University of Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Distributed Systems Group

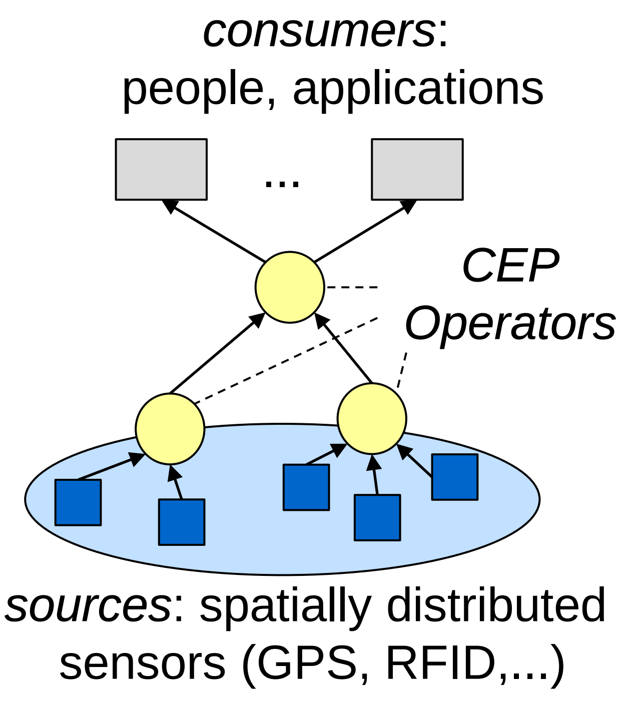
Minimizing Communication Overhead in Window-Based Parallel Complex Event Processing

Ruben Mayer, Muhammad Adnan Tariq, Kurt Rothermel

DEBS 2017

Motivation

- Timely reactions to situations in the surrounding world
  - Algorithmic trading, Internet of Things,...
  - Sensors gather low-level information
  - Complex Event Processing (CEP) operator networks detect events
- Big Data – new challenges for CEP
  - High event rates
  - Parallelization needed for operators



Example: Face Recognition Operator

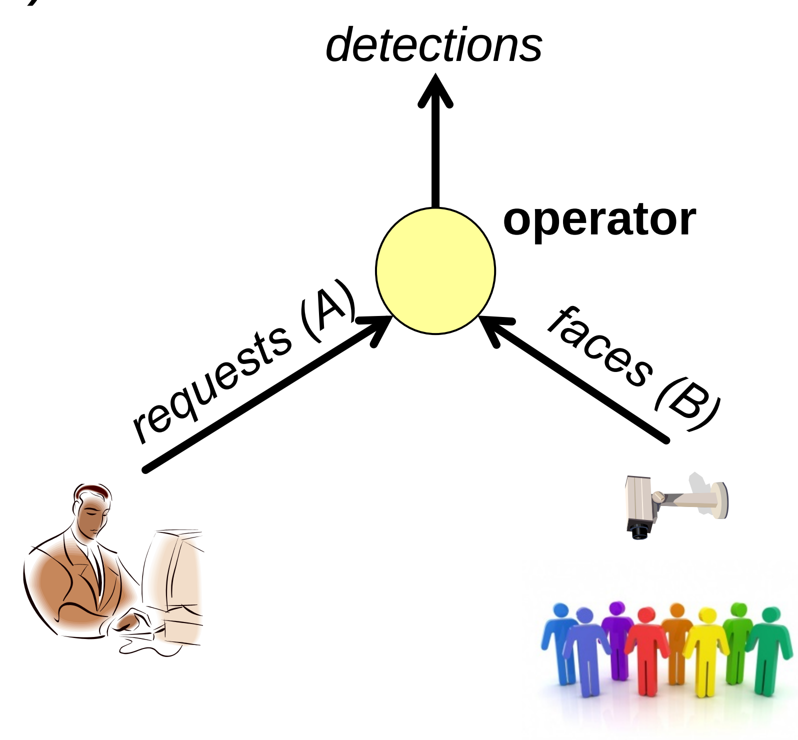
- Is a person of interest in the video stream?
- Query: Aperiodic(A; B; C) with

  A → <type = requested_person, time = t>

  B → <type = face, “face_match(A)”>

  C → time $\geq$ t + time frame

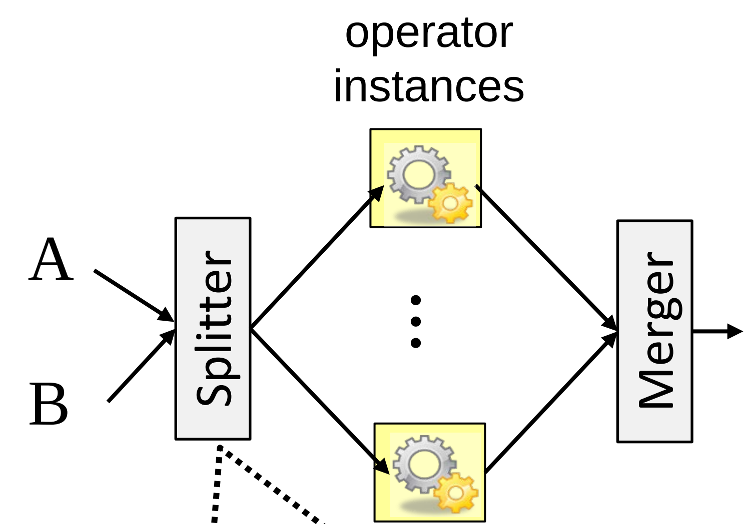
- Window-based query



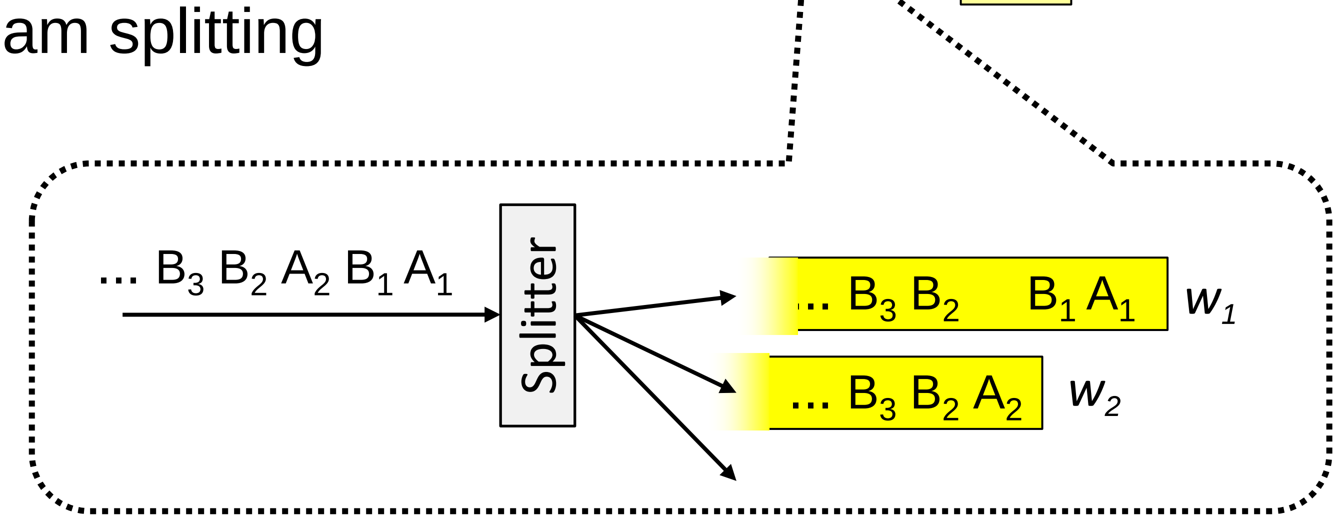
Operator Parallelization

- Data parallelization

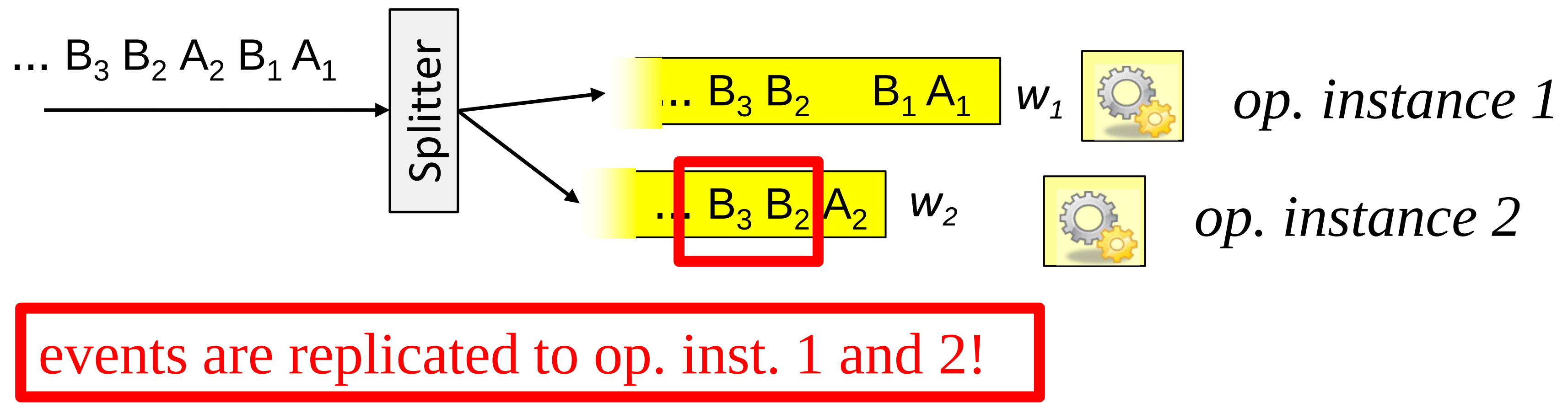
Query: Aperiodic(A; B; C) with
A → <type = requested_person,
time = t>
B → <type = face, "face_match(A)">
C → time ≥ t + time frame



- Window-based stream splitting
- Open window at request (A), close after 1 minute



Scheduling Problem



- Batching: Scheduling of subsequent overlapping window to the same operator instance
  → reduced network load
  → more computational load on single instances → higher latency

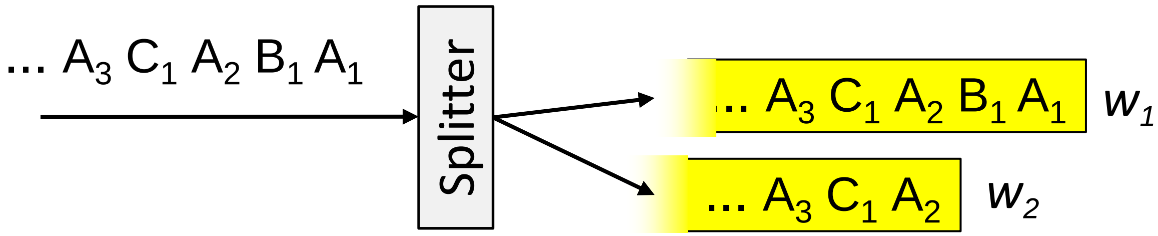
Scheduling problem: Maximize batching to an operator instance, while a given latency limit is still kept

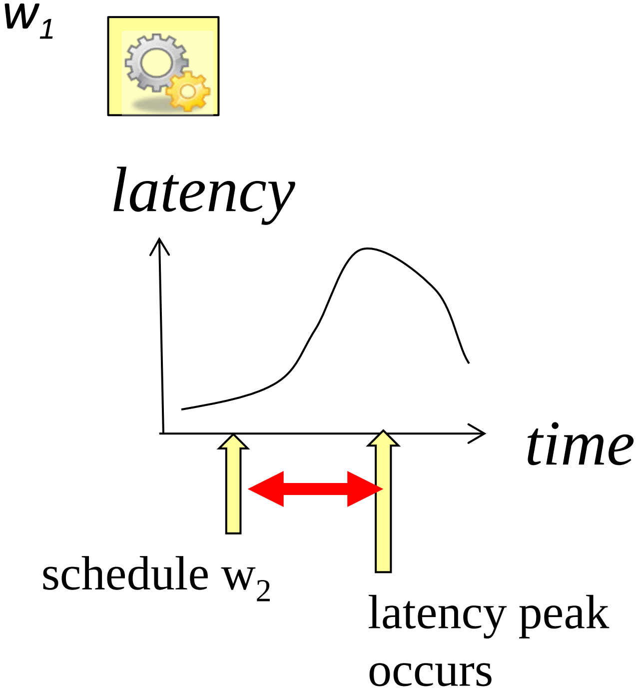
Contributions

- Problem analysis
  - Key factors of operator latency at overlapping windows
- Approach: model-based batch scheduling controller
  - Latency model
  - Scheduling algorithms
- Evaluations show efficacy and low overhead of the controller

Challenge

- Long feedback delays of control loop:





Challenges:

- Scheduling at window start
- Unknown events of the window at scheduling time

Implications

- Reactive scheduling?
  - *"Schedule **b** windows, measure latency peak, adapt scheduling"*
    - → State-of-the-art in stream processing
  - DCEP problem: schedule **open windows**
    - → Very long feedback delays to capture implications of scheduling
- Offline trained blackbox latency model?
  - The parameters are too many and the relation is complex
    - → Operator-specific
    - → Hard to predict outside of trained parameter ranges
- Our approach: Model-based scheduling controller

Approach Idea

Total operational latency of an event:
queueing + processing latency

$\lambda_o(e) = \lambda_q(e) + \lambda_p(e)$

→ queuing dominates operational latency
→ $\lambda_q(e)$ depends on λ_p and *iat* of previous events

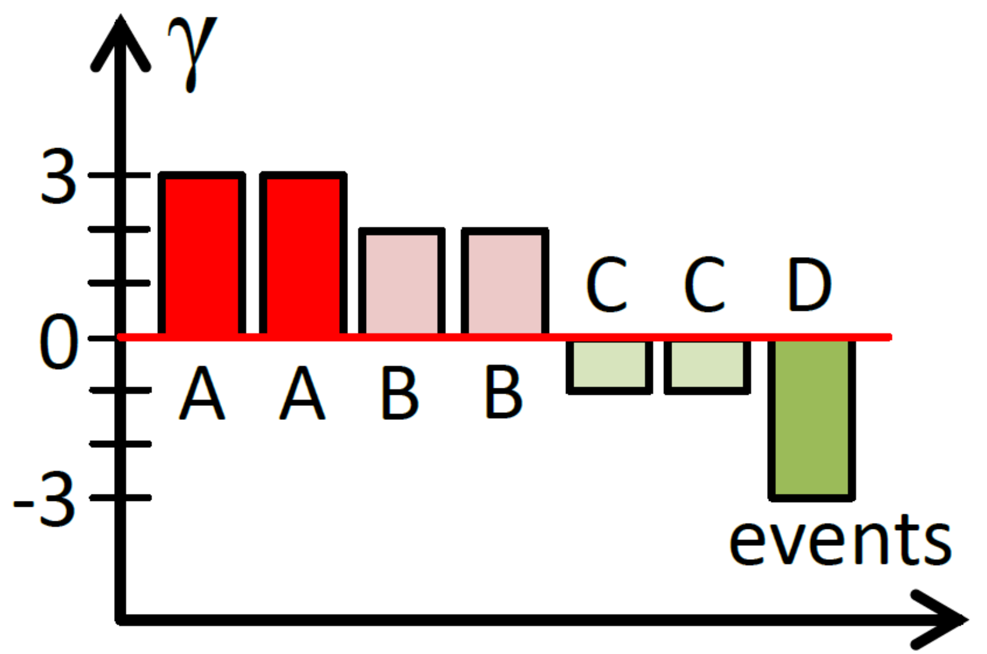
Idea: Predict the queuing latency peak

Approach:

1) Predict the set of events in w_{new}
2) Predict the impact of that set on the latency peak

Gain of Events

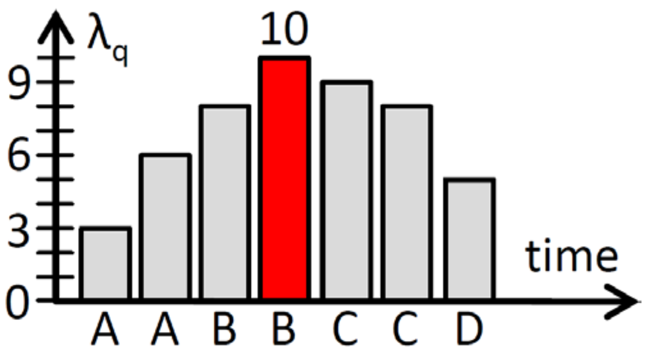
- Event e has processing latency λ_p and inter-arrival time *iat*
- If λ_p > *iat*, successor event has more queuing latency λ_q
- If λ_p < *iat*, successor event has smaller or zero λ_q
- Difference between λ_p and *iat* → gain: $\gamma(e) = \lambda_p(e) - iat$
- Example:

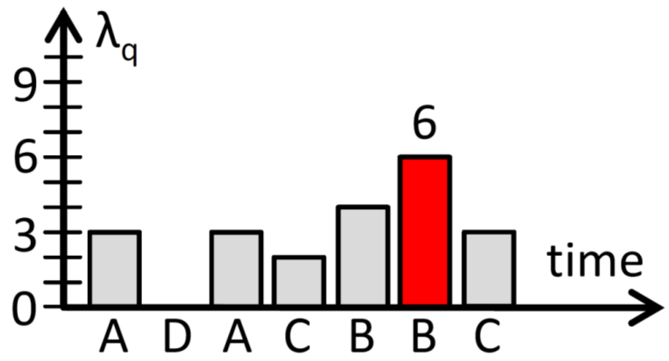


Sequence of Gains

Worst Case / Medium Case / Best Case

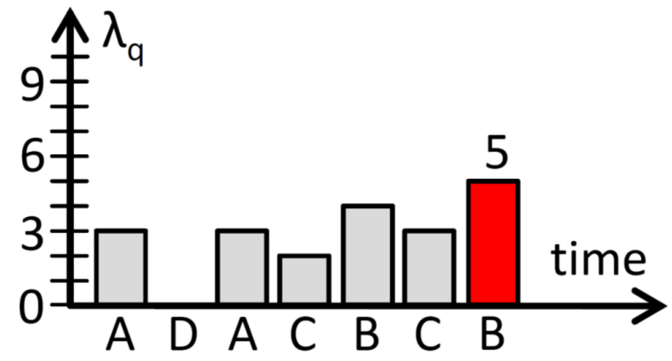
$\lambda_q^{max} = \Gamma^- = 10$ TU





$\lambda_q^{max} = \Gamma^- + 0.8 * \Gamma^+ = 6$ TU

$\lambda_q^{max} = \Gamma^- + \Gamma^+ = 5$ TU



- Generally: $\lambda_q^{max} = \Gamma^- + \alpha * \Gamma^+$, $\alpha \in [0, 1]$
  - α is termed the *compensation factor*
  - → the extent of interleaving of negative and positive gains

Predictions

Predict...

- ...total negative and positive gains

→ depends on events' processing latency and *iat* $\boxed{\gamma(e) = \lambda_p(e) - iat}$

- ...initial queuing latency

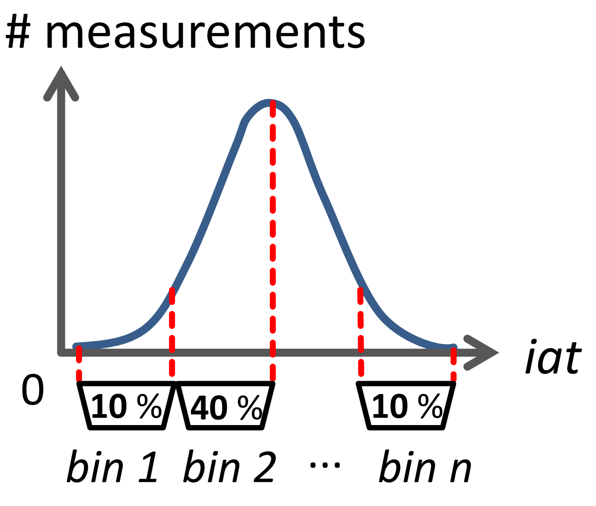
→ feedback from operator instances

- ...compensation factor

→ conservative heuristics, or expert knowledge

Inter-Arrival Time

Distribution of *iat* in n equally-sized bins



Processing Latency

- λ_p depends on overlap θ and proc. latency λ_p^w in single window

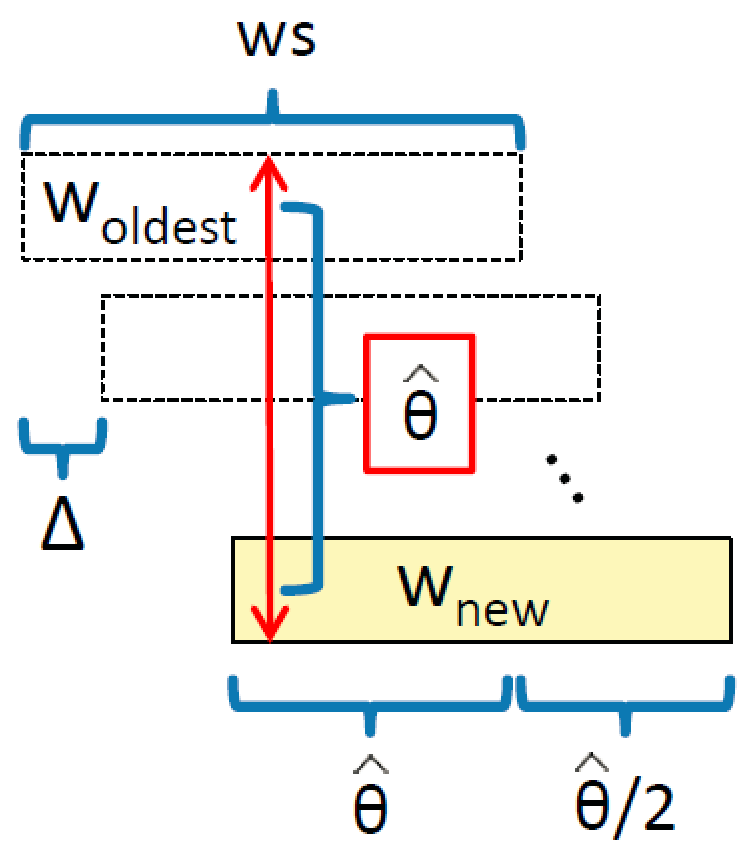
→ $\lambda_p = \theta * \lambda_p^w$

- λ_p^w depends on event type and position
    - Prediction of λ_p^w dependent on type
    - Prediction of set of events dependent on type

Overlap

- All events are predicted to have mean overlap $\bar{\theta}$
- Model: Weighted average based on current window shift Δ and window scope ws

$$\overline{\Theta} = \frac{\overbrace{(ws-(\hat{\Theta}-1)*\Delta)*\hat{\Theta}}^{\text{time while all windows are open}}+\overbrace{(\hat{\Theta}-1)*\Delta*\hat{\Theta}/2}^{\text{time while windows are closing}}}{ws}$$

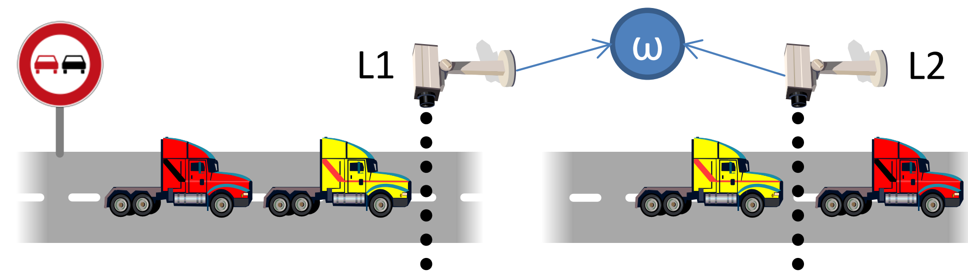


Evaluations: Setup

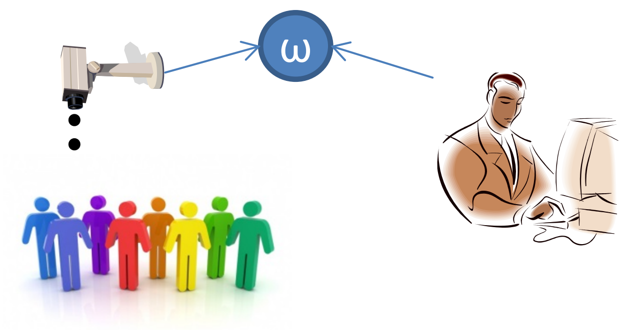
- We perform all experiments on a computing cluster consisting of 16 physical hosts with 8 cores
  - Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
  - 24 GB memory
  - 10-Gigabit-Ethernet connections
- Components of the data parallelization framework are evenly distributed among the available hosts

Evaluations: Scenarios

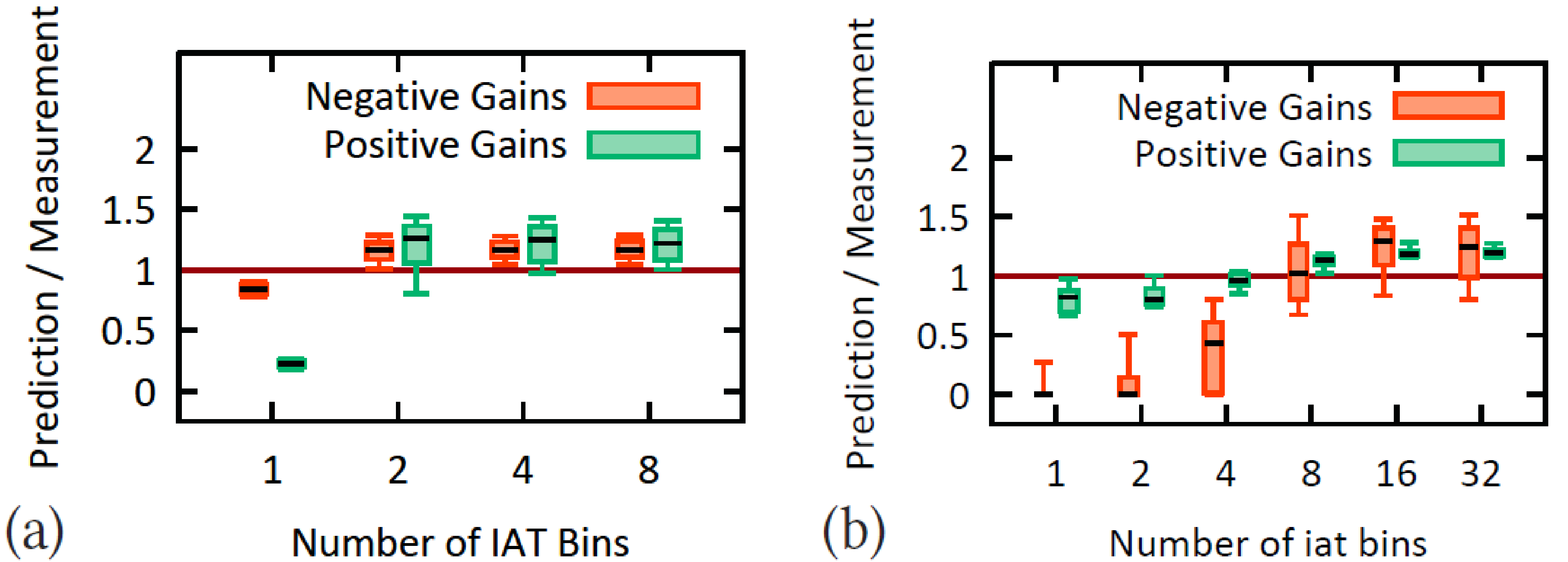
- Traffic monitoring operator
  - Overtaking detection



- Face recognition operator
  - Is a person of interest in the video stream?

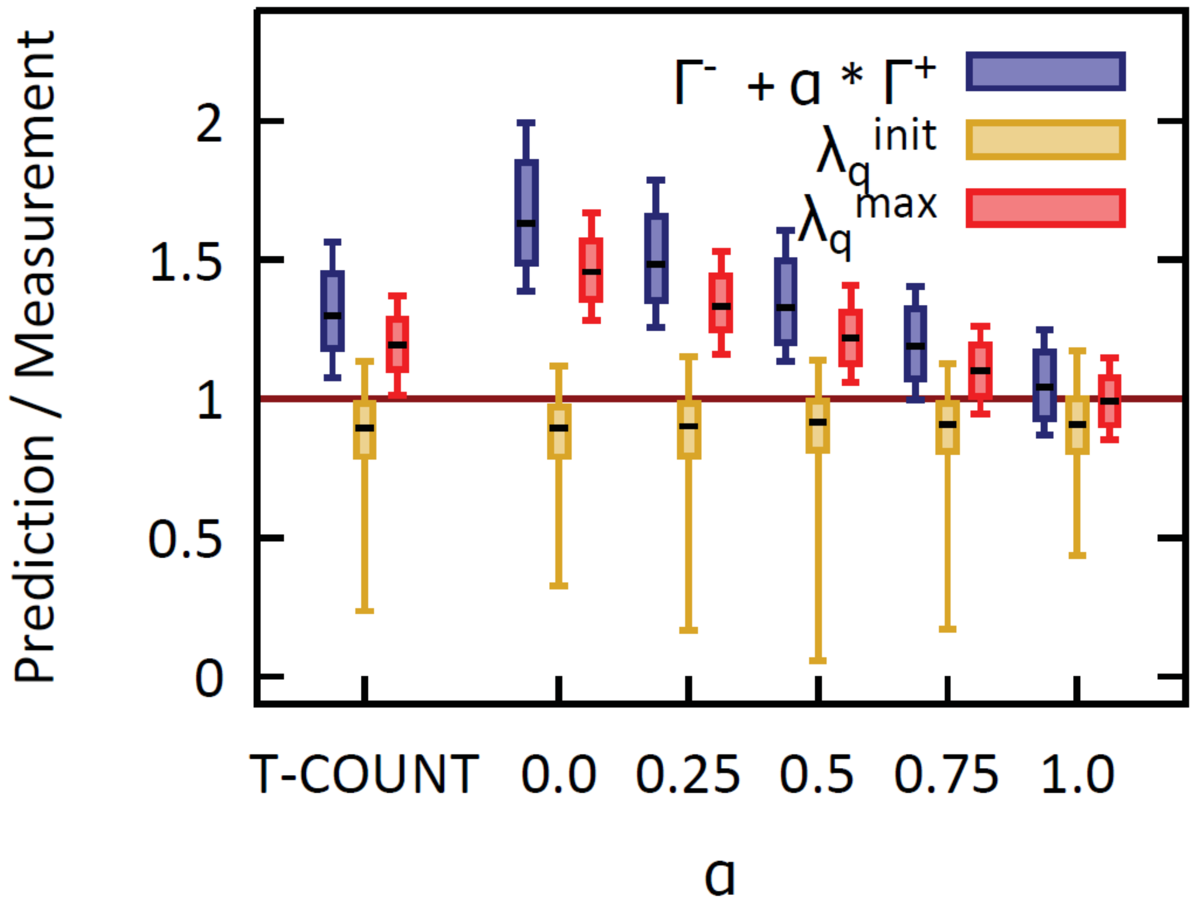


Evaluations: Negative and Positive Gains



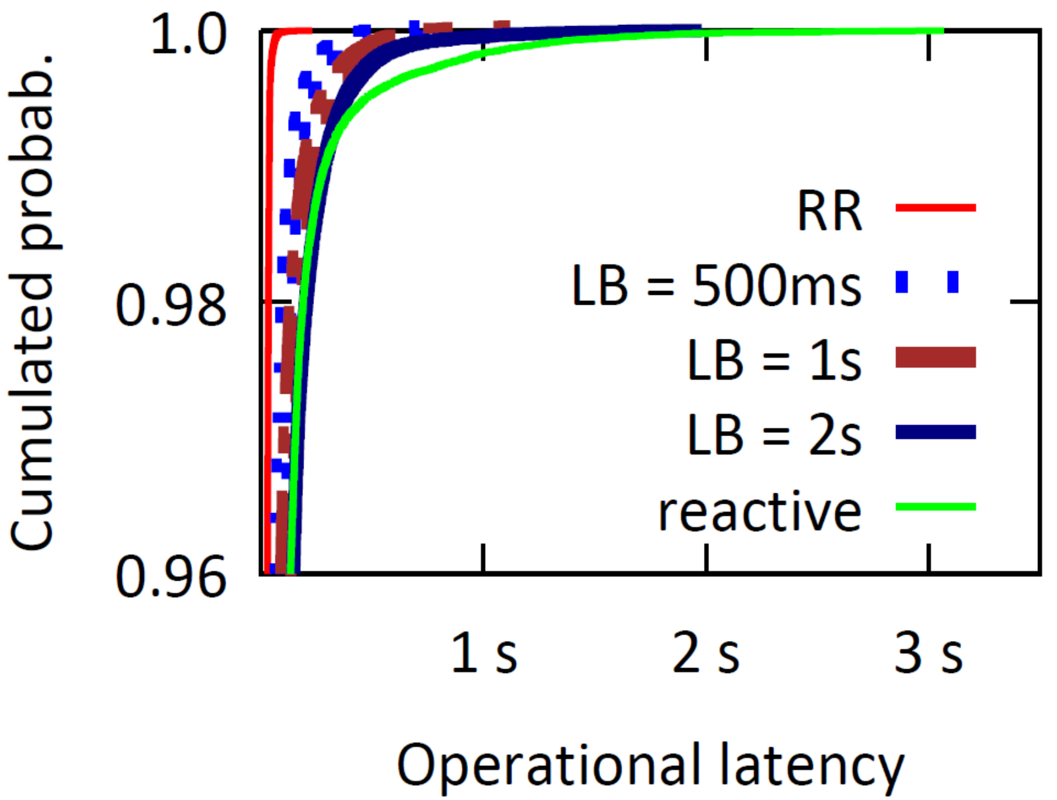
- Model is sufficiently accurate and precise

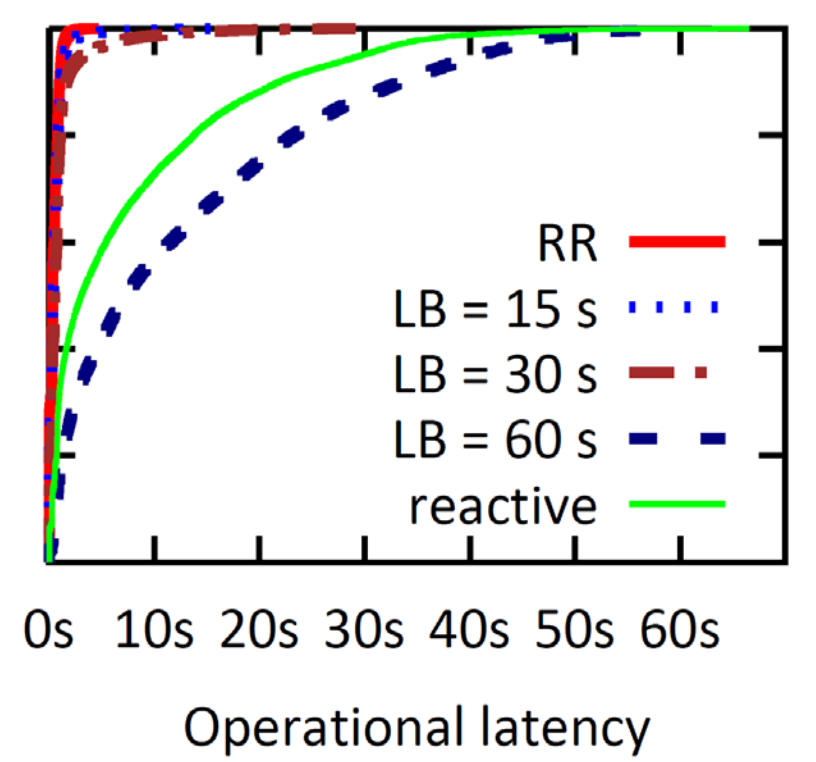
Evaluations: Compensation Factor



- Compensation factor can fine-tune the model

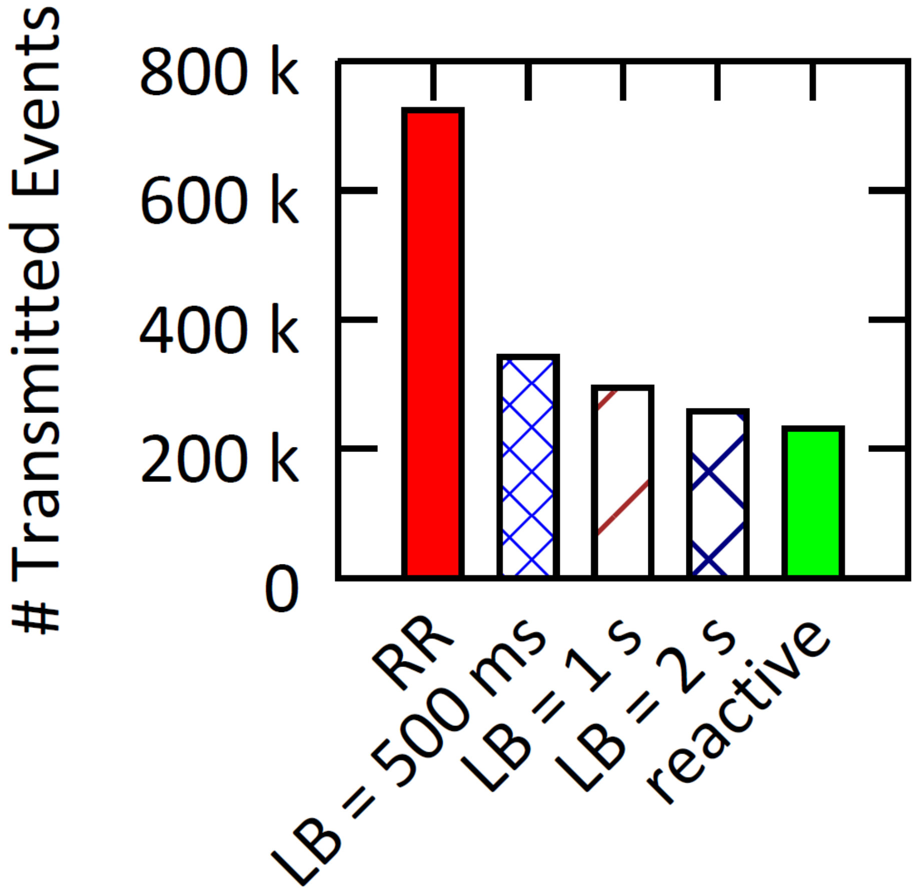
Evaluations

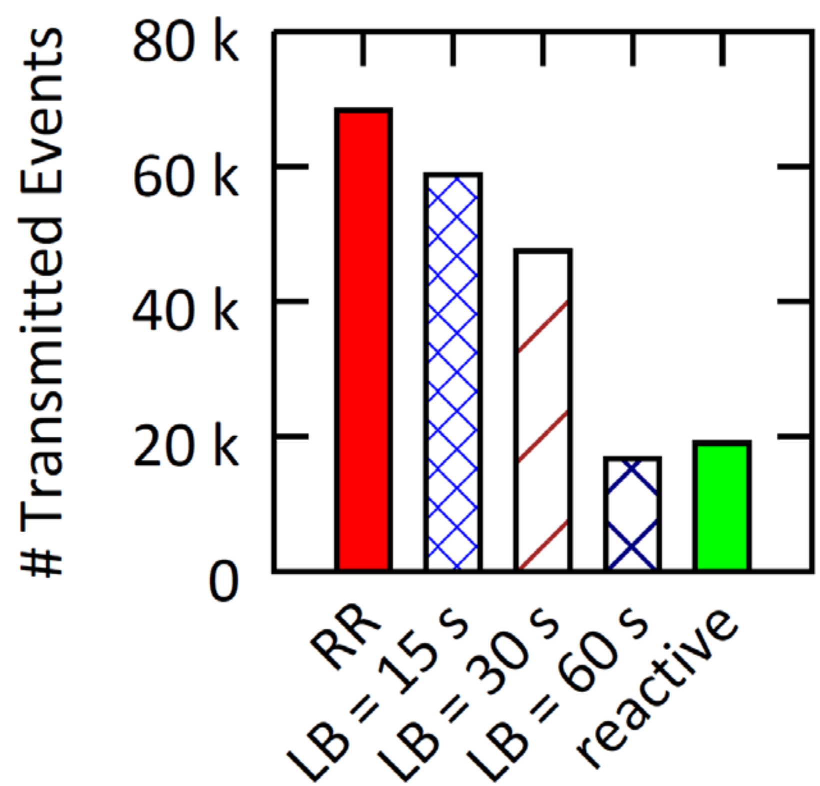




- Model-based controller keeps the latency bounds
- Reactive controller violates them

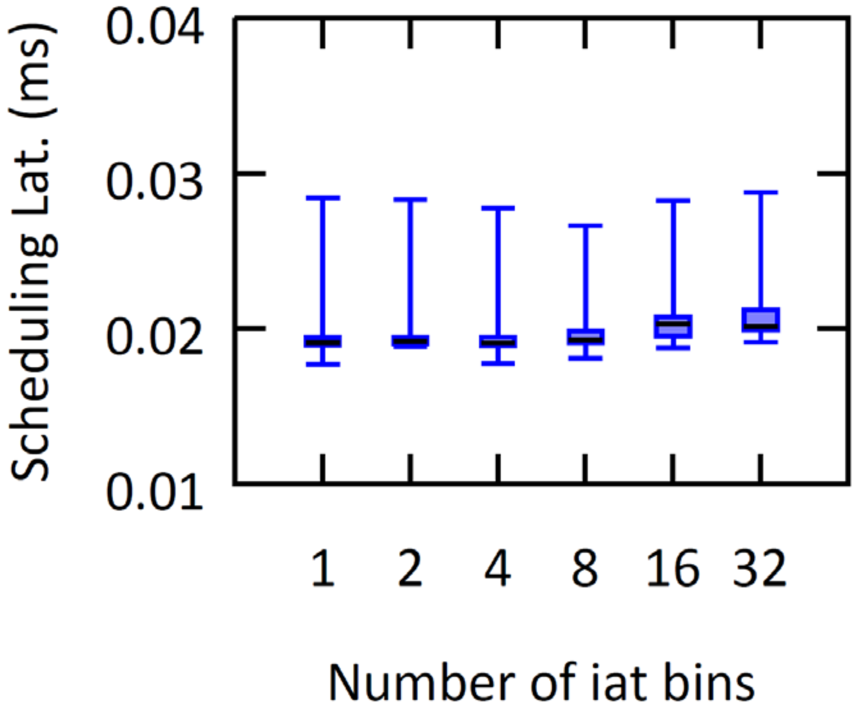
Evaluations: Communication Overhead

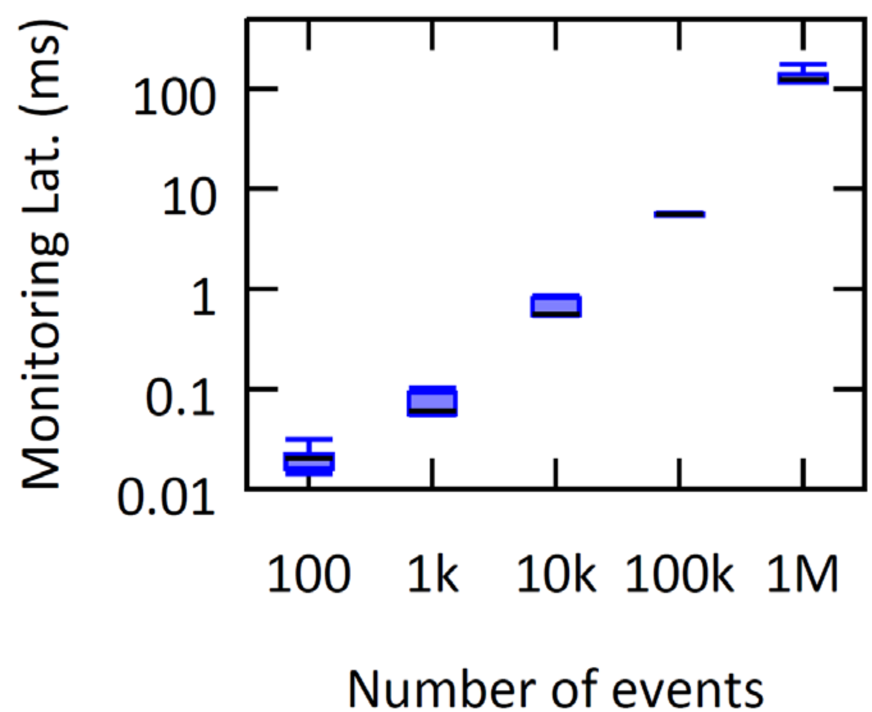




- Significant reduction of communication overhead!

Evaluations: Model Overhead





- Scheduling very fast → no bottleneck
- Updating model statistics reasonably fast

Conclusion

- Window-based data parallelization in DCEP poses scheduling tradeoff:
  - Replicate or batch an overlapping window?
- Trade-off is hard to control
- Approach: model-based batch scheduling controller
- Evaluations show the proposed approach saves bandwidth and keeps latency bounds

End of Presentation

Time for questions and answers.