Overload Management in Data Stream Processing Systems with Latency Guarantees

Evangelia Kalyvianaki[⋆], Themistoklis Charalambous[†],
Marco Fiscato[⋆], Peter Pietzuch[⋆]

[⋆]Imperial College London, Department of Computing
[†]Royal Institute of Technology (KTH), Sweden

Feedback Computing Workshop 2012

Data Stream Processing

Data streaming involves

Processing streams of data from distributed sources in near *real-time*

The *big data* era

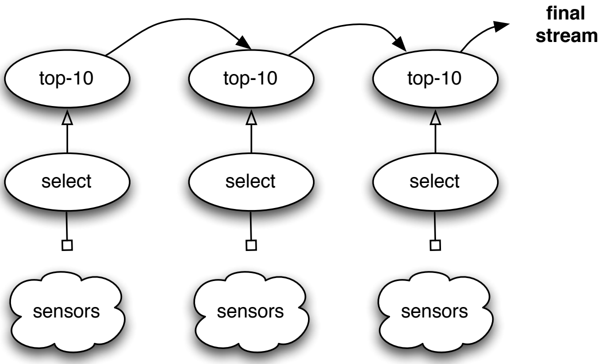
- Social networks, e.g. twitter and facebook
- Data-center monitoring, environmental sensing

Characteristics

- Large volume of data: e.g.
  340K tweets/day, 13K tweets/sec for popular tweets
- Workload fluctuations
- Results delivered at high throughput and low-latency

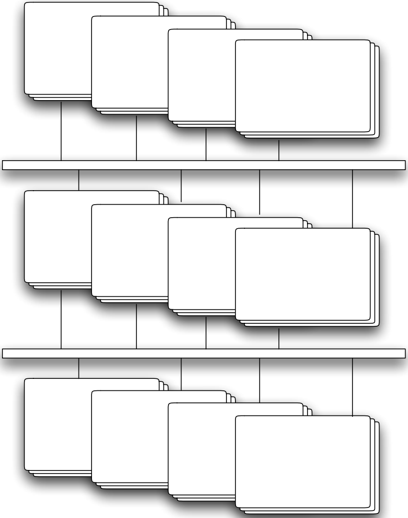
Data streaming query

What are the top-10 locations, in the city center, with the highest concentration in CO?



Distributed stream processing system (DSPS)

Set of hosts distributed in a data center

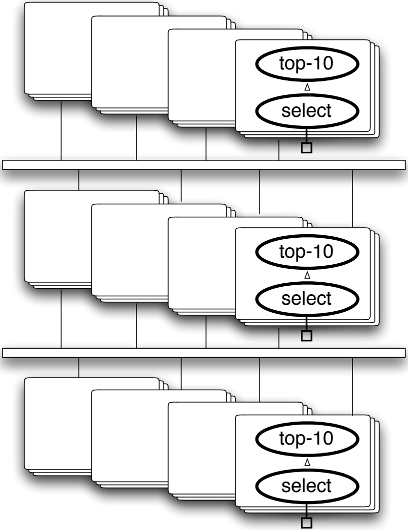


- Operator placement
- Resource allocation:
  CPU, net bandwidth

Today's large volumes of queries and workload data can exhaust resources and cause **overload**

Distributed stream processing system (DSPS)

Set of hosts distributed in a data center

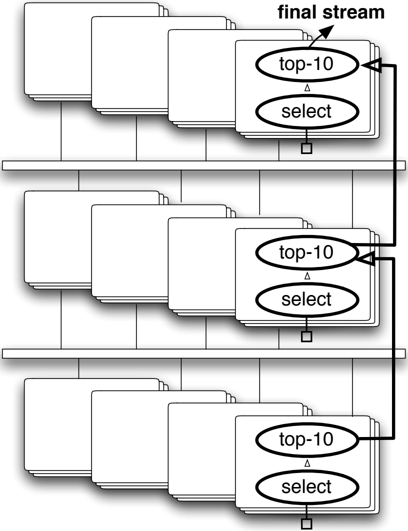


- Operator placement
- Resource allocation: CPU, net bandwidth

Today's large volumes of queries and workload data can exhaust resources and cause **overload**

Distributed stream processing system (DSPS)

Set of hosts distributed in a data center

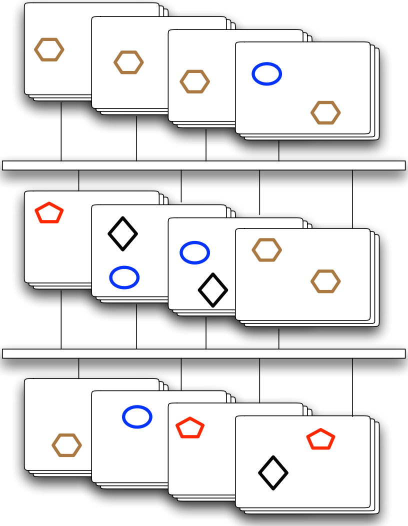


- Operator placement
- Resource allocation: CPU, net bandwidth

Today's large volumes of queries and workload data can exhaust resources and cause **overload**

Distributed stream processing system (DSPS)

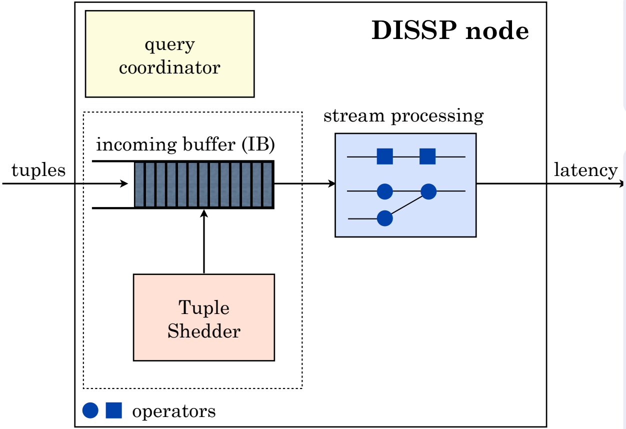
Set of hosts distributed in a data center



- Operator placement
- Resource allocation:
  CPU, net bandwidth

Today's large volumes of queries and workload data can exhaust resources and cause **overload**

Problem overview



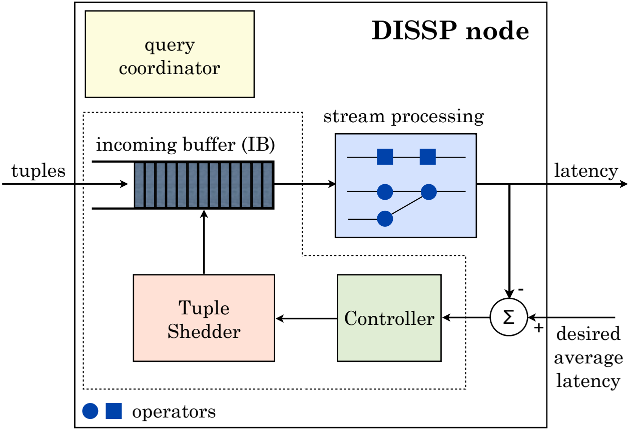
We **randomly** discard/shedd excess tuples

1: Results are meaningful after shedding, e.g. aggregates
2: Shedding feedback
3: Control tuple latency

Problem statement

The number of tuples to randomly discard from IB s.t. the end-to-end latency over time across tuples approaches a user-defined target

Problem overview



We **randomly** discard/shedd excess tuples

1: Results are meaningful after shedding, e.g. aggregates
2: Shedding feedback
3: Control tuple latency

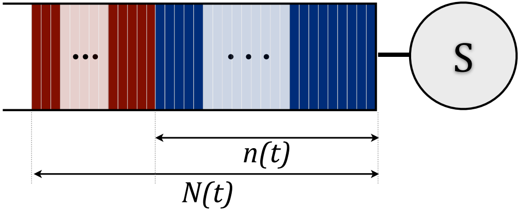
Solution overview

A feedback controller to randomly discard tuples and maintain end-to-end latency to a target value

Outline

1. Load shedding in data streaming — Motivation
2. Problem overview
3. Latency-based load shedding controller:
    - System model
    - Controller formulation
4. Experiment evaluation
    - Prototype DISSP deployment
5. Conclusions
6. Future research directions

System model



$c_s(t)$: tuple shedding cost, $c_p(t)$: tuple processing cost

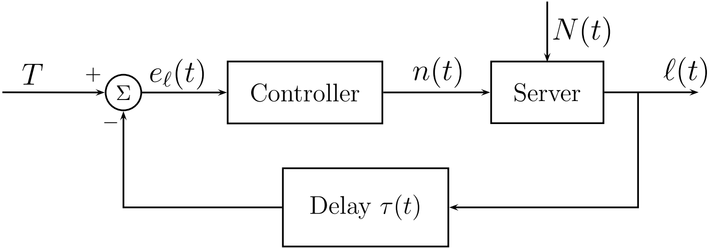
$$\text{tuple latency} \rightarrow \ell(t) \triangleq N(t)c_s(t) + n(t)c_p(t)$$

if T is the target latency, we seek $n^\star$ such that:

$$N(t)c_s(t) + n^\star(t)c_p(t) = T$$

Latency-based controller

$$n(t+1) = n(t) + pe(t - \tau(t)), \text{where: } e(t) \triangleq n^\star(t) - n(t)$$



$$n(t+1) = n(t) + qn(t)\frac{T - \ell(t - \tau(t))}{T}.$$

Evaluation

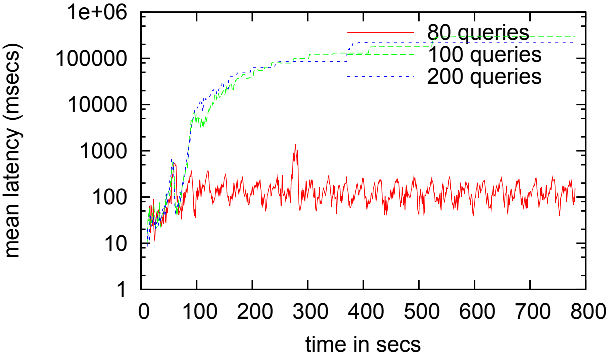
Workload

- Resource provisioning scenario over PlanetLab nodes
- Query: *what is the average CPU consumption every second over ten server machines from the PlanetLab network.*
- Ten source processes generate data from real-world traces of resource utilisations of PlanetLab nodes
- Time-varying tuple rate: 50t/s $\rightarrow$ 100t/s $\rightarrow$ 150t/s $\rightarrow$ 100t/s $\rightarrow$ 50t/s

Experimental setup

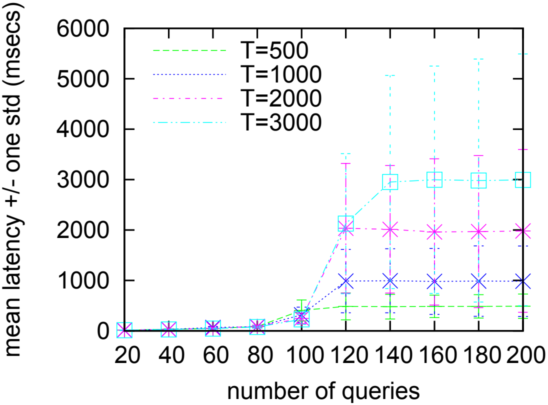
- Prototype single-node deployment of DISSP
- 2 server machines (4 CPU cores, 1.8 GHz, 4GB memory)

Queries performance without the controller



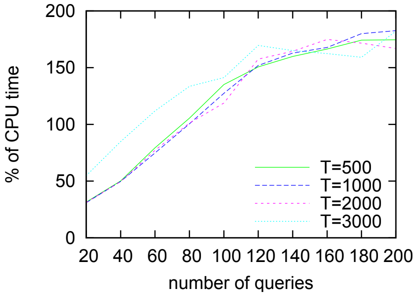
DISSP cannot run more that 80 queries

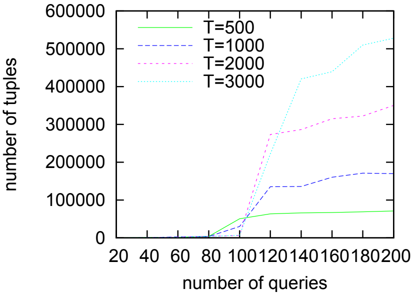
Queries performance with the controller



✓ mean tuple latency is controlled for various target values T

DISSP performance with the controller





- DISSP utilises a high % of CPU resources effectively
- For ↑ T, the controller keeps ↑ tuples

Overload management approaches

State-of-the-art

- Operator/stream re-use
- Query admission control
- Query rewriting
- e.g. System S from IBM
- Massively scalable systems, e.g. twitter Storm, Yahoo S4

Overload conditions still exist, e.g.

- Just before overload detection, during workload fluctuations
- Utilisation of load shedding to reduce additional costs

Related work

"Load Shedding in Stream Databases: A Control-Based Approach", by Tu et al, in VLDB 2006

Future Research Directions

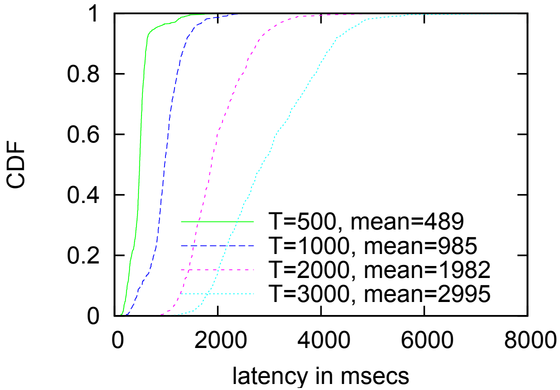
- Heterogeneous workload with various queries
- Distributed stream processing systems
- Semantic shedding to minimise application performance loss
- Explore feedback control methods for the big data management

Conclusions

- Controller for overload in data stream processing systems
- Latency-based controller
- Single-node evaluation on homogeneous workload
- Results show that the avg latency can be controlled effectively

Thank you! Any questions?
ekalyv@doc.ic.ac.uk
http://lsds.doc.ic.ac.uk/research

Queries latency performance



- mean latency is very close to target T
- latency variation remains small

Controller formulation

Equations:
$n(t+1) = n(t) + pe(t - \tau(t))$
$e(t) \triangleq n^\star(t) - n(t)$
$N(t)c_s(t) + n^\star(t)c_p(t) = T$
give

$$n(t+1) = n(t) + p\frac{T - \ell(t-\tau(t))}{c_s(t-\tau(t)) + c_p(t-\tau(t))}$$

$$n(t+1) = n(t) + qn(t)\frac{T - \ell(t-\tau(t))}{T},$$
we choose a small control gain q for controller's stability