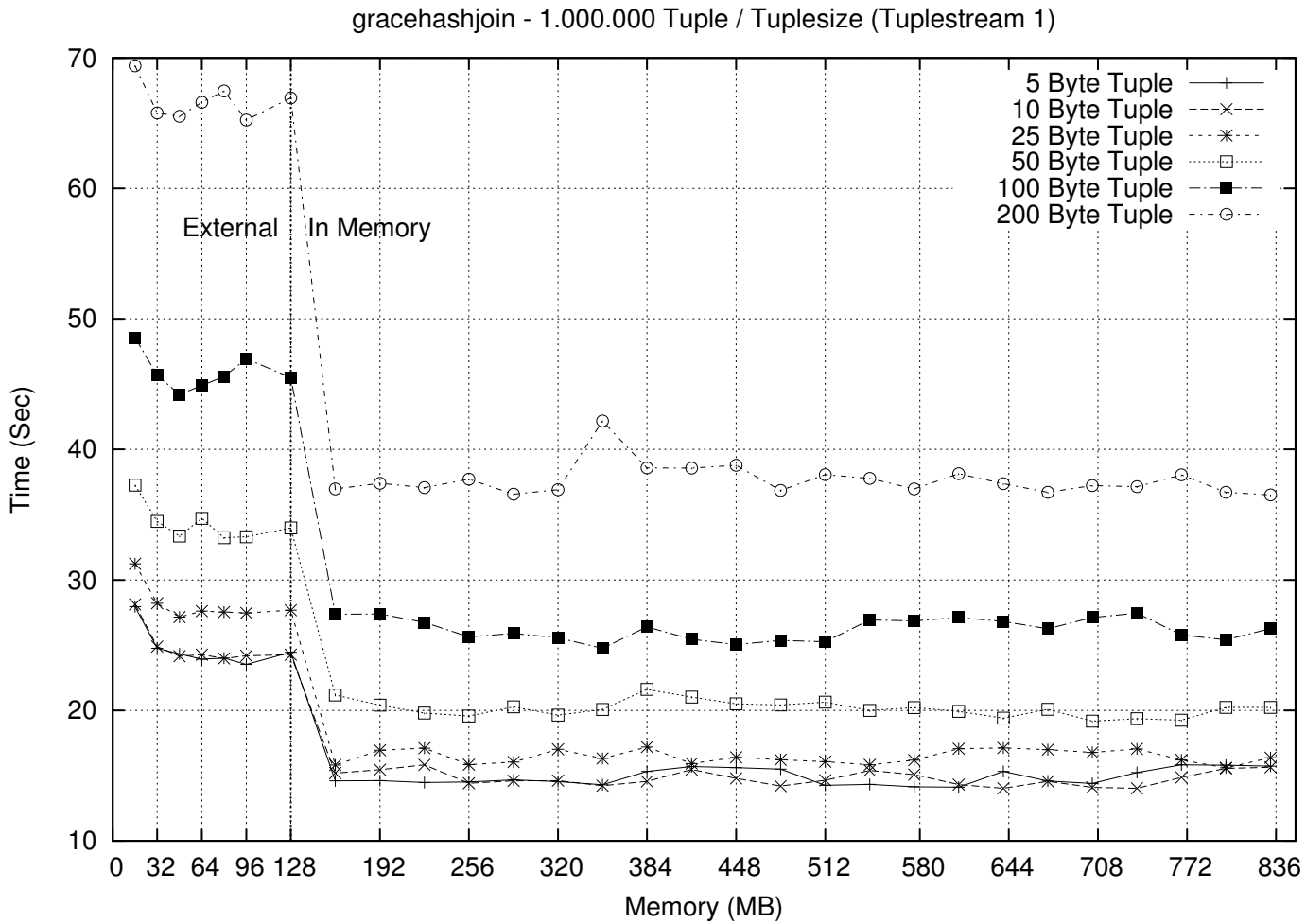
Laufzeitverhalten von itHashJoin



Laufzeitverhalten gracehashjoin - 1.000 Buckets



Laufzeitverhalten gracehashjoin - 1.000.000 Buckets



Bewertungskriterien Fortschrittsschätzung

Wie kann die Qualität einer Fortschrittsschätzung bewertet werden?

$Error_{Max}$

Maximale Abweichung zwischen Soll- und Ist-Wert.

$Error_{Int}$

Fläche zwischen Soll und Ist-Wert.

Bewertungskriterien Fortschrittsschätzung

Wie kann die Qualität einer Fortschrittsschätzung bewertet werden?

$Error_{Max}$

Maximale Abweichung zwischen Soll- und Ist-Wert.

$Error_{Int}$

Fläche zwischen Soll und Ist-Wert.

Bewertungskriterien Fortschrittsschätzung

Wie kann die Qualität einer Fortschrittsschätzung bewertet werden?

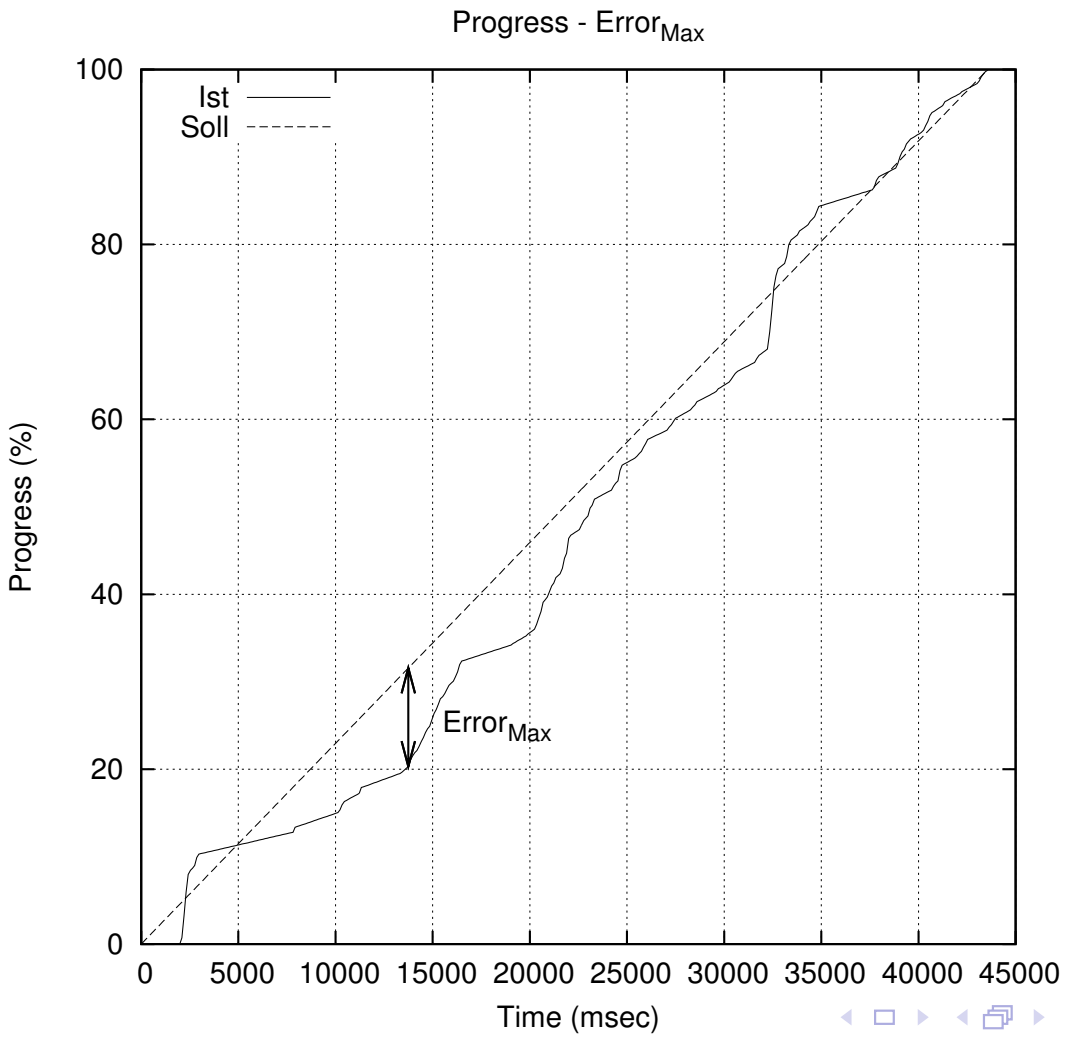
$Error_{Max}$

Maximale Abweichung zwischen Soll- und Ist-Wert.

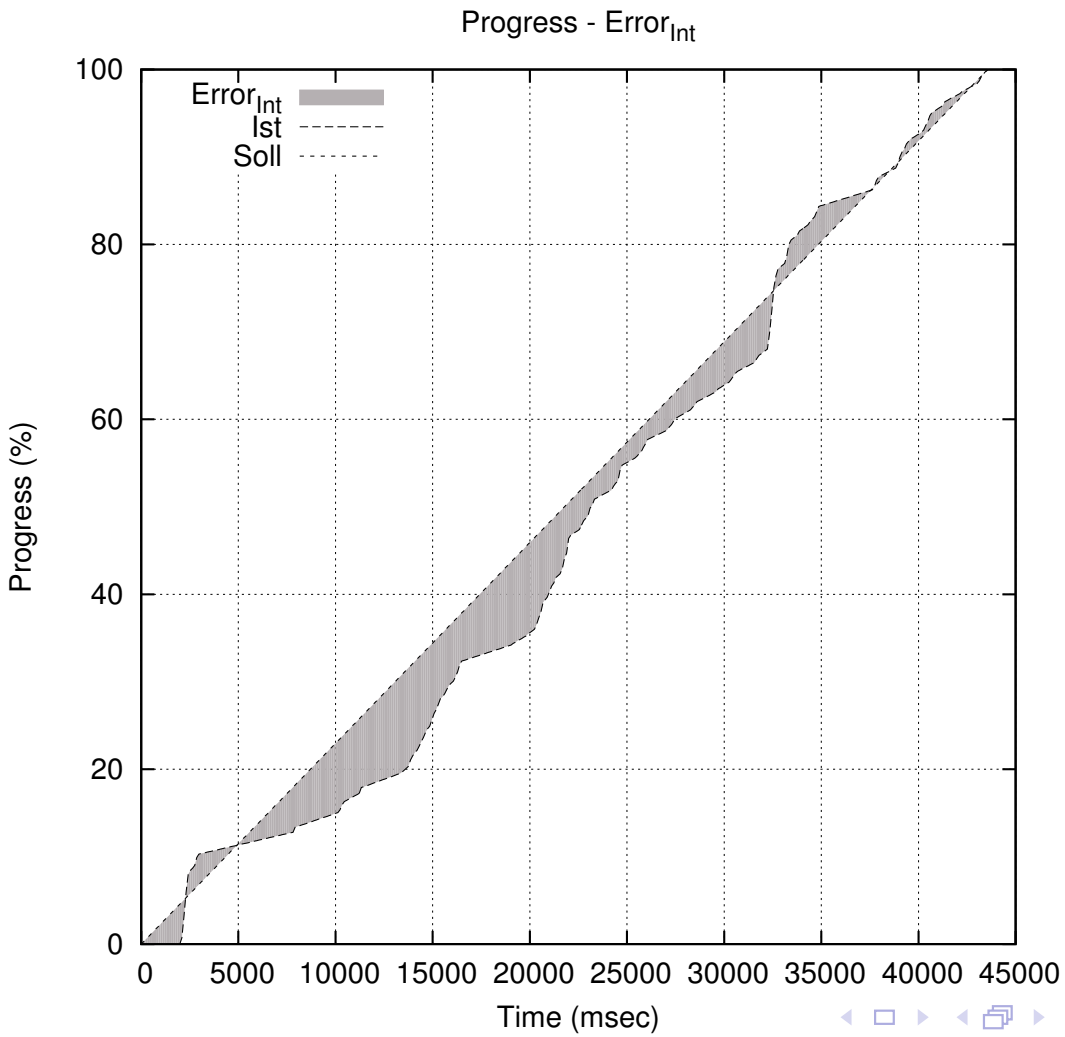
$Error_{Int}$

Fläche zwischen Soll und Ist-Wert.

Bewertungskriterien Fortschrittsschätzung - $Error_{Max}$

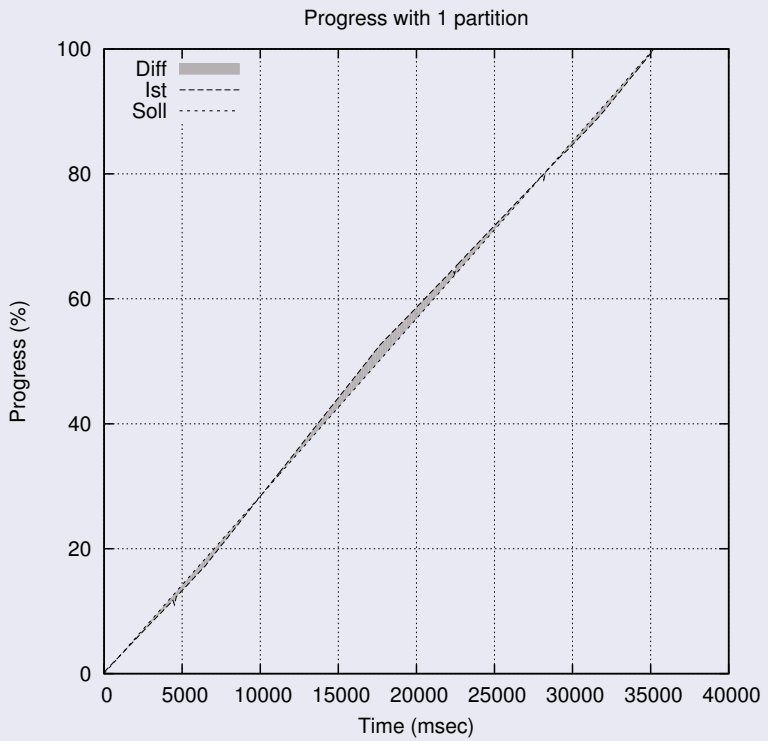


Bewertungskriterien Fortschrittsschätzung - $Error_{Int}$



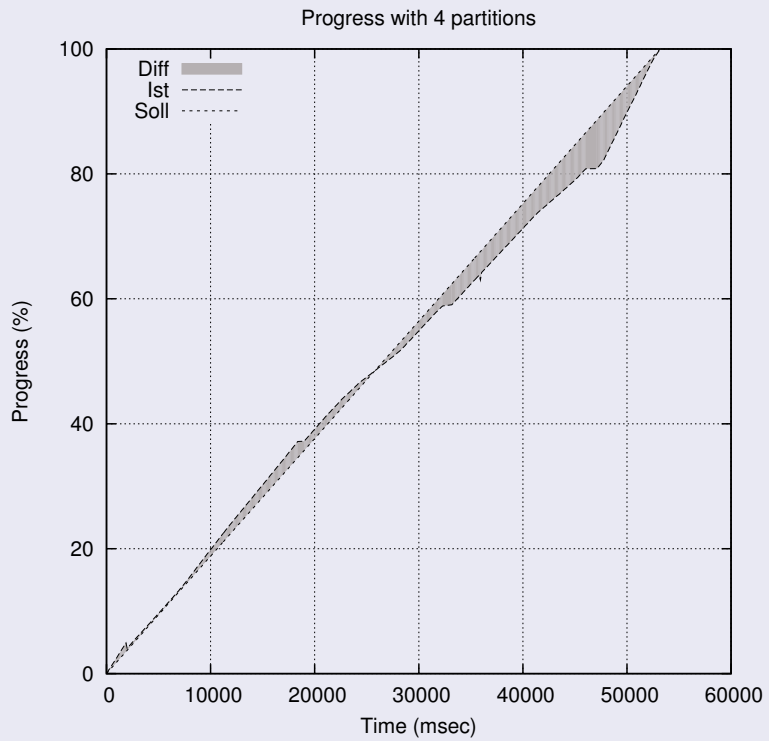
Fortschrittsschätzung itHashJoin

Eine Partition



$Error_{Max}$: 8.58, $Error_{Int}$: 30467.8

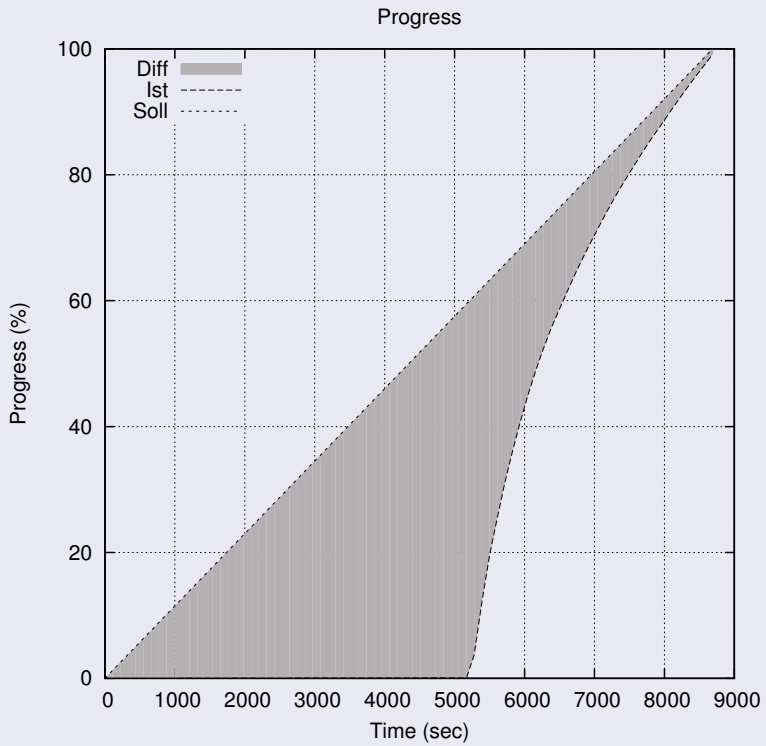
Vier Partitionen



$Error_{Max}$: 7.77, $Error_{Int}$: 127070.5

Fortschrittsschätzung sortmergejoin

Default-Selektivität 0.1



$Error_{Max}$: 59.43, $Error_{Int}$: 212316.5

Fortschrittsschätzung sortmergejoin - Probleme

Zwei Phasen

Sort-Phase: Beide Tupelstöme werden sortiert.

Merge-Phase: Die sortierten Tupelstöme werden parallel durchlaufen und Ergebnistupel werden erzeugt.

- Gewichten der Phasen nur möglich, wenn Aufwand beider Phasen bekannt
- Aufwand der Merge-Phase ist von Selektivität abhängig
- Selektivität steht erst am Ende der Berechnung fest
- Schätzungen der Selektivität erst ab 50 zurückgelieferten Tupeln

Fortschrittsschätzung sortmergejoin - Probleme

Zwei Phasen

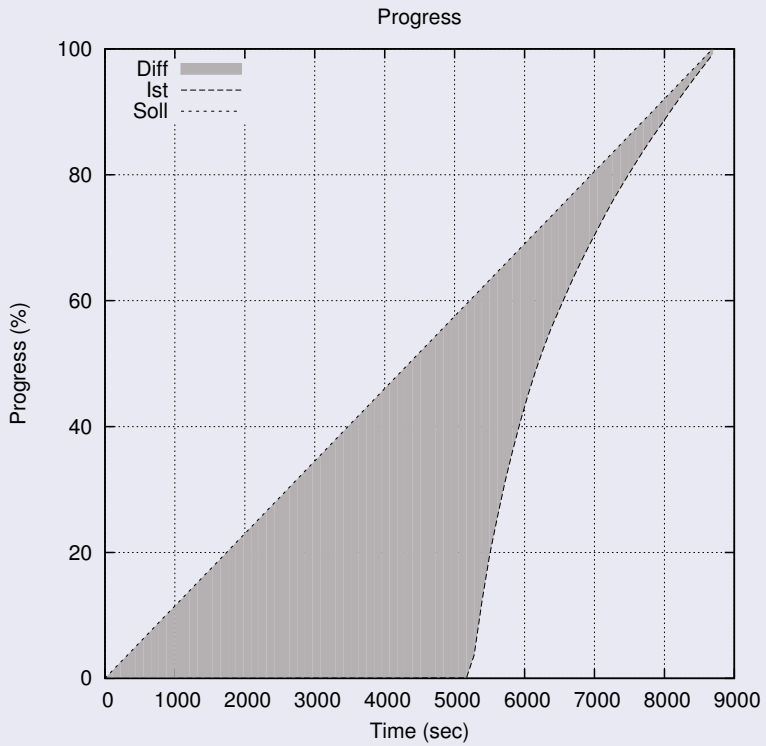
Sort-Phase: Beide Tupelstöme werden sortiert.

Merge-Phase: Die sortierten Tupelstöme werden parallel durchlaufen und Ergebnistupel werden erzeugt.

- Gewichten der Phasen nur möglich, wenn Aufwand beider Phasen bekannt
- Aufwand der Merge-Phase ist von Selektivität abhängig
- Selektivität steht erst am Ende der Berechnung fest
- Schätzungen der Selektivität erst ab 50 zurückgelieferten Tupeln

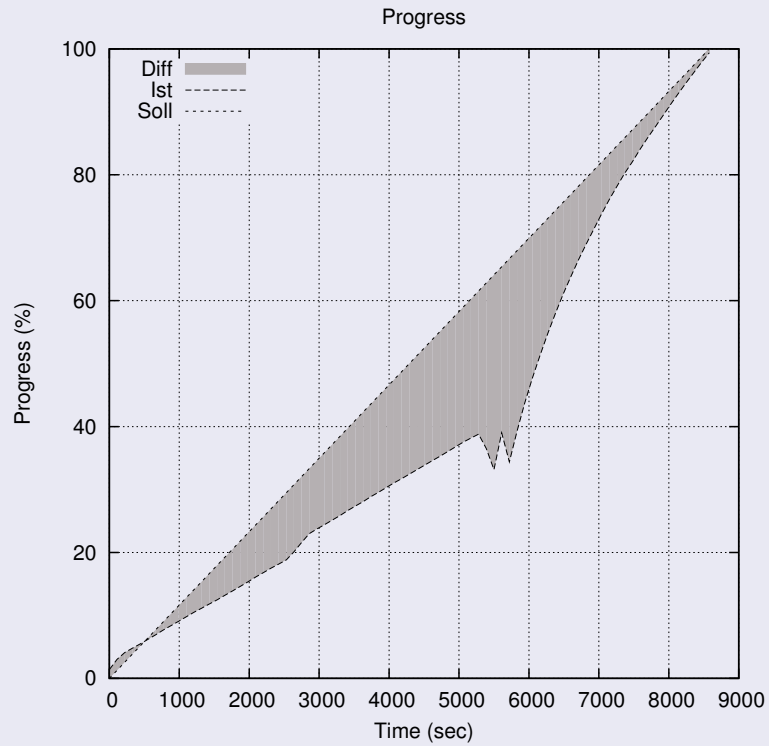
Fortschrittsschätzung sortmergejoin

Default-Selektivität 0.1



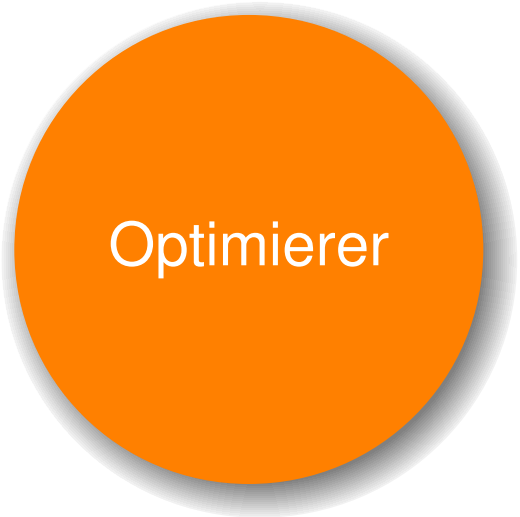
$Error_{Max}$: 59.43, $Error_{Int}$: 212316.5

Default-Selektivität 0.0001

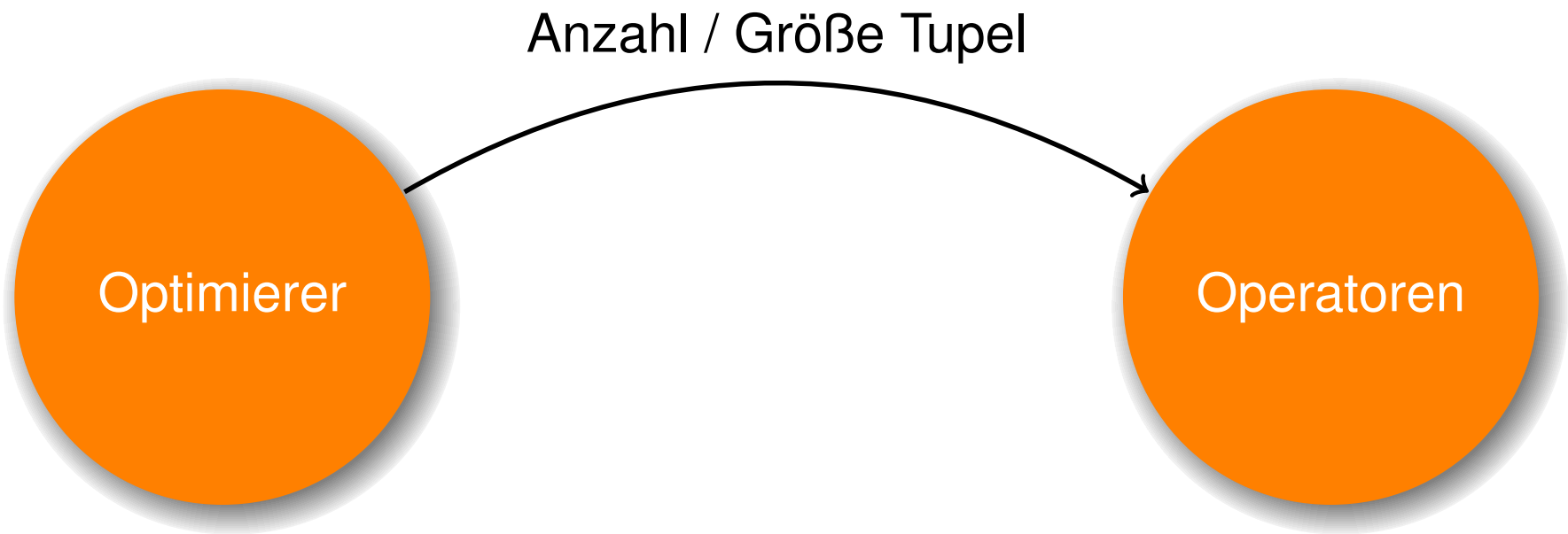


$Error_{Max}$: 32.30, $Error_{Int}$: 95463.5

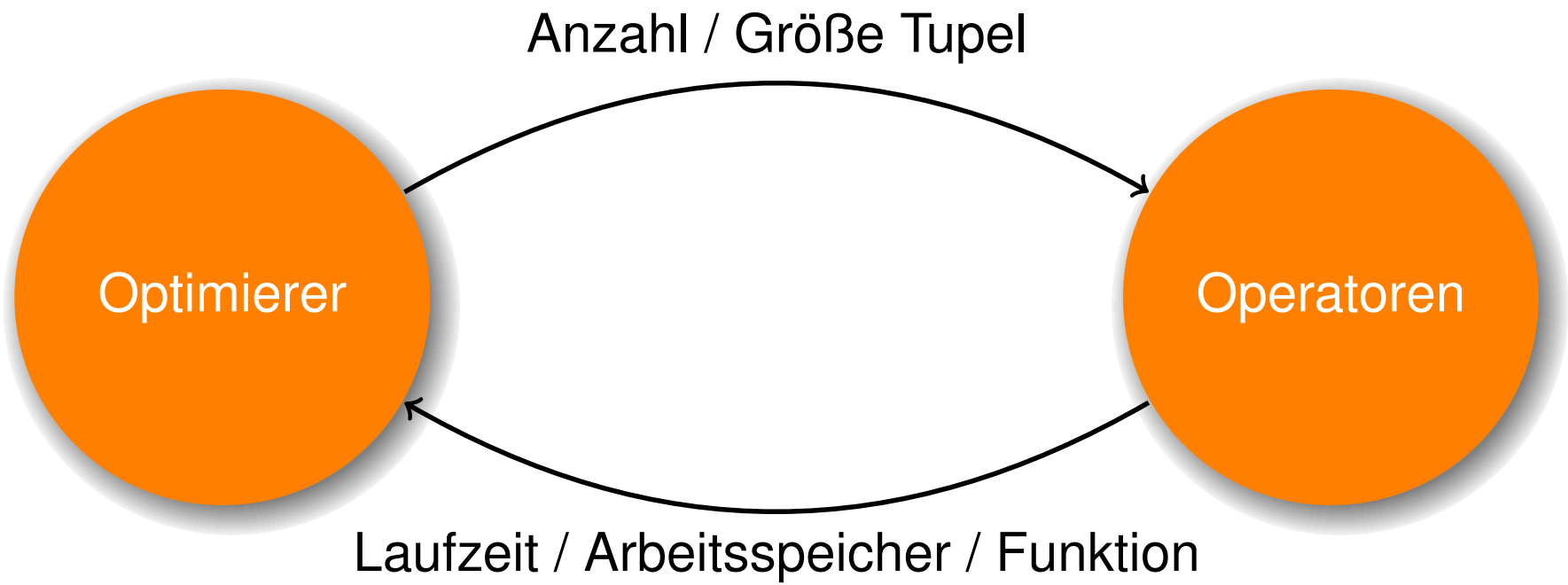
Anbindung des Optimierers



Anbindung des Optimierers



Anbindung des Optimierers



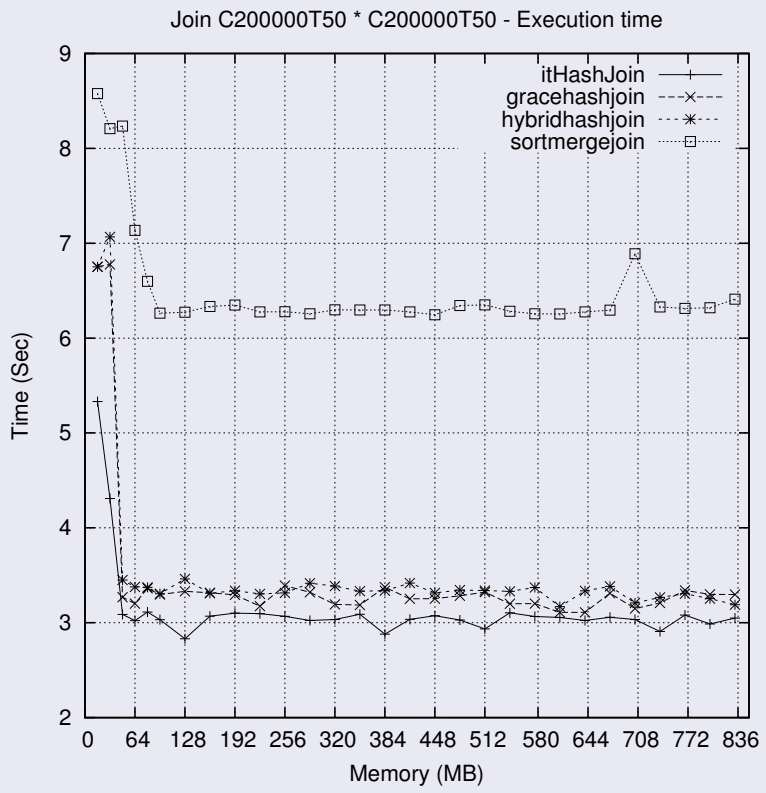
Anbindung des Optimierers - Schwierigkeiten

Schwierigkeiten

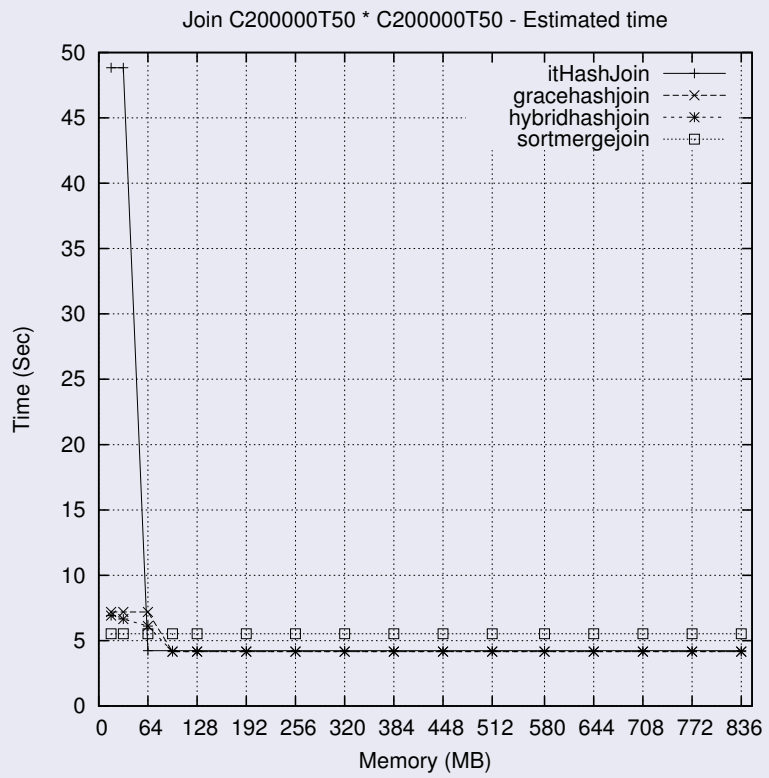
1. Selektivität ist nicht bekannt
2. Speicherverbrauch von Datenstrukturen ist schwer vorherzusagen
3. In den Kostenmodellen sind nur dominante Kosten enthalten
4. Die Berechnung kann nicht beobachtet werden
5. Ungenauer als eine Kostenschätzung im kalten Zustand
6. Schätzung ohne Operatorbaum teilweise nicht möglich (Parameterfunktionen)

Laufzeiten für den Optimierer

Reale Laufzeit

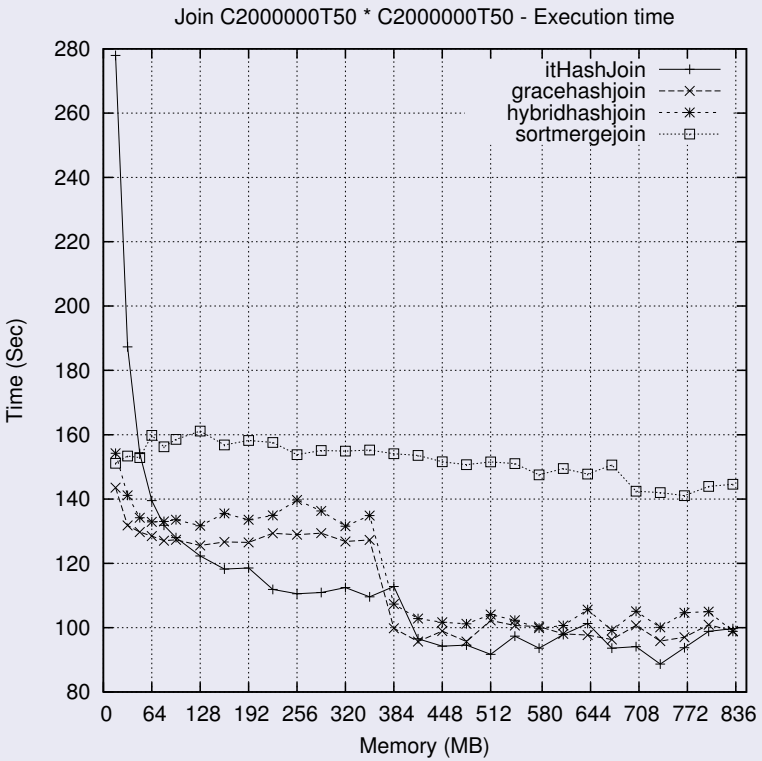


Geschätzte Laufzeit

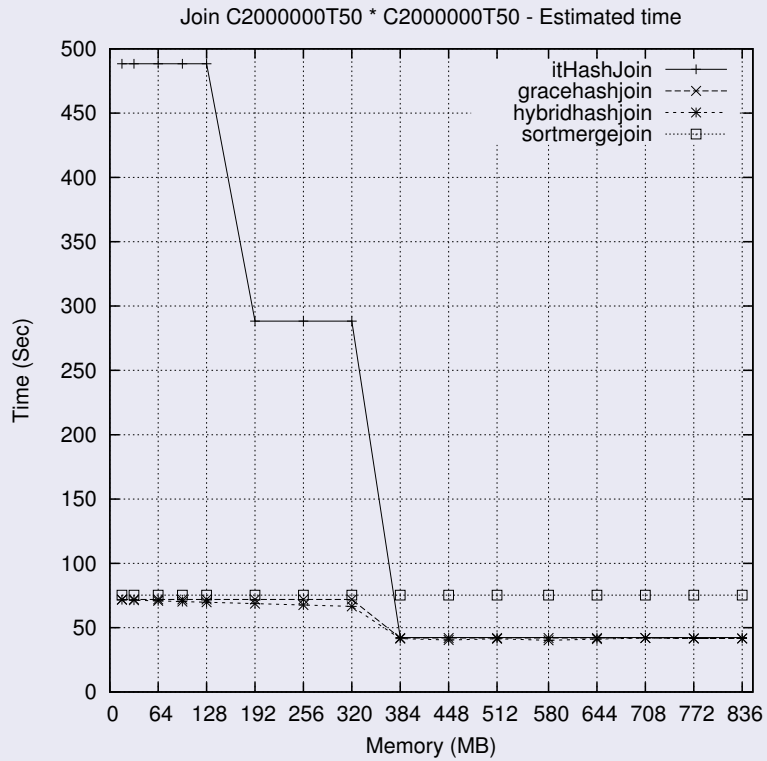


Laufzeiten für den Optimierer

Reale Laufzeit



Geschätzte Laufzeit



Approximation der Laufzeit durch eine Funktion

Idee

Der Zusammenhang zwischen Laufzeit und zugeteiltem Arbeitsspeicher soll durch eine parametrisierbare Funktion approximiert werden.

Nutzen

Diese Funktion hilft dem Optimierer bei der Entscheidung, welcher Operator wie viel Arbeitsspeicher erhalten soll.

Approximation der Laufzeit durch eine Funktion

In *SECONDO* hinterlegte Funktionen:

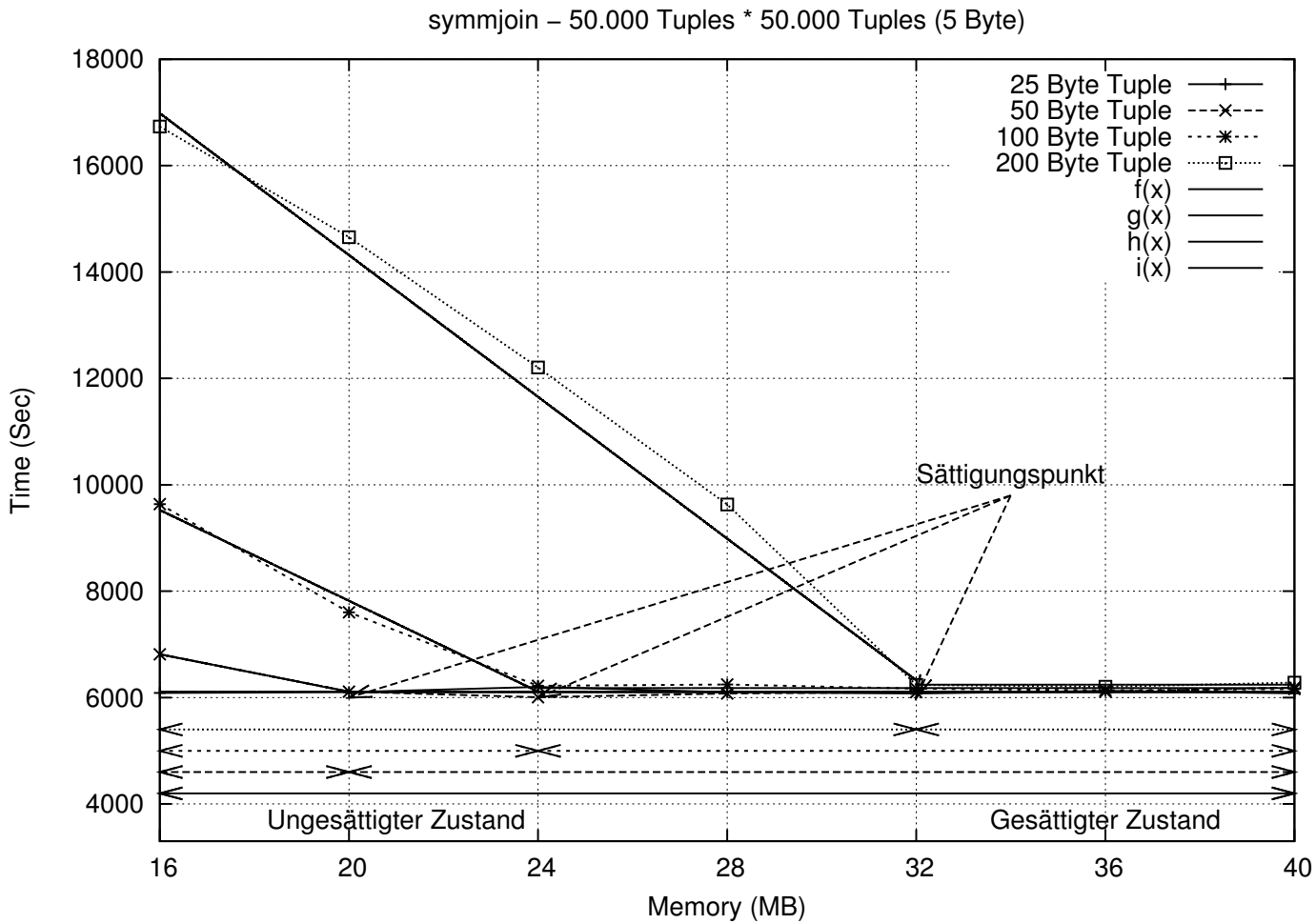
Lineare Funktion

$f(x) = mx + b$

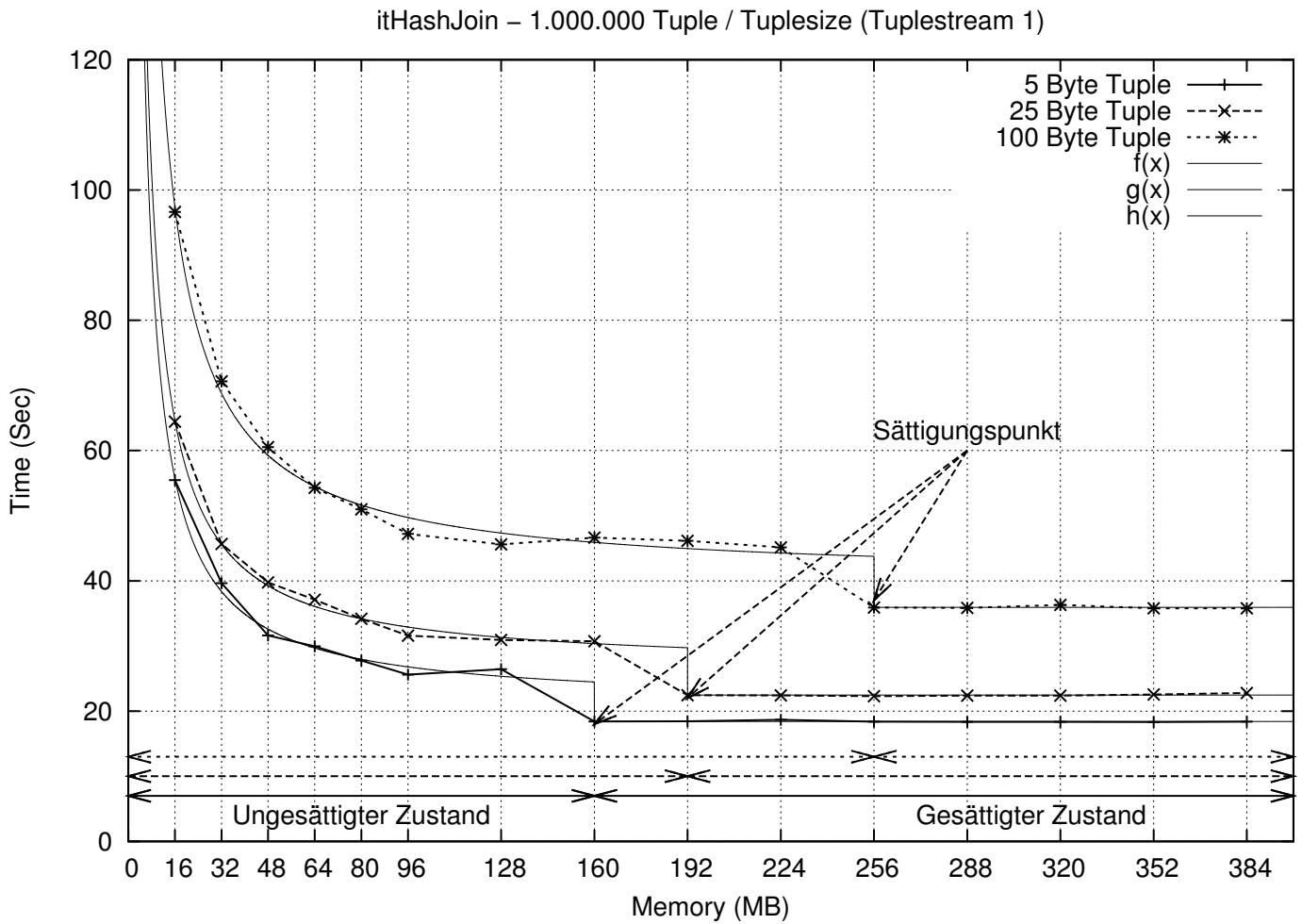
Antiproportionale Funktion

$f(x) = \frac{h}{x} + b$

Curve fitting - Funktion $f(x) = mx + b$



Curve fitting - Funktion $f(x) = \frac{h}{x} + b$



Zusammenfassung

Es wurde(n) . . .

- . . . neue *Kostenmodelle* implementiert und bestehende Kostenmodelle verbessert
- . . . die *RelationAlgebra* auf das neue Framework portiert
- . . . Konstanten aus den Kostenmodellen ausgelagert
- . . . neue *Hilfmittel* entwickelt
- . . . das *Laufzeitverhalten* von Operatoren durch Funktionen approximiert

Vielen Dank für Ihre Aufmerksamkeit