Chain:

Operator Scheduling For Memory Minimization In Data Stream Systems.

Brian Babcock, Shivnath Babu,
Mayur Datar, and Rajeev Motwani.

ACM SIGMOD Int. Conf. on Management of Data, pages 253-264, 2003.

Presented by:
Kareem Elgebaly

Outline:

- Introduction.
- Intuition.
- Proof.
- Experiments.
- Conclusion.
- Assessment.

Introduction:

Problem:

Data stream data rates are not only fast but also irregular.

(2 orders of magnitude)

Introduction:

Solutions so far:

Drop tuples: (e.g. Load Shedding {Aurora,STREAM})
Loss of data leads to inaccuracies.

Overflow on disk:
Disastrous performance degradation.

Introduction:

Answer:
Reduce memory needed for queuing.

How ?
Through better scheduling.

Why ?
No penalties in performance or accuracy.

Introduction:

Devise a scheduler that discriminates among operators according to their memory impact.

Fast Operators:
Expected to have a very fast run-time.

Selective Operators:
Operators that consume a lot of records.

Intuition:

Two dimensional problem:

Fast + Selective → **High Priority**

Slow + Unselective → **Low Priority**

Intuition:

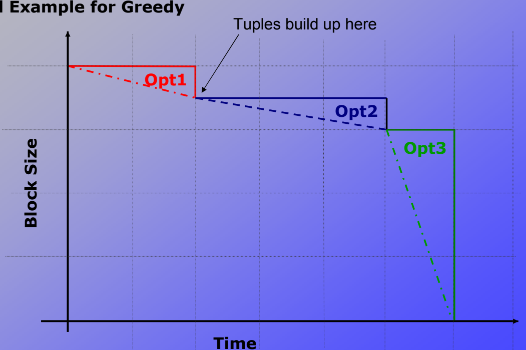
How many tuples per unit time does the Op consume?

Greedy evaluation:

Priority α (Selectivity / Time)

Intuition:

Bad Example for Greedy



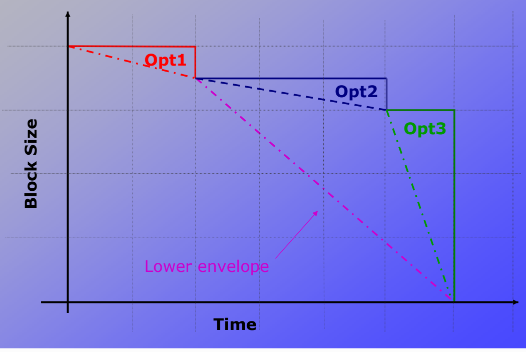
Intuition:

Answer is not straight forward…

A very good operator that takes results from a bad operator will never get scheduled.

(Local Minima)

Intuition:



Intuition:

Chain evaluation:

Priority α Lower Envelope Slope

Proof:

Claim:

Memory needed by Chain scheduling is within constant factor of optimal offline algorithm.

(Clairvoyant)

Proof sketch:

1. Greedy scheduling is optimal for convex progress charts
   (since) Best operators are immediately available
2. Lower envelope is convex
3. Lower envelope closely approximates actual progress chart

Proof:

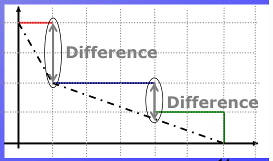
Claim:

Lower envelope closely approximates actual progress chart

1. At most one block in the middle of each lower envelope segment
   (Due to) tie-breaking rule

2. (Lower envelope + 1) gives upper bound on actual memory usage

3. Additive error of 1 block per progress chart



Experiments:

Setup:

Data Sets:

1) Synthetic Data Set.
2) Real Data Set.

Queries:

1) Single Queries.
2) Multi Queries.
3) Join Queries.

Experiments:

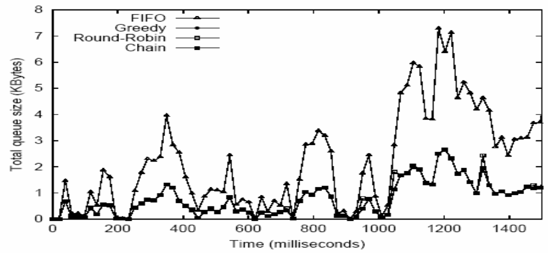


Figure 3: Queue size vs. Time (Single stream, two operators, synthetic data set)

Experiments:

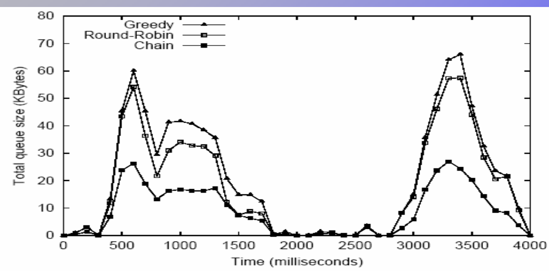


Figure 4: Queue size vs. Time (Single stream, four operators, real data set)

Experiments:

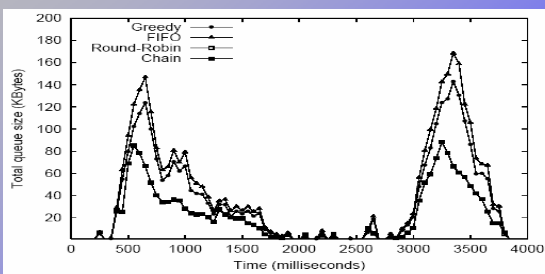


Figure 7: Queue size vs. Time (sliding-window join and 3 selections, real data set)

Conclusions:

	Pros.	Cons.
FIFO	No starvation	Poor performance
Round-Robin	No starvation	Poor performance
Greedy	Good performance	Starvation Stuck in local maxima
Chain	Near optimal performance	Starvation

Conclusions:

Chain is orthogonal to traditional memory requirements minimization techniques. Hence you are not trading Chain's benefits with anything, you are getting it for free.

Chain is an algorithm that guaranties certain performance standard without introducing any extra over heads.

Good !!!

Assessment:

Limitation:

Chain is no better than FIFO in normal rates.
Chain is most useful when rates are irregular.

(plus) no experimentation done to compare performance in such cases.

What if:

The SDMS was implementing an Early Selection optimization technique ?

Would Chain make sense?
Would it be any better than greedy?

Assessment:

TODO:

More QOS guarantees. Like (low response time)
tuples may wait for an unacceptable long time before it gets scheduled.

Chain doesn't take into account:

1) Parallelism
2) Shared sub plans (shared queues)

Questions?