OPERATOR SCHEDULING IN A DATA STREAM MANAGER

Authors: ***D. Charney*[†], *U.Çetintemel*[†], *A.Rasin*[†], *S.Zdonik*[†], *M.Cherniack*[§], *M.Stonebraker****

[†]Brown University

[§]Brandeis University

*MIT

Sedat Behar and Yevgeny Ioffe

REFERENCES

- **Aurora**: A New Class of Data Management Applications (2002)

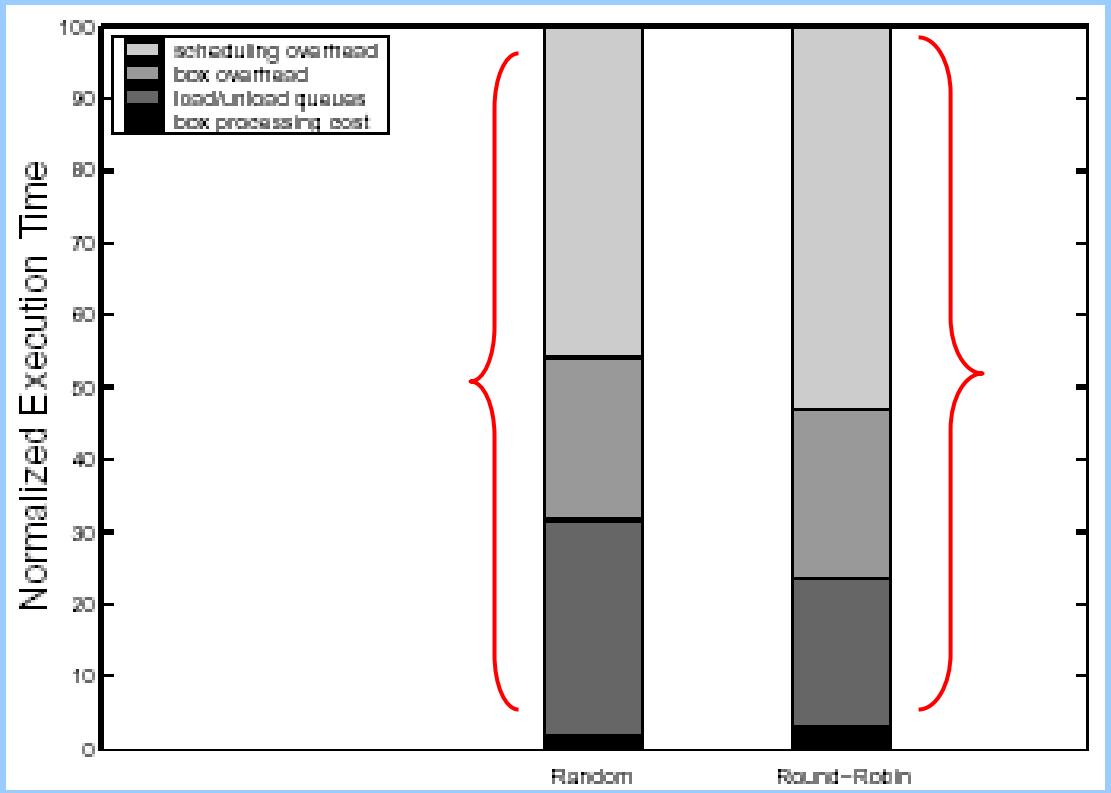
- **NiagaraCQ**: A Scalable Continuous Query System for Internet Databases (2000)

AURORA OVERVIEW

- **sensors → router → storage manager or external apps (tuple queuing) → scheduler**
- **scheduler → box processor → router → output**
- **QoS monitor => load shedder**

AURORA OVERVIEW

- Motivation:



- Key component is **scheduler**
  - fine-grained control over CPU allocation
  - dynamic scheduling-plan construction
  - latency based priority assignment

EXECUTION MODEL

- **thread-based** vs. **state-based**
  - cons of thread-based model: not scalable (OS)
  - pros of state-based model: fine-grained control of CPU and batching of operators/tuples.

- how to design smart, **yet** cheap scheduler?!

SCHEDULING - OP BATCHING

- **superbox** = sequence of boxes scheduled as a single group
  - reduce scheduling overhead
  - don't have to access storage manager every time
  - each is always a tree rooted at output box

- Two levels of scheduling:
  - **WHICH** superbox to process?
  - **IN WHAT ORDER** should boxes be executed?

SCHEDULING - SUPERBOX SELECTION ALGORITHMS

- Application-spanner (**AS**) – static
  - 1 superbox for each query tree
  - # of superboxes = # of continuous queries
- Top-k-spanner (**TKS**) – dynamic
  - identify tree rooted at output box spanning **top k** highest priority boxes for a given app (see figure later)
  - priorities determined by:
    - latencies of tuples residing on each box's input queues
    - QoS specifications

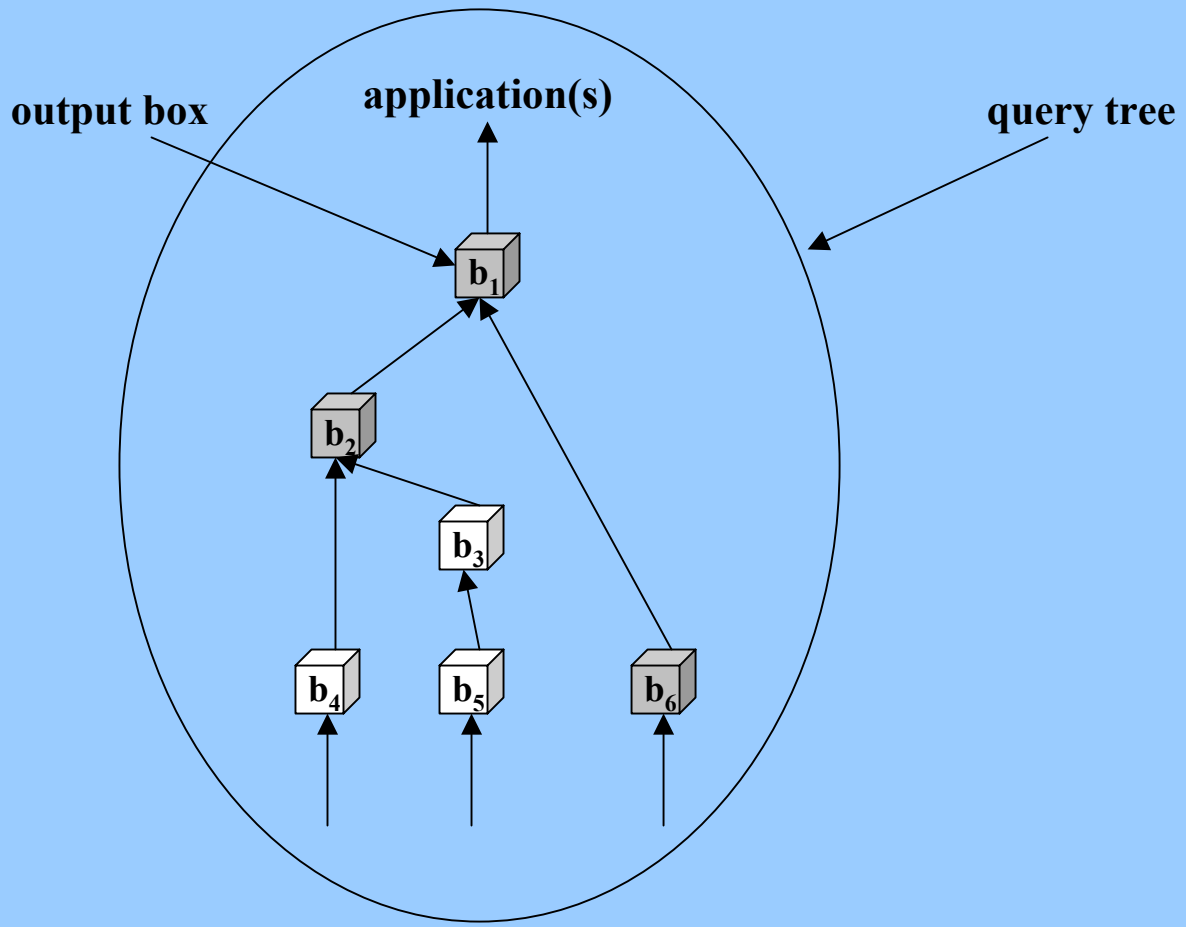
SCHEDULING - SUPERBOX TRAVERSAL ALGORITHMS

- **Min-cost** (MC)
  - Traverse the superbox in *post-order*
  - minimize # box calls/output tuple
- **Min-Latency** (ML)
  - output tuples as fast as possible
- **Min-Memory** (MM)
  - schedule boxes to yield max increase in free memory

TRAVERSAL – DETAILS (MC)

INITIALLY	
b_4	a
b_5	b
b_3	c
b_2	d
b_6	e
b_1	f

*



In case of MC: $b_4 \rightarrow b_5 \rightarrow b_3 \rightarrow b_2 \rightarrow b_6 \rightarrow b_1$

Total time to output all tuples: **15p + 6o** **WHY?**

Average output tuple latency: **12.5p+o** **WHY?**

*courtesy of Mitch

TRAVERSAL – ML and MM

- For Min-Latency traversal, we have the following:

$$o\_sel(b) = \prod_{k \in D(b)} sel(k)$$

$$output\_cost(b) = \sum_{k \in D(b)} cost(k) / o\_sel(k)$$

where **sel(b)** = estimated selectivity of a box ***b;*** **o_sel(b)** = output selectivity of box ***b;*** **D(b)** is $\{b_1, \ldots, b_n\}$ downstream.

- For MM we have the following:

$$mem\_rr(b) = \frac{tsize(b) \times (1 - selectivity(b))}{cost(b)}$$

where **tsize(b)** = size of tuple in ***b***'s input queue.

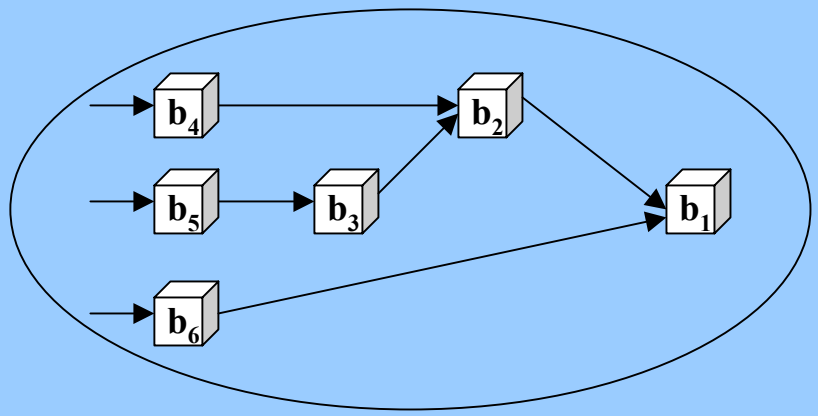
Intuitively, **mem_rr(b)** means the expected memory reduction rate for a box ***b*** – i.e. **by** how much does this box **maximize available memory**?

SCHEDULING - TRAIN

- *train* = sequence of tuples batched in 1 box call
  - reduce overall tuple processing costs

- WHY/HOW? 3 reasons:
  1. Given a fixed # of tuples, decrease total # of box calls (thus?)
  2. Improve memory utilization (how?)
  3. Some operators may optimize better with more tuples in queues (why?)

EXPERIMENTS



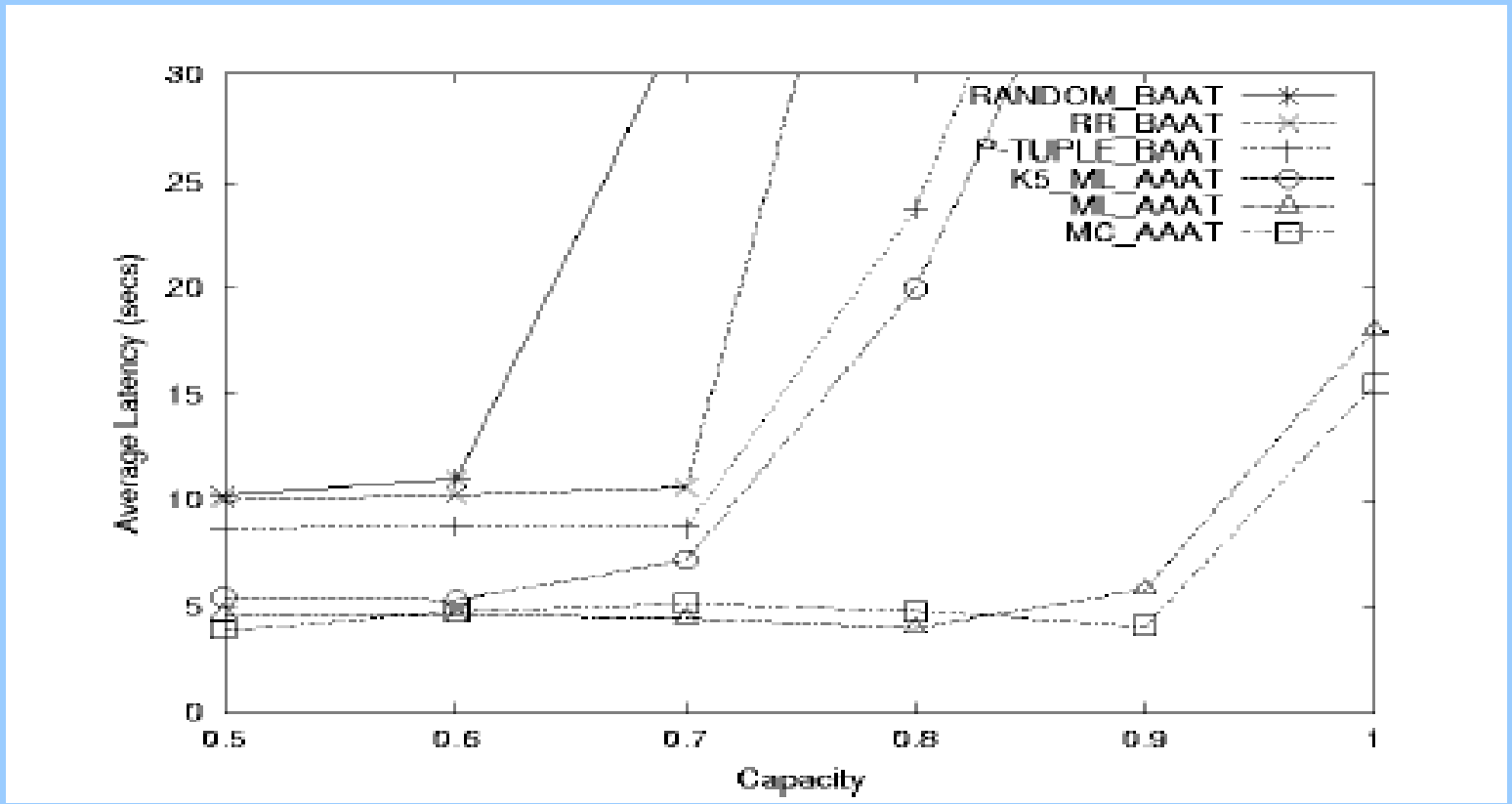
Application Tree:
Query as a tree of boxes rooted at an output box

Depth: Number of Levels in the application tree

Fan-in: Average number of children for each box

Capacity: Overall load as an estimated fraction of the *ideal* capacity

EXPERIMENTS



BAAT: Box-at-a-time
AAAT: Application-at-a-time
ML: Min. Latency
MC: Min. Cost
RR: Round-Robin

EXPERIMENTS

Results:

- As the arrival rates increase, the queues will saturate
- BAAT is not a good idea
- ML and MC are high-load resistant in AAAT
- As k increases, top-k-spanner algorithm approaches the AAAT algorithm

EXPERIMENTS

Superbox Traversal:

Overhead difference between the traversals is proportional to the depth of traversed tree

MC incurs less Box Overhead cost then ML

Additional box calls when depth of tree is incremented:

ML: O ($d*f^{d+1}$)

MC: O (f^{d+1})

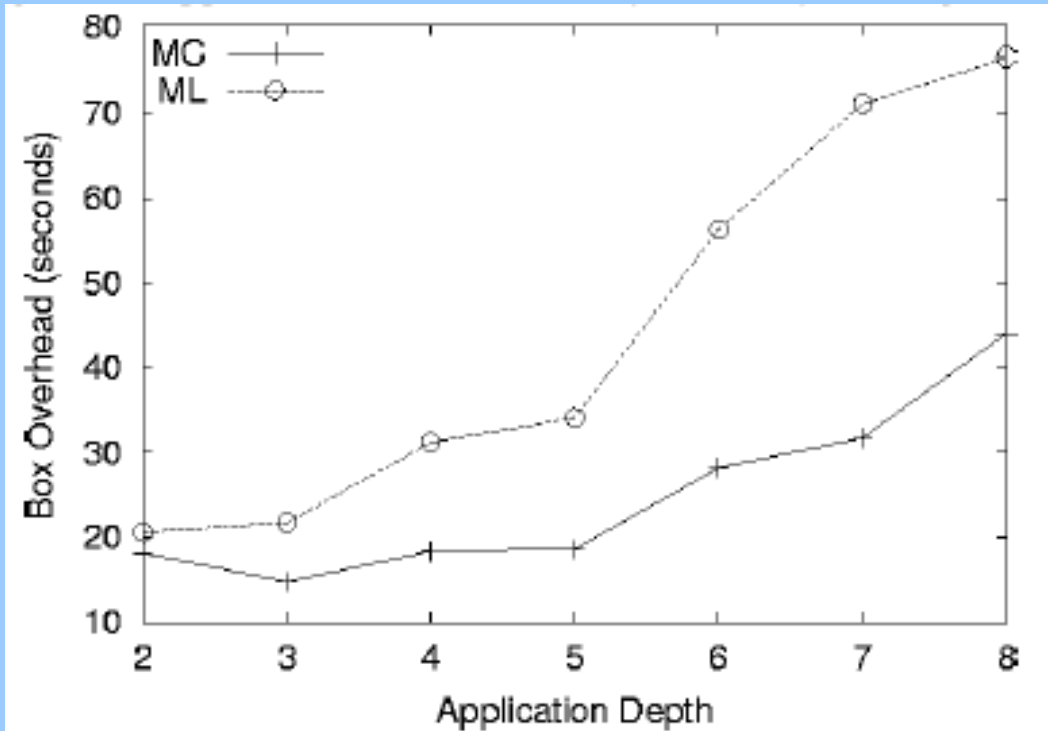
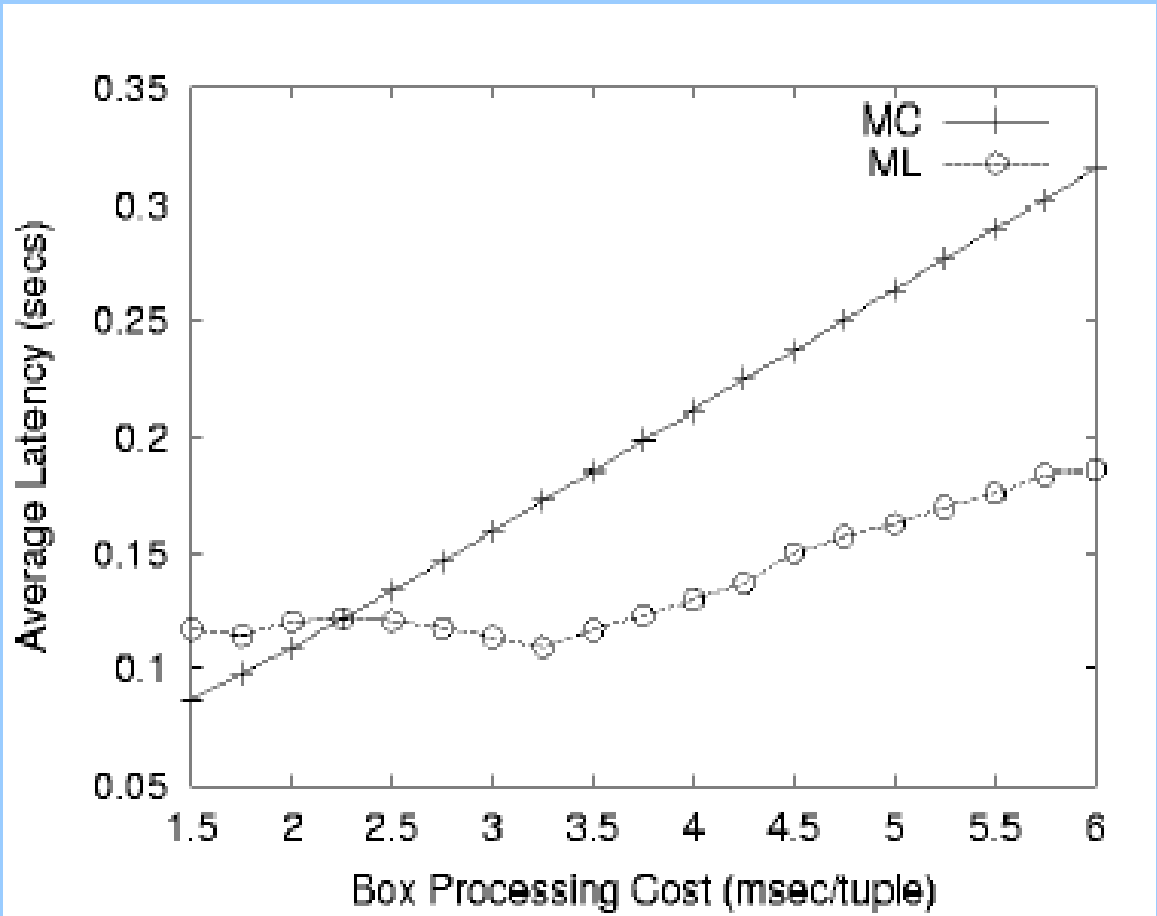


Figure 6: Box overheads for ML and MC

EXPERIMENTS

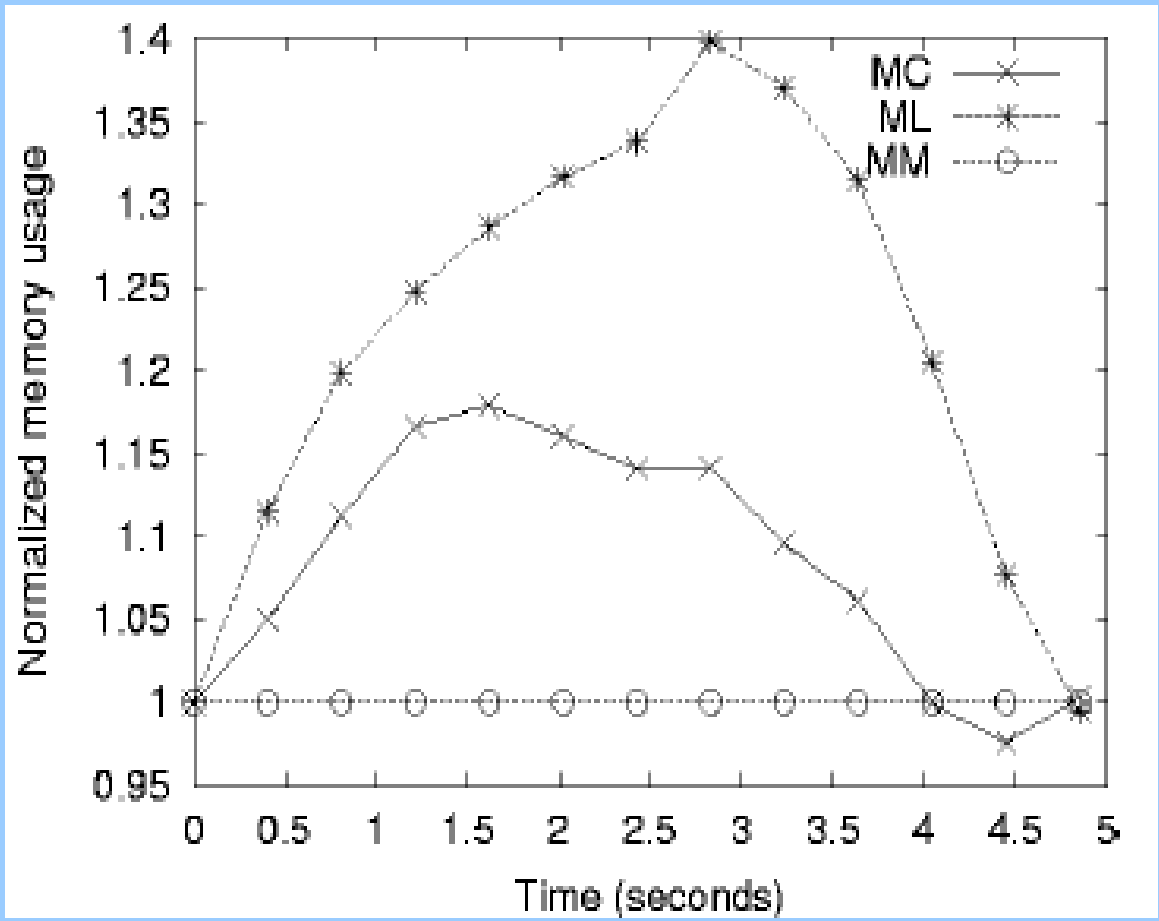
Superbox Traversal: Latency Performance



EXPERIMENTS

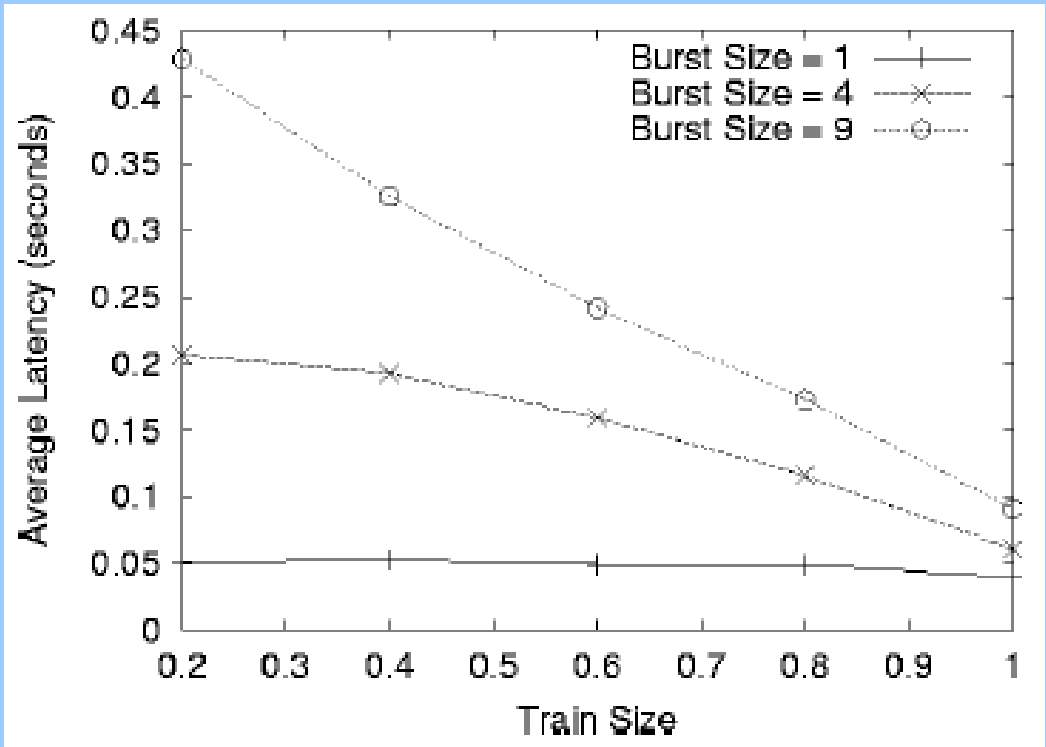
Superbox Traversal: Memory Requirements Over Time

- MC minimizes box overhead
- Common Query Network
- Same tuples are pushed through same boxes



EXPERIMENTS

- Tuple Batching – Train Scheduling

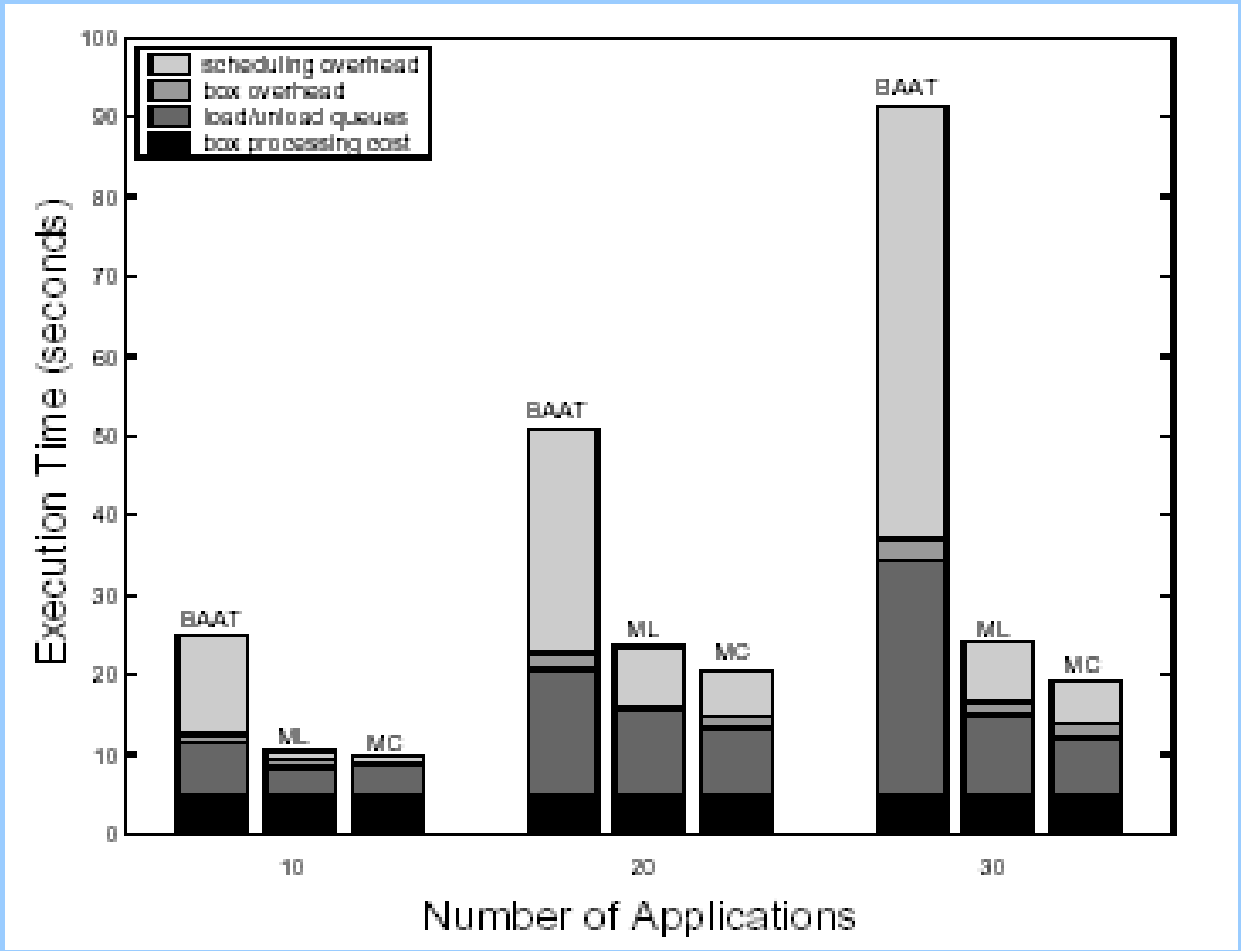


- Does not help when the system is lightly loaded

- As train size increases, more of the bursty traffic is handled

EXPERIMENTS

- Overhead Distribution



- Continuous queries, BAAT, ML, MC graphs under different workloads

- Train and Superbox are obviously good solutions.

- MC achieves smaller total execution times and reduced scheduling and box overhead

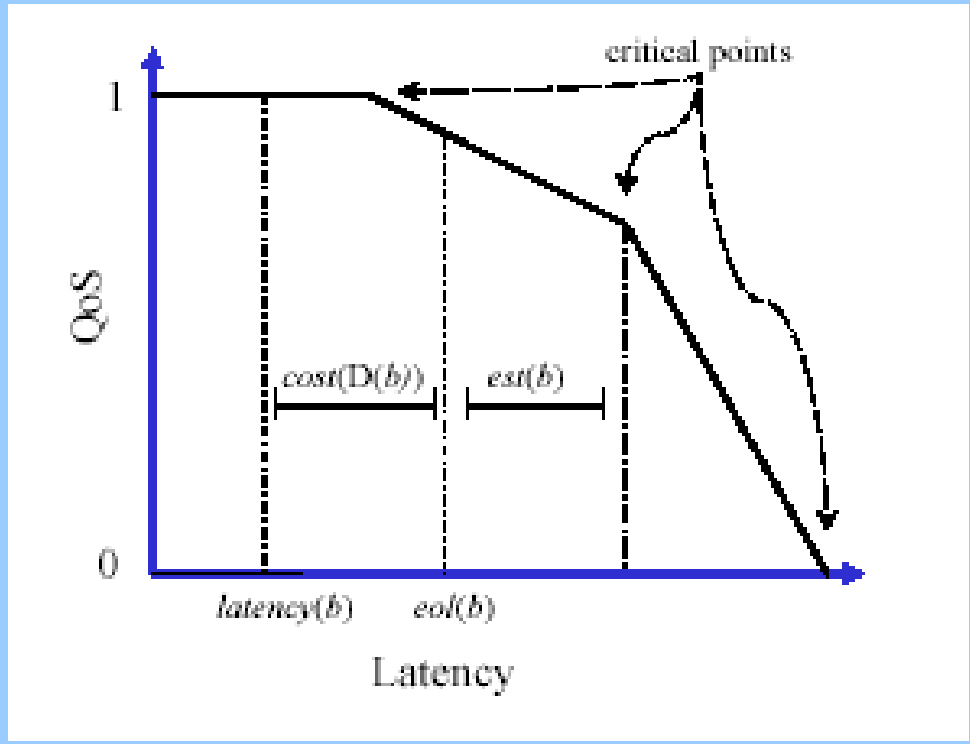
QoS Driven Priority Assignment

- Utility and Urgency
- Computing Priorities:
  Keep tracking the latency of the tuples in the queue and pick the tuples that has the most expected increase in the QoS Graph (not scalable)
- Utility:
  Expected QoS (per unit time) that will be lost if the box is not chosen for execution
- Expected Output Latency:
  An estimation of where box tuples are on the QoS latency curve at the corresponding output

QoS Driven Priority Assignment

Urgency Computation:

- Expected Slack Size: An indication of how close a box is to a critical point
- Urgency can be trivially computed by subtracting the expected output latency from latency value that corresponds to the critical point

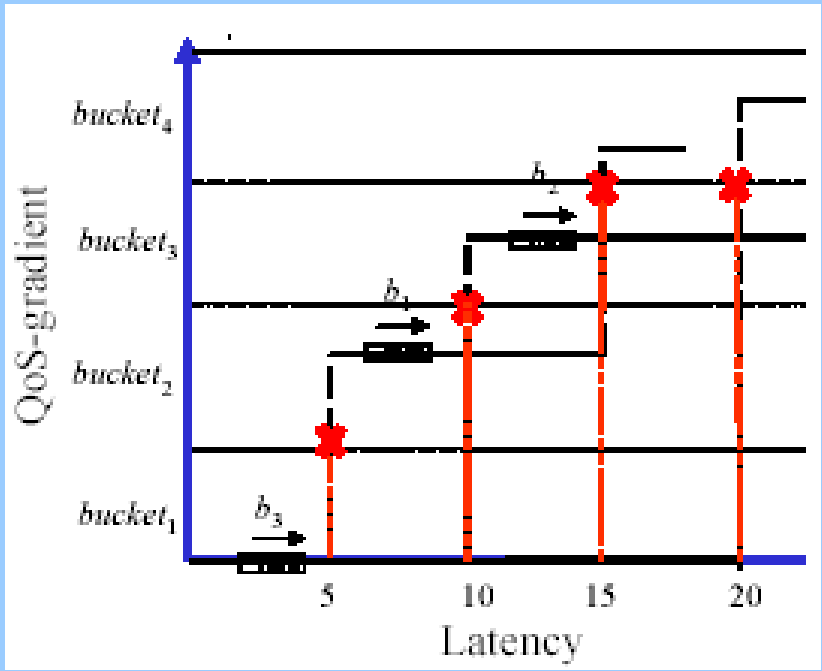


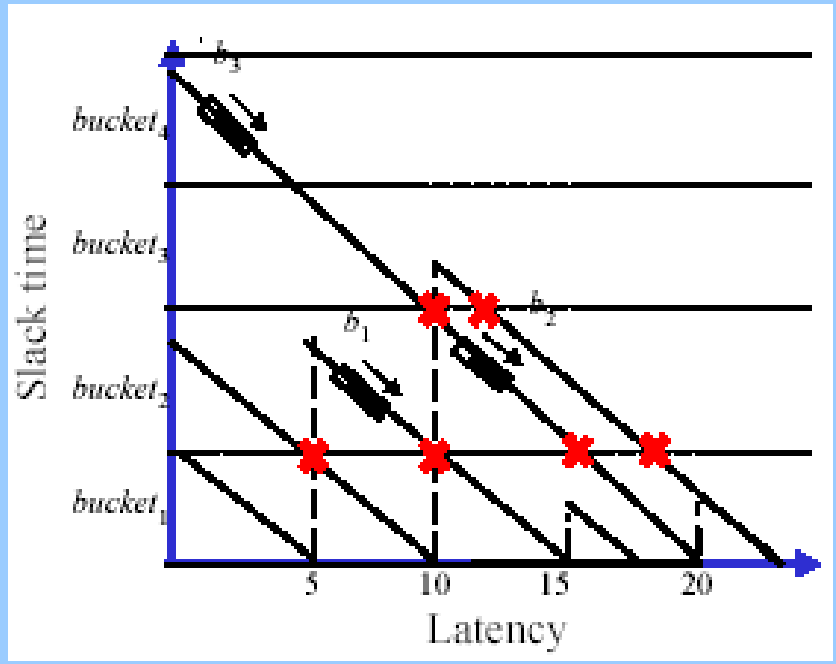
Combining Utility and Urgency: Choose the boxes with highest utility, and then choose the ones with minimum slack time. (i.e.; the most critical ones with highest utility)

QoS Driven Priority Assignment

- Approximation for Scalability
  Assign boxes to individual buckets based on their p-tuple value at runtime
- Gradient Buckets:
  - width of a bucket is a measure of the bound on the deviation from the optimal scheduling decision
- Slack Buckets:
  - deviation from optimal with respect to slack values

QoS Driven Priority Assignment





Bucket Assignment: O (n) time to go through the boxes and O (1) time computing the bucket for each box

Calendar Queue: A multi-list priority queue to make the bucket assignment overhead proportional to number of boxes that made a transition to another bucket

THINGS TO REMEMBER

- Processor Allocation Methods
- Train and Superbox Scheduling
- QoS Issues and Approximation Techniques

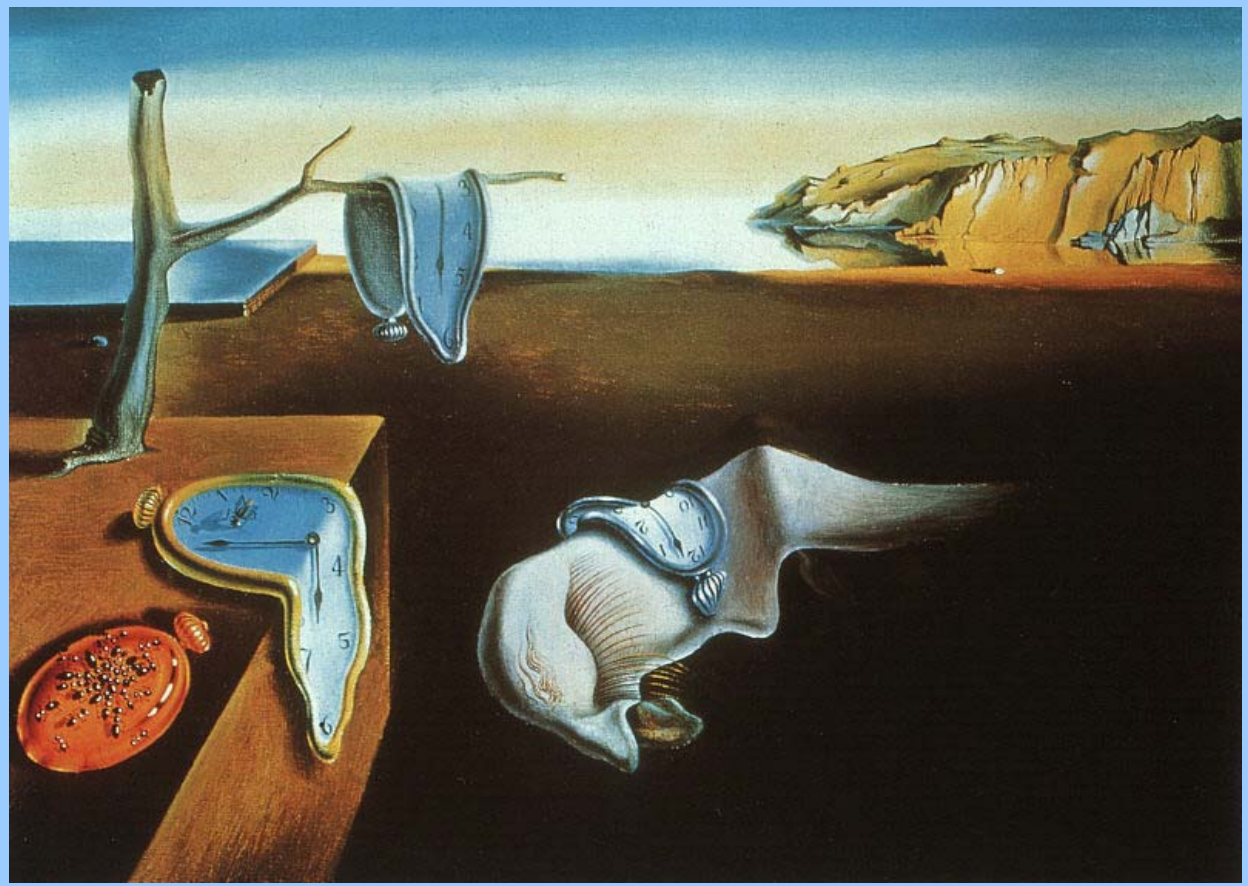
Discussion Questions

- What are the pros and cons of building notion of relative consistency into DSMS itself instead of its queries?
- Is it worthwhile to define QoS for a DSMS in terms of a ratio between queries? For example a relative periodic query scheduling policy.

Discussion Questions

- Should it possible to dynamically update the Relative Miss Ratio? What are some situations that would benefit from this? In streams?
- Low priority queries miss their deadlines and do not run, what is the parallel to this in a DSMS?

Persistence of Memory (Dali, 1931)



databases are persistent.

data streams, like memory, fade with time…